

On The Application of Neural Networks to Symbol Systems

by Simon Davidson

submitted for a PhD

in the Dept. of Electronic and Electrical Engineering,

at the University of Sheffield

November, 1999

Abstract

While for many years two alternative approaches to building intelligent systems, symbolic AI and neural networks, have each demonstrated specific advantages and also revealed specific weaknesses, in recent years a number of researchers have sought methods of combining the two into a unified methodology which embodies the benefits of each while attenuating the disadvantages.

This work sets out to identify the key ideas from each discipline and combine them into an architecture which would be practically scalable for very large network applications. The architecture is based on a relational database structure and forms the environment for an investigation into the necessary properties of a symbol encoding which will permit the single-presentation learning of patterns and associations, the development of categories and features leading to robust generalisation and the seamless integration of a range of memory persistencies from short to long term.

It is argued that if, as proposed by many proponents of symbolic AI, the symbol encoding must be causally related to its syntactic meaning, then it must also be mutable as the network learns and grows, adapting to the growing complexity of the relationships in which it is instantiated. Furthermore, it is argued that in order to create an efficient and coherent memory structure, the symbolic encoding itself must have an underlying structure which is not accessible symbolically; this structure would provide the framework permitting structurally sensitive processes to act upon symbols without explicit reference to their content. Such a structure must dictate how new symbols are created during normal operation.

The network implementation proposed is based on K-from-N codes, which are shown to possess a number of desirable qualities and are well matched to the requirements of the symbol encoding. Several networks are developed and analysed to exploit these codes, based around a recurrent version of the non-holographic associative memory of Willshaw, et al. The simplest network is shown to have properties similar to those of a Hopfield network, but the storage capacity is shown to be greater, though at a cost of lower signal to noise ratio.

Subsequent network additions break each K-from-N pattern into L subsets, each using D-from-N coding, creating cyclic patterns of period L. This step increases the capacity still further but at a cost of lower signal to noise ratio. The use of the network in associating pairs of input patterns with any given output pattern, an architectural requirement, is verified.

The use of complex synaptic junctions is investigated as a means to increase storage capacity, to address the stability-plasticity dilemma and to implement the hierarchical aspects of the symbol encoding defined in the architecture. A wide range of options is developed which allow a number of key global parameters to be traded-off. One scheme is analysed and simulated.

A final section examines some of the elements that need to be added to our current understanding of neural network-based reasoning systems to make general purpose intelligent systems possible. It is argued that the sections of this work represent pieces of the whole in this regard and that their integration will provide a sound basis for making such systems a reality.

Table of Contents

CHAPTER 1 On the Application of Neural Networks to Symbol Systems	1
1.0 The Potential to be Exploited	1
1.1 Of Symbols & Synapses	2
1.2 Underlying Philosophy of the Work	6
1.3 Objectives for the Work	8
1.4 The Structure of the Thesis	10
1.5 Summary of the Work	11
CHAPTER 2 Surveying The Field.....	13
2.0 Introduction	13
2.1. Neuroscience	14
2.2 Artificial Intelligence	18
2.3 Artificial Neural Networks	28
2.4 Artificial Intelligence & Neural Networks	38
2.5 Conclusions	44
CHAPTER 3 Detailed Review I: Coding and Storage Issues.....	46
3.0 Introduction	46
3.1 Connectionist Associative Memories and Coding Issues.....	47
3.2 Recurrent Memories	60
3.3 Coding and Representation of Symbol Structures.....	67
3.4 Conclusions	84
CHAPTER 4 Detailed Review II: Reasoning Systems.....	85
4.0 Introduction	85
4.1 Production Systems	86
4.2 Semantic Networks and Inheritance Issues	92
4.3 Marker Passing Inference Architectures	97
4.4 Other Important Work	108
4.5 Summary of Reasoning Systems and Discussion.....	115
4.6 Conclusions	117
CHAPTER 5 Architecture Design: Data Representation & Manipulation.....	118
5.0 Introduction	118
5.1 Basic System Architecture	119
5.2 Knowledge Bases	120
5.3 Data and Control.....	121
5.4 The Symbolic Principle	123
5.5 Rejected Candidate Symbol Encodings.....	131
5.6 Development of a Robust and Flexible Symbol Encoding	135
5.7 Look-up versus Computation	135
5.8 Hierarchies of Knowledge	138

5.9 Hierarchies of Learning	142
5.10 Symbol As Representative for a Constituent Structure	146
5.11 Unsupervised Development of Symbol Encoding	147
5.12 Choice of Symbol Encoding.....	149
5.13 Detailed Architecture Structure	150
5.14 Conclusions	154
CHAPTER 6 Foundations for the Neural Building Block	155
6.0 Laying the Foundations	155
6.1 Recurrent Network	156
6.2 Replicable Neural “Unit”	156
6.3 The Data Coding Scheme.....	157
6.4 Properties of K-from-N Codes	163
6.5 Consequences of Commitment to Redundancy.....	168
6.6 Conclusions	171
Appendix: Counting the Size of a ‘Neighbourhood’	172
CHAPTER 7 Analysis of Storage and Retrieval for the Simple Network	174
7.0 Introduction	174
7.1 The Patterns to be Stored.....	174
7.2 Network Output Updating	175
7.3 Storing Patterns	177
7.4 Weight Matrix Saturation	179
7.5 The Process of Retrieval: Network Dynamics	182
7.6 Analysis of Memory Capacity	186
7.7 Energy Landscape	199
7.8 Simulation of the Simple Network	200
7.9 Discussion.....	206
7.10 Conclusions	206
7.11 Appendix: Graphs from Simulations.....	208
CHAPTER 8 Dynamic Patterns.....	214
8.0 Introduction	214
8.1 The Idea Behind Dynamic Patterns	215
8.2 Information Stored in Each Pattern	217
8.3 Alternative Implementations	219
8.4 Learning Dynamic Patterns	225
8.5 Analysis of Storage Capacity for Dynamic Patterns	227
8.6 Case Studies & Simulations	233
8.7 Comparison of Static and Dynamic Storage	235
8.8 Conclusions	239
8.9 Appendix: Graphs of Simulation Results	240
CHAPTER 9 Pattern Association and Network Control	243
9.0 Introduction	243
9.1 Symbolic View of Network Control.....	243
9.2 Energy Landscape of Network Control.....	244
9.3 ‘Dual Pattern’ View of Control	246

9.4 Implementation of Pattern Association	248
9.5 Analysis using Dynamic Pattern Learning	249
9.6 Simulation and Results	251
9.7 Discussion.....	252
9.8 Conclusions	253
CHAPTER 10 Learning Strategies.....	254
10.0 Introduction	254
10.1 Learning Hierarchies: Aims and Issues	255
10.2 Complex Synaptic Structure.....	258
10.3 Options for the Weight Vector Components	263
10.4 Implications of Creating Correlation Between Neurons	269
10.5 Options for Component Interdependence During Learning.....	271
10.6 A Note on the Generation of New Symbols.....	275
10.7 Investigation of One Strategy for Learning.....	276
10.8 Simulations of Simple Learning Scheme	284
10.9 More Complex Learning Schemes	286
10.10 More Complex Synaptic Structure	288
10.11 Memory Capacity of a Network Using Learning Hierarchies.....	290
10.12 Discussion.....	290
10.13 Conclusions	291
CHAPTER 11 Towards True Neuro-Symbolic Computation	292
11.0 Introduction	292
11.1 Barriers to the Connectionist Dream	293
11.2 Issues in Hierarchy and the Limits of Real Components	302
11.3 On Escaping the Bottleneck of Forward-chaining Rules	310
11.4 On Parallel Processing at the Symbolic Level	319
11.5 Towards a Unified Neuro-Symbolic Theory	332
CHAPTER 12 Conclusions & Future Work.....	334
12.0 Introduction	334
12.1 Comparison of Requirements and Results	334
12.2 Future Work.....	338
12.3 Future Architectural Expansion.....	341
12.4 Conclusions	344

List of Figures

CHAPTER 1 On the Application of Neural Networks to Symbol Systems	1
CHAPTER 2 Surveying The Field.....	13
Fig. 2-0 One possible explanation of information flow in the neocortex.	16
Fig. 2-1 An example production system.	21
Fig. 2-2 An example semantic network.	23
Fig. 2-3 Multi-layered Perceptron, with recurrent output.....	29
Fig. 2-4 Kohonen network: Lateral feedback vs. distance from winning neuron	32
Fig. 2-5 Single pattern learning in the non-holographic associative memory.	35
Fig. 2-6 Fodor & Pylyshyn’s argument for Coherence of Inference.	42
CHAPTER 3 Detailed Review I: Coding and Storage Issues.....	46
Fig. 3- 0 Storage density vs. proportion of active synapses, Nadal & Toulouse (1990).....	52
Fig. 3- 1 Cascade Associative Memory (CASM), Hirahara et al. (1997).....	62
Fig. 3-2 A syntactic tree with branches of different depths.....	68
Fig. 3- 3 Defining relationships between two objects. From Hinton (1990).	72
Fig. 3- 4 Between-level time-sharing. From Hinton (1990).	73
Fig. 3-5 Trre structure with roles and fillers.....	74
Fig. 3- 6 Symbolic Compression Scheme. From Pollack (1990).	76
Fig. 3- 7 A simple embedded tree.....	77
Fig. 3- 8 Chalmers’ syntactic transform model.	79
Fig. 3- 9 Chrisman’s confluent transformation model.....	80
Fig. 3- 10 Adamson and Damper’s extension of RDR.	81
CHAPTER 4 Detailed Review II: Reasoning Systems.....	85
Fig. 4- 0 Trade-offs in search and preparation (from Newell, 1990).....	87
Fig. 4- 1 The Distributed Connectionist Production System (DCPS).	89
Fig. 4- 2 An example of the problem of redundancy in semantic networks.....	93
Fig. 4- 3 A problem of semantics: The Nixon diamond.	94
Fig. 4- 4 Exception inheritance reasoning (Al-Asady, 1995).	95
Fig. 4- 5 A fraction of Lange & Dyers’ inference network	98
Fig. 4- 6 An example of a predicate and a fact in SHRUTI	100
Fig. 4- 7 An example network from SHRUTI.....	101
Fig. 4- 8 An assembly from CONSYDERR	104
Fig. 4- 9 Example vectors and tensor for AURA, from Austin (1995)	108
CHAPTER 5 Architecture Design: Data Representation & Manipulation.....	118
Fig. 5-0 High-level System Architecture	119
Fig. 5-1 A section of a relational database.....	120
Fig. 5-2 Flow of activity from control to data networks during processing.	122
Fig. 5-3 Hierarchical decomposition of a symbol “Sentence”.....	123
Fig. 5-4 Simple example of the use of symbols in mathematics	127
Fig. 5-5 Pre- and post-processing steps would permit arbitrary encoding	129
Fig. 5-6 A symbol, X, acts as both a source and a target in many processes	131

Fig. 5-7 The single pointer, encoded-usage scheme of symbol encoding.	132
Fig. 5-8 Double pointer scheme allocates one pointer to each type of information.	133
Fig. 5-9 The triple pointer scheme with inheritance.	134
Fig. 5-10 A purely look-up table approach to processing.	136
Fig. 5-11 Creation of a category whose members share a common property.	138
Fig. 5-12 Data structure with Inheritance.	139
Fig. 5-13 Usage pointer for two symbols.	140
Fig. 5-14 Direct association of two symbols.	143
Fig. 5-15 Indirect association of symbols using feature extraction	144
Fig. 5-16 Learning hierarchies allow mixtures of features of different levels	145
Fig. 5-17 Adjustment of feature detectors that supersede direct (temporary) links .	145
Fig. 5-18 Every symbol can embody multiple symbols of equal complexity.	147
Fig. 5-19 Each new association links modified versions of core symbols	148
Fig. 5-20 A more detailed schematic for the symbol processing architecture.	151
Fig. 5-21 Symbol D created to represent relationship between A, B and C	152
Fig. 5-22 Unpacking a symbol to access content information.	153
CHAPTER 6 Foundations for the Neural Building Block	155
Fig. 6-0 Example of the K-from-N code illustrating its robustness.	161
Fig. 6-1 Graphs of Information Density vs. Fraction of Firing Neurons.	164
Fig. 6-2 Legal K-from-N vectors with redundancy.	165
Fig. 6-3 Target circles for convergence: initial and final	169
Fig. 6-4 Patterns obtained by ‘shifting’ from a base pattern.	172
Fig. 6-5 A vector at shift k can have multiple parents at shift k-1.	173
CHAPTER 7 Analysis of Storage and Retrieval for the Simple Network	174
Fig. 7-0 Basic network showing N fully connected elements and external input ...	175
Fig. 7-1 Graph of potentials for the neurons in a single region	176
Fig. 7-2 The learning process in the basic network	178
Fig. 7-3 Graph of connection probability vs number of stored patterns.	181
Fig. 7-4 Potential distributions for neurons in known pattern state.	187
Fig. 7-5 Probability that neuron that should be silent might fire	188
Fig. 7-6 Graph of expected errors per pattern vs. the number of stored patterns	189
Fig. 7-7 Graph of total expected errors vs. number of stored patterns.	190
Fig. 7-8 Example 5-from-10 vector at several levels of corruption.	191
Fig. 7-9 Potential distribution for neurons in a non-learned state	191
Fig. 7-10 Distribution of potential difference between firing and non-firing neurons	193
Fig. 7-11 Graph of error probability for two neurons.	194
Fig. 7-12 Graphs of the total expected error vs. number of stored patterns.	195
Fig. 7-13 Graph: expected max. stored patterns vs. # of firing neurons for N=200.	196
Fig. 7-14 Total information stored for perfect recall in an N=200 network.	197
Fig. 7-15 An energy landscape for the simple network.	199
Fig. 7-16 Graph of max patterns stored without error vs. size of basin of attraction	205
CHAPTER 8 Dynamic Patterns.....	214
Fig. 8-0 State flow for a fixed-point attractor and a limit cycle.	215
Fig. 8-1 Transition between two dynamic patterns.	216
Fig. 8-2 One dynamic pattern cycle, made up of L subsets, describes pattern p.	217
Fig. 8-3 Graph of information content for one pattern vs. number of firing neurons.	218
Fig. 8-4 Feedforward connections one subset to the next during learning.	219

Fig. 8-5 State space showing the trajectories of two pattern, p0 and p1.	220
Fig. 8-6 Possible connection schemes over multiple time steps.....	221
Fig. 8-7 External input to a region should cause changes in the output trajectory...223	
Fig. 8-8 Input specifies point of entry into recall loop.	224
Fig. 8-9 Alternative stimulation scheme with constant input.	224
Fig. 8-10 Output trajectories of two regions, with B receiving input from A.	225
Fig. 8-11 Timelines neuron potential showing influence of refractory period.....	229
Fig. 8-12 Potential distributions for two neurons.	232
Fig. 8-13 Max stored patterns vs. size of firing subset, from burst mode simulation.237	
Fig. 8-14 Total storage capacity (bits) vs. number of firing neurons per subset.	237
CHAPTER 9 Pattern Association and Network Control	243
Fig. 9-0 Changes in the energy landscape due to changing control input.	245
Fig. 9-1 An implementation of control by association.	248
Fig. 9-2 Graph of recall errors vs. storage level for network with control.	251
CHAPTER 10 Learning Strategies.....	254
Fig. 10-0 Traditional hierarchical neural network such as an MLP.	256
Fig. 10-1 Modification of a function over time to incorporate one new point.	257
Fig. 10-2 Diagram of the complex synapse studied in this chapter.....	259
Fig. 10-3 Graphs of synaptic potential against time for two combiners.....	262
Fig. 10-4 Weight vector for neuron i in terms of components for each synaptic unit.263	
Fig. 10-5 Output history vectors as seen by several different synaptic units.	264
Fig. 10-6 Weight vector components with restrictions on development.	267
Fig. 10-7 Desired distribution of weights of different synaptic units for each neuron.269	
Fig. 10-8 Modifications to the pattern during consolidation make it easier to store.270	
Fig. 10-9 Synaptic unit optimisation using ordered sequential dependence.....	272
Fig. 10-10 Synaptic unit optimisation using ordered, non-sequential dependence. .274	
Fig. 10-11 Movement of the hierarchical weight vector during learning.	279
Fig. 10-12 Histograms of neuron potentials (a) before and (b) after consolidation. 280	
Fig. 10-13 Potential bands in a more advanced learning scheme.....	287
CHAPTER 11 Towards True Neuro-Symbolic Computation	292
Fig. 11-0 Desired mode of computation in a neuro-symbolic processor.....	297
Fig. 11-1 Control of data network using subtraction.	300
Fig. 11-2 The ideal module.....	304
Fig. 11-3 Hierarchical Network with Limited Fan-in Neurons	306
Fig. 11-4 Collapsing the Hierarchy to Preserve Generality.....	307
Fig. 11-5 Conjunctive and disjunctive reasoning	321
Fig. 11-6 Combining multiple K-from-N vectors.....	326
CHAPTER 12 Conclusions & Future Work.....	334
Fig. 12-0 Dividing the network into multiple regions.	340

List of Tables

CHAPTER 1 On the Application of Neural Networks to Symbol Systems	1
CHAPTER 2 Surveying The Field.....	13
CHAPTER 3 Detailed Review I: Coding and Storage Issues.....	46
Table 3-0 Comparison of Network Storage Capacities in the noiseless case. From Casasent & Telfer (1991)	55
Table 3-1 Stored patterns per neuron, M/N, for several network types and several levels of noise during recall. From Casasent & Telfer (1991)	55
Table 3-2 Storage capacity, M/N, for different output encoding schemes and different recall accuracies, P'c	56
CHAPTER 4 Detailed Review II: Reasoning Systems.....	85
CHAPTER 5 Architecture Design: Data Representation & Manipulation.....	118
CHAPTER 6 Foundations for the Neural Building Block	155
Table 6-0 Comparison of three data coding schemes	159
Table 6-1 Efficiency of information storage for different values of N	160
Table 6-2 Measure of neighbourhood size, no. of golden vectors and storage efficiency for several network configurations.....	168
CHAPTER 7 Analysis of Storage and Retrieval for the Simple Network	174
Table 7-0 Comparison of simulated vs. theoretical network capacity of the simple network	204
CHAPTER 8 Dynamic Patterns.....	214
Table 8-0 Comparison of information storage for static and dynamic patterns (in burst mode), with N = 300 and F = 1.	236
Table 8-1 Efficiency of a number of networks, in descending order of efficiency.....	242
CHAPTER 9 Pattern Association and Network Control	243
CHAPTER 10 Learning Strategies.....	254
Table 10-0 Relative weight vector component dominance for different synaptic levels.....	265
Table 10-1 Modification of It product for level d leads to reduction in max. product size. ..	266
CHAPTER 11 Towards True Neuro-Symbolic Computation	292
CHAPTER 12 Conclusions & Future Work.....	334

List of Symbols

Chapter Six: Foundations for the Neural Building Block

- t: time.
N: number of neurons in the network.
p: fraction of neurons in a pattern that are firing (value '1').
K: number of neurons in a pattern that are firing = pN.
 $p(x)$: probability distribution of variable x.
I(N): Information content (in bits) for N-bit vector.
 $E_{\max}(N)$: Storage efficiency (or storage density) for N-bit vector.
S: Signal strength of an N-bit vector.
 Φ_i : i -th golden vector.
 $d(x, \Phi_i)$: distance function between vectors x and Φ_i .
 $V(K, N)$: number of legal K-from-N vectors.
 $Z(M, K, N)$: number of legal K-from-N vectors within distance M of a golden vector.
 $G(M, K, N)$: maximum number of golden K-from-N vectors.
d: max. distance of final pattern from target pattern after convergence.
e: max. distance of initial pattern from target pattern before convergence.

Chapter Seven: Analysis of Storage and Retrieval for the Simple Network

- V_1, V_2 : Arbitrary vectors.
u: scalar product of two vectors.
 W_{ij} : Weight from output of neuron j to input of neuron i.
 $X_i(t)$: Input to neuron i at time t.
 $U_i(t)$: Potential of neuron i at time t.
 $O_i(t)$: Output of neuron i at time t.
 T_i : Threshold of neuron i.
V: Activity regulator, ensures that only K neurons fire.
B: Connection strength between two neurons that make a connection.
Z: number of learned patterns.
h: probability of a connection between two neurons.
 E_x : Energy of the system in timestep x .
 E_{diff} : Difference in energy between two states.
 $I_{\max}(M, K, N)$: Maximum information content in K-from-N golden vectors at separation 2M.
 $Z_{\max}(M, K, N)$: Maximum number of stored K-from-N vectors.
 $C(M, K, N)$: Maximum information content of network defined by parameters K, N and M.
 U_f and U_{nf} : Expected potential of firing and non-firing neurons, respectively.
 σ_f and σ_{nf} : Standard deviation of potential for firing and non-firing potential, respectively.
q: Probability that neuron has a given potential.
 \mathbf{G}_f : Set of firing neurons.
 \mathbf{G}_{nf} : Set of non-firing neurons.
 $N_{\text{err}}(y)$: Number of erroneous bits in vector for pattern y.
 $\bar{s}(t)$: State vector of network at time t.
c: level of pattern corruption, or number of shifted bits relative to target pattern.
z: normal deviate (from statistics).
 μ : mean potential of neuron.
erf(x): error function (from statistics).

Chapter Eight: Dynamic Patterns

- L: number of subsets into which the firing neurons of a pattern are divided.
- D: number of neurons in a subset.
- F: number of cycles of forward connectivity between firing subsets.
- $\mathbf{D}_0, \mathbf{D}_1, \dots, \mathbf{D}_{L-1}$: L subsets of D firing neurons in a dynamic K-from-N pattern.
- I(K, L, N): Information content (in bits) of a dynamic K-from-N split over L cycles.
- ν : Decay constant for neuron potential.
- h_D : probability of a connection between two neurons in dynamic pattern case.
- P**: Pattern set to be learned.
- Z: Number of patterns to be learned.
- \mathbf{D}_k^i : k-th subset of pattern i.
- B: Connection strength between two neurons that make a non-zero connection.
- c: probability that two neurons fire within time F of each other.
- c_z : probability of forming a connection while learning pattern z.
- Z_S and Z_D : number of stored patterns in a *static* and dynamic network, respectively.
- Tref*: length of neuron refractory period in cycles.
- $e_x(t)$: Number of bit errors in external input pattern at time t.
- $e_n(t)$: Number of bit errors in network state at time t.
- a : random variable depending on presence or absence of connection between two neurons.
- Nerr: expected number of errors on each cycle of dynamic pattern recall.
- E: During simulation, number of patterns recalled without errors.

Chapter Nine: Pattern Association and Network Control

- L_1 and L_2 : numbers of subsets (and hence periods) of two patterns.
- f_b : beat frequency between two patterns of different cycle time.
- \mathbf{p}_t^j : t th subset of data pattern j.
- \mathbf{c}_t^j : t th subset of control pattern j.
- X**: Set of input data pattern vectors.
- Y**: Set of output data pattern vectors.
- n : Number of pattern mappings (or transformations) made using a given control pattern c_j .
- c: Level of pattern corruption = number of erroneous bits in the input vector.

Chapter Ten: Learning Strategies

- W_{ia}**: Weight vector of level a synaptic unit.
- t_a : duration of activity of level a synaptic unit after stimulation by pre-synaptic activity.
- α_a : learning rate of level a synaptic unit.
- e_a : probability that the level a synaptic unit is present for a given synapse.
- $O_a(t)$: Output of the level a synaptic unit of given synapse at time t.
- W_{ijm} : Individual level m weight component for connection between neurons i and j.
- l_m : Length of vector of level m synaptic weights for given neuron.
- $O(t)$: Cumulative output history over several cycles.
- $U_m(t)$: Neuron potential at time t due only to input from level m synaptic units.
- B_a : Standard connection strength for active level a synaptic unit.
- S: Number of sources of input to a given neuron.
- ΔW_{im} : Change in level m weight vector for neuron i.
- β_m : forgetting constant for level m synaptic units.
- C: Number of cycles of a dynamic pattern in which consolidation takes place.
- f_m : Number of repeats of a single pattern during consolidation.
- $Q(W_m)$: Distortion of level m weight vector due to learning single pattern.
- r_m : positive constant that decreases for increasing m .

On the Application of Neural Networks to Symbol Systems

1.0 The Potential to be Exploited

As we approach the end of the nineties it is clear that the field of neural networks has still not fulfilled its early promise. In the early days of research there was much talk of the development of artificial neural machinery that would rival and even surpass the intelligence of man. With each passing decade the public at large was told that a solution was only a few years away and the home would soon be invaded by a swarm of intelligent and obedient robots servants ready to take all the hard work out of domestic life. Early optimism, as usual, gave way to the dawning realisation that the issues involved were much more complex than first imagined and that a solution was decades away.

More than forty years later we are still waiting. Marvin Minsky, one of the founding fathers of Artificial Intelligence, is quoted as saying that the advent of the thinking machine is somewhere between four and four hundred years away (Stork, 1997). This is perhaps his way of saying that researchers still have no clear vision of what the issues are. Fortunately, even after all this time, interest and enthusiasm to pursue that most elusive of goals are still available in abundance. This is due in no small way to recent advances in our understanding of the most complex thinking machine around today: the human brain.

Certainly when we compare the most powerful computers of our age to the average human brain we find much to encourage us in our development of neurally inspired computing machines. The amount of computation the human brain is required to perform, even in the most mundane of daily tasks, is orders of magnitude greater than anything we are currently capable of reproducing electronically, while its memory capacity is seemingly without limit. An exercise in reasoned guesswork (von Neumann, 1958) put the total quantity of information stored in a human brain at around 10^{20} bits.

During every moment of its waking life a human brain must interpret a torrent of information from the five senses, fuse that data to form a coherent picture of the world and then co-ordinate its responses appropriately. It must be supremely

adaptive, learning new relationships between objects and ideas whilst using this information to plan courses of action in environments in which only partial knowledge is available.

There can be no doubt that the understanding of the mechanisms of the human brain has been one of mankind's most enduring preoccupations. Interest in this area can be traced back as far as the beginning of recorded history. Much of this study, however, has been conducted in terms of philosophy rather than science largely due to the relative complexity of the issues involved and the relative immaturity of the mathematical and scientific knowledge available; it is only during this century that we have had the tools required to make any serious progress in the 'reverse engineering' of the brain and the understanding of that most elusive of goals: the nature of intelligence itself.

1.1 Of Symbols & Synapses

Over the past few decades, two different approaches have emerged with the common aim of turning the study of mind and intelligence into an engineering discipline rather than merely a scientific one: one has acquired the name of 'Artificial Intelligence' (AI) and the other 'Artificial Neural Networks' (ANN).

On the surface they seem to be fundamentally different in the assumptions they make, the methods they employ and the problems they try to solve. The presence of each has been, at times, an annoyance to the other. At worst, the limited supply of research funds for research in this area has set one side against the other.

The AI philosophy has been around for about fifty years since the work of Turing and others led to the first useful computers in the early 1940s. The basic aim of AI are rather broad and, strictly speaking, do not preclude neural networks as a sub-field. A single definition is hard to find, but one reasonable attempt is "*The art of creating machines that perform functions that require intelligence when performed by people*" (Kurzweil, 1990). The underlying assumption is that the very processes of thought could be translated into algorithms. Thus, it remained to discover these algorithms to create intelligent machines.

In the AI approach, knowledge is traditionally represented as structures of symbols and manipulated using formal logic. These symbols are often represented in a form that approximates words of a human language in order to make it easy for the user to enter new knowledge and, subsequently, to understand the results of any computation executed on the knowledge database.

The alternative approach, that of Artificial Neural Networks (ANNs) has also been around for about fifty years. Its appearance has been attributed to the work of McCulloch and Pitts (McCulloch and Pitts, 1943) who postulated that it would be possible to realise any logical function using only simple sum-and-threshold units. The development of neural networks, as a field, moved away from the logical base of the McCulloch-Pitts model (permitting arbitrary mappings from inputs to out-

puts of the neural building blocks) while preserving the basic sum-and-threshold model. As the millennium approaches we find that statistical mechanics, information theory and the mathematics of chaos are increasingly used to provide the framework in which research is carried out (e.g. Gardner, 1988; Kelso, 1997).

It would seem that the long term goals of AI and ANN research are the same, yet the types of problems being studied and the level of representation used for each problem are often radically different. But what is the true relation between the two? Does only one approach provide a correct account of intelligence while the other is merely wasting intellectual effort? Or alternatively, separate though they are, does each in fact supply a necessary part of the whole solution to practical intelligent systems implying that greater progress would be made if there were greater cross-fertilisation between the two?

The assumption in this work is the latter; that both fields of research are necessary to help us to make progress in the design of intelligent systems that are both flexible and robust.

The next few paragraphs contrast some of the features and properties which are traditionally associated either with AI systems or with neural networks. Each comparison is made between stereotypic views of the two approaches to highlight several key concepts.

1.1.1 Representation and Processing of Data

In the AI paradigm, data is traditionally represented as structures of symbols. Any symbol can be 'accessed' to retrieve its contents and strings of symbols can be manipulated to generate new structures and symbols. Using a finite number of grammatical rules and a finite set of symbols, it may be possible to generate an infinite number of strings, thus permitting the representation of arbitrarily complex relationships between objects and ideas. By manipulating the hierarchy of symbolic data, the system can focus in on a tiny part of the problem or 'zoom out' and process the relationships between higher level entities. In this way, a system of limited resources can tackle problems of arbitrary complexity.

The neural network approach sees data as being represented by the output of units whose typical functionality is to sum and threshold input from other units or the external environment. In many neural net models the state of these units is able to evolve in time, by which means it is able to perform some kind of data processing. The construction and manipulation of data structures by neural networks is not a well developed area of study at this time.

1.1.2 Scale of the Problems Tackled

Neural Networks have typically been applied to problems which involve the mapping of a function or the storage and retrieval of bit vectors. Usually, the statistical properties of interacting non-linear elements are under investigation. The scale of the problems is sufficiently limited that the application of these results to the real world is often not considered.

AI research has typically jumped in at the deep end with problems of real-world interest. For example, the representation of the knowledge in a story written in so called 'natural language' or an algorithm to play chess at grandmaster level (such as IBM's Deep Blue which made the world press last year in its defeat of chess champion, Gary Kasparov).

On this point, the AI guru Minsky has argued that one of the major errors in AI research over the last twenty years or so has been its preoccupation with the solution of problems in particular domains, such as language, problem solving and the construction of intelligent robots. This preoccupation, he claims, has led to a lack of focus on what he regards as the roots of the problem: the fundamental understanding of learning and knowledge representation (Stork, 1997).

1.1.3 Autonomous Development of Competence

Part of the appeal of Neural Networks is that (depending on the nature of the problem space) they can be capable of modifying their structure over time by a process of trial and error in order to 'learn' how to exhibit the appropriate behaviour. The underlying assumption is that a sufficiently large network with a certain (but, as yet, undetermined) initial structure could adapt itself to *any* problem space without the need for an outside agent to determine the way of representing or solving the problem. The human brain is the ultimate example of this universal adaptability.

In much of the AI research to date, the object has been to demonstrate that the program or machine could exhibit enough intelligence to solve a task. Considering the scale of the problem, a mere *existence proof* for a solution was sufficient to be regarded as a success. The fact that its internal structure was painstakingly devised by a human, and that no evolutionary path can be shown, was irrelevant.

1.1.4 Logic, Truth and Extrapolation

For most of its existence the AI approach to knowledge relied on *truth maintenance* whereby every piece of information derived from the knowledge database was *guaranteed* to be correct by virtue of the laws of logic, so long as the original knowledge base contained only true statements. Extrapolation from known information into the unknown consisted only of *deductive* reasoning which is regarded as an ontologically safe procedure.

Truth maintenance makes book keeping easy, since every derived 'fact' is also a fact. But while such an approach is theoretically very appealing, in practical applications such systems often performed very poorly. The problem with the basic concept of truth maintenance was that the kinds of data with which most system would be confronted in the real world usually could not be written as a series of absolute truths. The notions of uncertainty and probability, which turned out to be so necessary for real applications, were not readily formalisable in this way.

The acceptance that the approach was too restrictive led to the development of more powerful techniques such as fuzzy logic (Zadeh, 1965) and expert systems (Patterson, 1990) which allowed a richer and more flexible representation of the relationships between objects and ideas. The formal logic limitation was lifted in favour of a more quantitative scheme.

From the neural networks perspective, the ‘new’ concepts which supported the development of expert systems and fuzzy logic were already built in to the foundations, so to speak. There has never been a notion of ‘truth maintenance’ in the ANN literature. Any attempts to model true deductive reasoning and formal logic using ANNs has been a peripheral activity (such as Ballard & Hayes, 1984).

The neural network principle is often used to map one set of patterns to another using the statistical relationships between input and output signals (Rumelhart, Hinton & Williams, 1986). A network will usually make mistakes during learning (and even while in actual use). The training phase is intended to provide examples of the real data to come in order to minimise the errors the system will make in the field, but there is usually a probability of error associated with any result.

Extrapolation in the neural network context can be far from error free and it is often necessary to pay careful attention to the network’s design to ensure that this *generalisation* performance is within acceptable error bounds (Bishop, 1995). It should be noted that the process of generalisation is one of *inductive* reasoning, a more flexible method of gaining ‘knowledge’ than deductive inference but one which is no longer guaranteed to be truth maintaining.

Returning to the issue of developing AI systems beyond truth maintenance towards fuzzy and probabilistic representations: in taking that step, the distinction between AI and neural networks becomes much less clear and one wonders in such cases whether the remaining differences are more cosmetic than real. A theme which is central to this work is that neural networks are merely one way of formalising a non-linear statistical model. Fuzzy logic and expert systems could be considered as just another way of doing exactly that.

1.1.5 Conclusions

The comparisons made above are not the only ones that can be made; more will present themselves naturally as the properties of the architecture and network are considered later on. It should also be noted that the comparisons are somewhat simplified; for each property described as an advantage of, say, AI it would be possible to construct a neural network which could display the same property. But the intention here is to try to highlight certain important properties of intelligent systems that we will try to embody in the work later on and to illustrate that each of these properties has a tendency to be associated with one (but usually not both) of the two major approaches.

To sum up, we extract one basic theme from the comparisons made in this section: while neural networks appear to be good at extracting the statistical struc-

ture of data, so far they have not shown the same ability as a generic AI system to provide the complex data structures and data manipulation framework that seem to be necessary for many real-world applications. Conversely, AI systems traditionally lack the subtlety of representation either to express the complex relationships between many pieces of data, or to extract and subsequently apply these relationships in a practical and robust way; these are properties which *can* be demonstrated in neural networks.

1.2 Underlying Philosophy of the Work

A little research into the aims and results of neural network and AI research led quickly to a number of key ideas which were pursued in the work to be described in this thesis. As will become clear, there was insufficient time available to fully explore the many possibilities encompassed by these underlying assumptions, either in the architecture or in the neural implementation which followed. However, to provide an element of context for the what will follow, this section briefly reviews the key ideas. The rest of the thesis attempts to justify them.

1.2.1 Symbols Represent Objects, Categories or Concepts

A symbol is a token that can represent any object, category or idea. It can be combined with other symbols in a structured relationship, producing an expression of arbitrary complexity, which itself can be represented by a single symbol. This definition of a symbol and its meaning follow the traditional lines laid down in forty years of AI research and is uncontroversial.

However, also by tradition, the encoding of the symbol itself is arbitrary provided that it is unique: it need bear no relation to the expression that it represents. Later points in this and later chapters will present arguments why this should not be so in an efficient neuro-symbolic system.

1.2.2 Symbols Delimit Relationships and Focus Resources

Symbols allow relationships to be defined between objects or categories. For each such definition, a system of finite resources brings its focus of attention to bear on a subset of the objects about which it has stored information, thus permitting it to break down knowledge structures of arbitrary complexity into chunks of manageable size.

Learning one relationship between a number of symbols should not affect other relationships which have been previously stored unless there is a logical reason to do so. Thus a symbol is a means of limiting the extent of a learning event, preventing it from interfering with existing, but unrelated, knowledge.

1.2.3 Symbol Encoding Must Have Structure

An element of the Classical AI paradigm is the representation of symbolic relationships in structures and their manipulation by processes which are sensitive to their constituent structure. If symbol encoding is arbitrary, there is no information available to an executing process to indicate how it should be handled. Thus, the symbol encoding itself *must* have a structure which indicates how it should be manipulated by any process. Symbols which are to be handled in a similar way would have similarities in their encoding which could be quantified using some metric.

1.2.4 Symbols Encoding is also a Function of Content

A symbol can represent an underlying structure (expression), made up of a number of other symbols which stand in a certain relationship with respect to each other. This structure is not the same as the encoding of the symbol itself (see the previous point). Some properties of the underlying expression are relevant to the expressions which will incorporate the symbol. But many are not. The useful properties *should be represented in the symbol encoding* in such a way that the symbol can act on behalf of its expression without having to access it directly (which takes time and consumes resources). This concept is the same as the previous one, but viewed from the point of view of the underlying expression, rather than the process.

1.2.5 Hierarchy and Inheritance are the Keys to Efficient Storage

Efficiency of storage can be obtained by identifying common features present in the relationships to be learned and representing the relationships in terms of these features. This is a common theme in neural networks where the features are statistical relationships between input bit patterns and between input and output patterns. In symbolic systems the concepts of categories and the inheritance of properties go hand in hand. Combining statistical feature extraction with inheritance is a means of increasing the storage efficiency of a symbolic system in a quantifiable manner.

1.2.6 Managing Hierarchy and Inheritance is the Key to Generalisation

The promotion of a learned relationship from an object to all elements of a class to which it belongs is a form of inductive inference since it transcends the level of explicitly given information.

The management of inheritance consists of the complimentary operations of the promotion of relationships to parent categories and of the division of one existing category into many during category refinement, which might occur upon encountering a counter-example. In this way, the system attempts to maintain all relationships at as high a level as possible (i.e. as general as possible).

The process of inheritance management may call upon many sources of guidance, such as deductive inference using other previously unconnected relationships, trial and error, or slow re-convergence of categories through lack of use.

1.2.7 Symbol Encoding Must Imply Causally Implemented, Correct Processing

Ultimately, the network must be stand-alone, creating symbol encodings and transforming them as directed by a control process which is itself a network of encoded symbols. The symbol encoding must be such that this property is upheld. By the law of infinite regress (also known as the homonculus principle) there cannot be another process at a higher level which interprets the meaning of symbols and changes the course of processing as a result.

1.2.8 Symbol Encoding Must Change as a Result of Learning

As an extension of the previous point, the encoding of each symbol cannot be fixed at the time it is created since all of its relationships are not known at that time. Thus the encoding must change as a result of further learning, allowing it to make those new mappings as efficiently as possible while at the same time preserving those relationships in which it is already a constituent.

1.3 Objectives for the Work

The first part of this chapter contrasted the twin disciplines of neural networks and artificial intelligence, describing (in very general terms) some of the properties they possess in the arena of so-called 'intelligent systems'. The emerging idea from that discussion is that a hybrid AI-ANN system offers (at least in principle) the opportunity to capitalise on the best ideas from both disciplines. In effect, to implement some kind of symbol system in a neural framework.

At the beginning of this work it was not clear what the form of the system should be. Thus, a list of objectives was set to aid in its definition. These objectives were initially somewhat vague but allowed the scope of the work to be specified at a high level. The goal definitions were revised as the work proceeded to incorporate an increasing understanding of the issues involved. There were six goals defined for the work. They are described next.

1.3.1 To Provide a Robust Substrate for Symbolic Computation

The final network must be capable of executing operations on encoded symbol structures. Requirements for *robust* computation must be explored to ensure that the results produced by the network are quantifiably *self-consistent* and reliable.

By robust, we mean that when subjected to finite input noise and other truth-preserving transformations of the input signal within a defined level of distortion, the network has the capacity to correct some or all of the errors. By self-consistent, we imply a higher level of error correction: that given a series of symbols in which an error has escaped detection at the single symbol level, the network has the possibility of detecting it and correcting for it due to inconsistencies in the symbol string. Thus, the network will not produce occasional bizarre and contradictory results without explanation.

1.3.2 To Provide a Means of Storing Large Amounts of Data Efficiently

Clearly, there is a link between the size of the network (in terms of the number of neurons, the number of synapses per neuron, etc.) and the quantity of information that it can store. Whether or not this data can be stored in an efficient manner is another question.

This goal can be easily quantified. Several metrics have been used in the past, such as the maximum number of orthogonal patterns reliably recalled or the maximum number of bits reliably recalled per physical bit of memory in the network.

In addition, we might also expand our thinking to include non-orthogonal pattern sets and more complex data structures. To quantify network storage it will be necessary to consider more complicated metrics, such as the maximum number of levels of data structure that can be stored, the maximum number of patterns that can be associated with any single pattern or the time required to access a given piece of data.

1.3.3 The Efficiency of the System Should Increase as the Network Size Increases

This goal acknowledges a deep seated expectation that larger networks should be able to handle their data more efficiently than smaller ones. Feature extraction, either in multi-layered systems (e.g. the Multi-layered perceptron; Rumelhart, *et al.*, 1986) or single layered systems (principal component analysis; Jolliffe, 1986) facilitate more efficient storage by creating representations of new data in terms of known regularities in the underlying distributions. We expect that, in principle, the larger the network the greater the opportunity to capitalise on such regularity.

In order to increase efficiency, the storage capacity of a network must increase more than linearly with the number of neurons assuming a fixed maximum number of synapses per neuron.

1.3.4 To Make Data Available for Computation *as Appropriate*

If a network is storing data on a wide range of subjects we expect that some data will be relevant to the question at hand, while other data will not. Unnecessary information should not interfere with the ongoing computation, but should be able to come into play as, and when, required. How does the network decide when information is relevant and when it isn't? How is data brought to bear when needed and ignored when it is not needed? Answering these questions is extremely important if we wish to produce an efficient system from finite resources. This explains the very deliberate use of the words *as appropriate* in the goal definition.

1.3.5 To Provide a Learning Mechanism which Facilitates the *Efficient* Acquisition of New Information

One of the hard problems of intelligent system design is providing a means by which the system can efficiently learn new information. The word *efficient* is key here. There must surely be an infinite number of ways to represent a given piece of knowledge but some are more efficient than others.

In this goal definition, the term '*efficiency*' can be measured in more than one way. Most simply, it could be applied to the quantity of knowledge that can be stored per unit of system resource; thus we are thinking only of *storage efficiency*. But in addition, one might measure the speed with which stored knowledge can be accessed and brought to bear on any given task. Here we are considering the memory access speed as a function of the database size, which is a kind of *execution efficiency*. Both measures of efficiency are important and are implied in the goal definition.

1.3.6 The System Should Be Realisable

In defining this goal, we seek to disallow solutions which will be impractical to implement. Clearly, what is not practically realisable today might become so tomorrow but some obstacles will still remain longer than others such as: (1) systems which rely on levels of interconnectivity which are too high to be realised in the foreseeable future or (2) systems in which the quantity of information required for each parameter update must be drawn from too wide a field. Thus the development of the system should take implementability issues into account.

1.4 The Structure of the Thesis

The remainder of this thesis is divided into eleven chapters. In chapter two, existing research which is relevant to this work will be briefly reviewed. This includes important results from neural network theory, AI and neuroscience. Discussion in this chapter will highlight the central ideas which will be brought together in the architecture development. Chapters three and four will provide much deeper coverage on key areas of the literature, including associative memories and reasoning systems.

Chapter five presents the network architecture development itself. It focuses on the requirements for knowledge representation and computation in the network and defines the system whose implementation and evaluation will form the rest of the work. The goals outlined in the introduction will be refined and quantified during this development.

From chapter six onwards, the major consideration switches to the implementation of the neural building blocks. The data coding scheme is considered in chapter six while in chapter seven the properties of a simple neural unit are explored. In chapters eight to ten this simple neural unit is developed, successive chapters adding new levels of complexity to its functionality. Chapter eight dis-

cusses the use of dynamic patterns to improve storage efficiency and to make the network more independent of external control. The simple static memory is thereby replaced with a temporal pattern of activity.

Chapter nine adds both a control network and the capability to form associations between data patterns. In chapter ten, the individual synapses become much more complex to investigate a concept which is referred to here as *hierarchies of learning*, again with the goal of an increase in system performance and efficiency. Each synapse now requires several bits of data to describe its state, and the algorithms which define the learning procedure become more complex.

Chapter eleven returns the focus to the high level architecture, discussing ideas for a unified neuro-symbolic system. The final chapter brings the work to a close: recapitulating the objectives, summarising the results and suggesting future work.

1.5 Summary of the Work

To provide a framework for the rest of the thesis, this section summarises the major results which will be presented. The survey of existing work in chapters two to four describes elements of AI and neural network theory, their perceived advantages and disadvantages, as well as attempts to overcome the inherent problems using hybridisation architectures.

The development of the architecture, which follows, is essentially an investigation into the necessary properties a symbol encoding must have to be applicable to a practical neural network implementation of a symbol system. In addition to the usual properties of being able to represent an expression in other expressions forming a hierarchy and being able to access its contents at any time, it is argued that a symbol in a practical neural system must exhibit additional characteristics that stem from the idea that the encoding of the symbol itself is directly linked to its meaning and that this encoding must be hierarchical but not necessarily decomposable at the symbol level. It is postulated that the symbol encoding cannot be fixed at its inception since it must evolve as the network learns new relations.

Finally, two potentially related but distinct concepts are presented. First the implementation of inheritance and hence of generalisation using a hierarchy of pointers and second the ability of the network to slowly assimilate knowledge of different persistences by continually evolving its encoding, referred to as *learning hierarchies* and again based on a hierarchical pointer approach.

An outline architecture is then proposed which would exhibit those properties of symbol systems that have been identified in the literature as being desirable. The development describes the sorts of operations that must be executed at the symbol level and is based around the proposed symbol encoding.

The remainder of the thesis is concerned with the development and characterisation of the neural building blocks which will make up the network. The first such chapter compares a number of potential neural encoding schemes. From three candidates the K-from-N encoding is chosen since it represents a controllable and predictable encoding which the impact of known levels of single bit errors can be quantified and hence compensated for.

Later chapters analyse candidate networks which use the K-from-N encoding. The first structure is based on a recurrent form of the non-holographic associative memory of Willshaw *et al.* (1969). It is shown that the network is capable of learning patterns in a single presentation and that for low K/N ratio the capacity of the network can far exceed that of an equivalent Hopfield network, which it resembles. The memory capacity is analysed as a signal detection in gaussian noise problem.

The next stage in the development is to decompose each static K-from-N pattern into L subsets of D-from-N, each pattern now stored and recalled with a period of L cycles. It is shown that this dynamic pattern approach increases the storage capacity of a fixed size network, but at a cost of transmission bandwidth and signal to noise ratio.

The network is extended to include associations between triplet of patterns, implementing the functional mapping aspect of the architecture. While the resulting capacity is reasonable, it is asserted that it is insufficient to permit the model to act as the needed building block of the network.

The final chapter in the network development considers the neural implementation of the learning hierarchies scheme proposed during the symbol encoding development of chapter five. Here, the network tries to develop feature detectors to optimise the storage of given associations, addressing the stability-plasticity dilemma and presenting the many available trade-offs in the design of more complex learning systems.

The last chapter before the conclusions tries to unify both the neural network and the symbolic ideas proposed in the work. To do so, it examines the current impediments that we need to remove in order to realise a true neuro-symbolic computer capable of exhibiting general intelligence, proposes some solutions to the problems and suggests how earlier ideas in the thesis could be used to address these issues. The weakness of forward-chaining reasoning is examined, as well as the failure of current reasoning system to handle and generate symbolic structures capable of modifying the very structure of the machine that generated it. The lack of focus on the management of limited system resources is also explored as part of a discussion on possible ways of implementing parallel processing at the rules level. Finally, a proposal is made for handling sub-symbolic, so-called intuitive, reasoning.

2.0 Introduction

The study of neural networks encompasses a wide range of disciplines from pattern recognition, signal processing and statistics through to neuroscience and applications of chaos theory. By contrast, the science of Artificial Intelligence has traditionally claimed the language oriented disciplines as its own: natural language processing and structured knowledge representation. These traditional boundaries are being continually eroded, however, generating yet more new results as each discipline cross-fertilises with the other.

Since the fields with which this work is concerned are so wide in scope, the survey of existing work is divided into three separate chapters. This chapter, while presenting an overview of the fields of artificial intelligence and neural networks, will restrict itself only to those aspects which are relevant to the work presented and mention in passing those areas which have been avoided. The next two chapters will examine in depth specific areas that are of particular relevance to the work.

Being quite long, this chapter has been divided into four major sections. The first section discusses aspects of the neuroanatomy of the human brain, emphasising particular insights that have influenced the work. This is hardly an extensive coverage of the subject (which would be out of place here, in any case) but will illustrate several key concepts which are central to the work. The second section focuses on the symbolic logic approach which is the field of Artificial Intelligence; it highlights a number of key concepts in knowledge representation and intelligent problem solving, including first-order logic, production systems and expert systems.

The third section is concerned with relevant aspects of neural networks. The final section describes attempts to consolidate the connectionist and AI approaches. Specifically, it attempts to answer the question whether the *neuronal* aspects of a 'neuro-symbolic' system are merely superfluous implementation details of a Classical *symbolic* system? If not, how can we benefit from the advantages of both disciplines to produce something better than either alone?

2.1. Neuroscience

Much of what motivates research into intelligent systems comes from the knowledge that such systems already exist in the brains of higher animals; principally Man. This section describes aspects of the human brain's structure and chemistry which have influenced this work in some way.

There is no claim here that anything that has been inspired from the working of the brain should be taken as an explanation of how the brain actually functions. It will be some years yet before neuroscience has fully unravelled the true nature of brain function; in the meantime, we have no obligation to restrict ourselves to a neural architecture that mimics that of the human brain.

2.1.1 A Single Neuron

A single neuron is, in itself, a highly complex biochemical machine. In most models of a neural network only a small fraction of this complexity is represented. In a real brain, individual neurons vary considerable in their size and connectivity. However, most neurons share a large number of characteristics (Crick and Asanuma, 1986):

Each neuron grows complex input tree structures (called *dendrites*) onto which other neurons may make connections called *synapses*. Each neuron produces a single output connection (called an *axon*) which projects to many other neurons, sometimes in distant parts of the brain. Neurons receive electrical stimulation from other neurons via chemical messengers transmitted at the synapses. When enough potential has been received in a certain space of time, the neuron 'fires', generating a spike down the length of its axon. Most neurons can produce pulses at a rate of up to 500 per second, but on average this rate is between 50 and 100 pulses per second.

After a neuron has fired there is a time when the neuron must reset its electro-chemical apparatus and restore its static potential. During this time, it is unable to fire again regardless of stimulation. This is called the *absolute refractory period* and is between 1 and 2ms in most neurons. After this period is over the neuron enters a second period called the *relative refractory period* in which the neuron can be made to fire, but only with higher than normal levels of stimulation. This period lasts for about 5ms.

Turning now to the synapse itself, it is known that there is a wide variety of different chemical messengers used as the medium of information exchange. Some are fast acting and exist for only a short time, others work only indirectly and their effects are felt over a much longer period of time. All neurons of a single type seem to release only a small subset of the possible neuro-transmitters. The distribution of transmitter substances among neuron types is clearly as much a part of the neuro-architecture of cortex as the physical connectivity of the neurons themselves.

2.1.2 Different Types of Neuron

There appears to be a wide variety of neurons which are specialised for a particular function. These specialisations include the size and shape of the neuron, the neuro-transmitter(s) used to communicate with other neurons as well as the structure of its axon and dendritic tree.

Research into the form and function of the different neuron types have indicated that any one neuron *either* make excitatory or inhibitory connections to other neurons, but *never both*. This contrasts sharply with most mathematical models of neural networks which permit any neuron to make excitatory and inhibitory connections without restriction and even allow an excitatory connection to become an inhibitory one during learning (such as the MLP of Rumelhart, Hinton and Williams, 1986). In the neo-cortex, this arrangement may have come about only through constraints of the biochemistry in which it is implemented, or by virtue of the evolutionary path which shaped it. However, there may be more fundamental principles built into its structure which provide clues as to efficient methods of using large parallel networks. This remains to be clarified.

Most neurons in the neocortex are excitatory in nature. Furthermore, most excitatory neurons are pyramidal in shape. Each pyramidal neuron synapses with many others of their kind (up to ten thousand) in both its own and surrounding regions of cortex. In addition, it is these excitatory neurons which provide the main output of each neural sub-system to other brain systems, as will be described in the next section.

The inhibitory neurons in the neo-cortex use neuro-transmitters that are different from those employed by excitatory neurons and seem to follow different rules of connectivity to the pyramidal neurons (Crick & Asanuma, 1986). They are far less numerous than the pyramidal neurons, with estimations (Crick & Asanuma, 1986) indicating that every pyramidal neuron is innervated by six such inter-neurons, on average.

Inhibitory neurons make connections only with a small region, hence their other name of *inter-neurons*. They never extend their axons into other regions of the cortex. The conclusion that many researchers, including Crick and Asanuma, have reached is that these inhibitory inter-neurons provide the negative feedback that balances the otherwise entirely positive feedback between the *information carrying* pyramidal neurons.

2.1.3 Neuronal Organisation of the Neocortex

Our attention now turns to the actual structure of the lowest level of neural organisation. Within the neocortex, the neurons themselves are not spread out uniformly. They tend to clump together into groups. According to Shepherd each such group comprises of about 110 neurons and he gave such groups the name 'micro-column' (Shepherd, 1990).

A higher level of organisation was also identified by Shepherd (which he named the hyper-column) in which groups comprising about 50,000 neurons could

be labelled as a unit by virtue of their noticeably dense inter-connectivity. Later work challenged the modularity of this higher-order unit (Blasdel, 1992 as cited in Calvin, 1995) though the idea of the micro-column remains valid at this time.

This discussion will concentrate on the low-level micro-column organisation. Within such a column there is a high degree of intercommunication between neurons with relatively less communication between columns. Shepherd suggested that the micro-column is the replicatable unit from which the cortex is constructed.

These layers exist in all areas of cortex, but the relative size of each layer seems to depend on the particular specialisation of the cortical region. Regions in the sensory areas tend to have denser layers II and III (the input processing layers) than regions in the motor cortex, which in turn have relatively more developed layers V and VI (the output layers). See the figure below, derived from Shepherd, (1990).

The following details, on the probable flow of information within the cortex, are taken from a combination of Crick & Asanuma (1986), Kolb & Whishaw (1996), Shepherd *et al.* (1990), Martin (1996) and Calvin (1995 and 1998). Input from the senses passes through the thalamus to arrive in layer IV of the neo-cortex. It is then distributed (via layer I) to the pyramidal neurons of layers II and III. The output of these layers passes down to layers V and VI and is also distributed to nearby neocortical regions (again, arriving at layer IV).

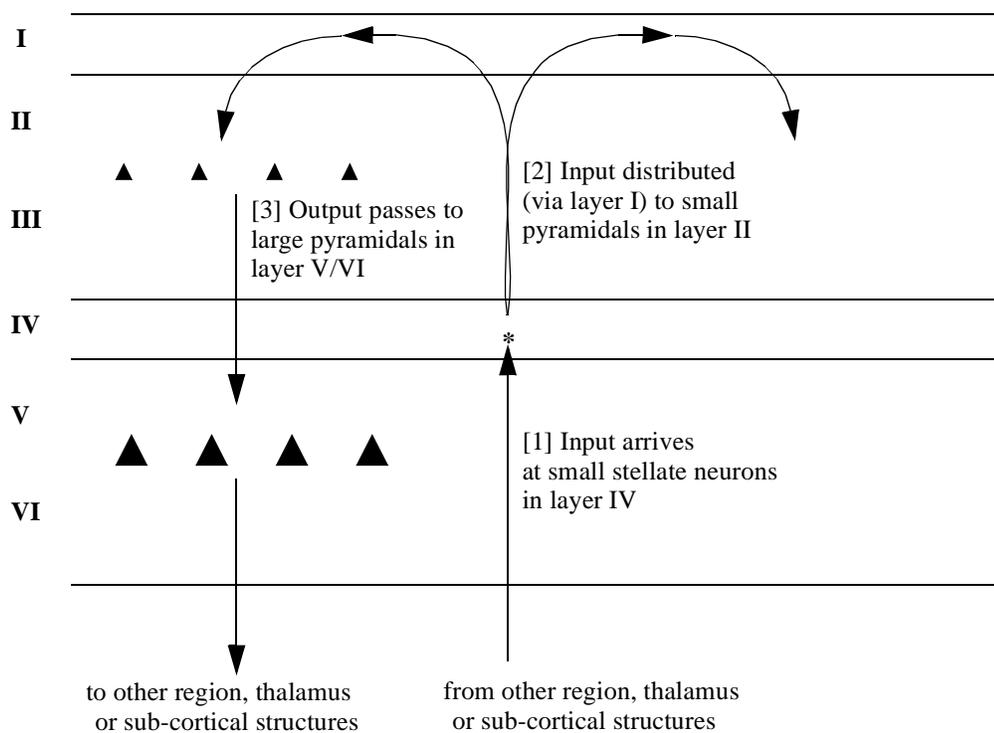


Fig. 2-0 One possible explanation of information flow in the neocortex.

The output of layers V and VI (from large pyramidal neurons) usually projects to non-cortical structures, the exact nature of which depends upon the

function of the particular cortical region. In addition, layer V and VI pyramidal neurons project back to the thalamus, thereby reciprocating the feedforward connections that the thalamus made with layer IV.

Summarising, it would seem that the flow of information begins by entering the thalamus and then follows the route through cortical layer IV to layer I and from there vertically downwards through II, III, V, VI and back to the thalamus. It is interesting to speculate how the pyramidal neurons of layers II/III might form some kind of map of the input space (distributed via the input layer, IV, and the superficial layer, I) and likewise the pyramidal neurons of layers V and VI might form some kind of mapping appropriate for the output of each micro-column.

2.1.4 Neural Activity

In many neural network models, each pattern to be stored is made up of '1's and '-1's such that the mean level is zero and it is uncorrelated with any other pattern (e.g. Hopfield, 1982). This implies that 50% of the neurons are 'firing' at any given time. In contrast, experimental evidence suggests that in the human brain the percentage of neurons which are active at any given time is closer to 5% (Amit, 1989).

With a lower level of activity the total amount of information that is represented by the pattern of neurons is less than the theoretically maximum value (which occurs when every neuron is firing half the time and there is zero correlation between them). Is there some other reason why the activity is so low within the human cortex? This point will be taken up again in the review of associative memories in chapter three and in the discussions on the symbol encoding scheme in chapters four and six.

2.1.5 Types of Memory

In the last hundred years or so, the field of psychology has given rise to much data on the mechanism underlying memory in both animals and man. It is now well accepted that the memory is not a homogeneous slate onto which ideas can be written and subsequently erased, but a complex hierarchy of different types of memory which can be classified in terms of the type of information that they store and the duration for which each memory persists (Klein, 1991).

One example is the declarative, working, memory that is responsible for the short term storage of relationships between objects and ideas (Graham, 1990). Items stored here may have a duration of seconds or minutes. Closely related to the working memory is the long term reference memory which retains such information over months or years.

It is not yet known if short and long term memories are physical distinct structures based on potentially different organisational principles, or a single structure which undergoes physiological changes to permanently store data. In recent years, it has been speculated that memory should not be pigeon-holed either as long-term or short-term; that a whole range of processes may be involved in consolidation, creating a hierarchy of 'memory persistencies'. Of course, there is any

number of theories which attempt to explain these processes (see Lynch & Baudry, 1984a and Crick, 1982, for examples).

What is also clear is that the synapses between neurons, which are now assumed to be the site of changes involved in learning, are not single-value multipliers as they are modelled in most neural network architectures. They embody highly complex chemical processes (Graham, 1990; Hall, 1992) the analysis of which is a science in itself.

2.1.6 Summary of Neuroscience

From this brief exposition on cortical organisation, a few key points emerge. First, the neo-cortex (which represents about 80% of the neural population) is largely composed of simple repeating units of around 110 neurons. Specialisation of these units is mainly brought about by virtue of the sources of data that they process and the relative interconnectivity between cortical regions.

Secondly, information appears to be represented using pyramidal neurons whose outputs are all excitatory in nature. In an active (i.e. working) area of cortex, only about 5% of the neurons appear to be firing above a background level. In contrast, inhibition by locally connected inter-neurons seems to act as control mechanism and not as a inter-columnal carrier of information.

Finally, the synaptic connections between neurons are not simple single-valued entities but complex biochemical systems in which many interacting chemicals play a part. Current theories postulate a hierarchy of interacting systems that together give rise to the process of learning.

2.2 Artificial Intelligence

2.2.1 The Basic Ideas Behind AI

Distilling a single notion of the idea behind AI is no easy task: the subject itself encompasses a wide range of topics, each of which has its own set of goals. For example, some approaches are concerned with machines that *act or think like humans* whereas others are concerned only with machines that *think or act rationally* (Russell & Norvig, 1995). Cognitive Science (a field hybridised with psychology) falls into the former category, occupying itself largely with the modelling of the thought processes of humans and other higher animals.

Since the early days of AI, new fields such as Cognitive Science have been spawned, each with its own objectives, leaving AI itself as a generic ‘applied intelligence engineering’. The common thread in all such sub-fields had been the use of language based schemes for the representation and manipulation of information, though there are some exceptions, even neural networks itself being a legitimate areas of research in AI.

2.2.2 First-Order Logic

Perhaps the most successful formalism of knowledge in AI has been the language of First-Order Predicate Logic (FOPL or just FOL). It permits the specification of relations between objects and the definition of properties for objects, using *predicate functions*. Objects can be categorised, allowing properties to be assigned to objects by virtue of their membership of those categories. Expressions are created using the normal range of logic constructs such as AND (\wedge), OR (\vee), NOT (\neg), together with the implication symbol (\Rightarrow). For example, to say that anything which is a mammal and has two legs is a human, one could write:

$$\text{mammal}(x) \wedge \text{legs}(x, 2) \Rightarrow \text{human}(x)$$

Thus for any object, x , for which the left hand side of this expression is true, the right hand side is also true by implication. The existential quantifier (\exists) allows the representation of the fact that at least one case exists in which the expression is true. For example, to say that there is at least one honest man:

$$\exists x, \text{man}(x) \wedge \text{honest}(x)$$

Finally, the universal quantifier (\forall) allows relations to be defined, based on a shared property. For example to say that every dog dislikes ice-cream, we write:

$$\forall x, \text{dog}(x) \Rightarrow \neg \text{likes}(x, \text{ice-cream})$$

First-order logic is powerful not only because it allows the formal representation of objects and relations (which can then be manipulated using the rules of logic) but also because such relations can be defined for whole categories of objects.

The aim of the logical manipulation might be to prove a formula (such as in General Problem Solver, Ernst & Newell, 1969) or, in database applications such as the commercially available Oracle package, to deduce which data items match a user query.

2.2.3 Knowledge Bases & Deductive Inference

A knowledge base is a collection of facts and/or relations defined between a set of objects, expressed in a formally defined language. The formality of that language facilitates the transformation of its sentences to derive new sentences which are logically entailed from them. The grammar of the language, together with the rules of inference, ensure that only true statements are produced by a formal system which is given only true statements as input. In this discussion first-order predicate logic (FOPL) will be used as the example language but there are others (e.g. temporal logic, modal logic).

All inferences in a formal system are *deductive* rather than *inductive* in nature. This is necessary to support the premise that only true statements can emerge as a result of inference on a knowledge base which itself consists only of true statements. Induction generates sentences which can be less specific than the

sentences from which they are derived. As a result they are subject to error, but permit the system to generalise from previous experiences when presented with novel situations. In traditional AI, only deductive inference was widely considered. Only in recent times has induction come in for much scrutiny (e.g. Wolpert, 1995).

The application of deductive inference is usually performed in one of two ways: either as forward chaining or as backwards chaining, depending on the task being performed. In forward chaining, every time a new fact is added to the knowledge base the system must infer all of the new facts which are now true as a consequence. For example, the database contains the following two sentences:

$$\text{furry}(x) \wedge \text{drinks_milk}(x) \Rightarrow \text{cat}(x)$$
$$\text{drinks_milk}(x)$$

Initially, no new inferences can be made. Imagine that the database is now given the sentence:

$$\text{furry}(x)$$

then it can apply its inferential rules to all of the sentences of the database in order to deduce any new truths. In this case it learns using the Modus Ponens rule that:

$$\text{cat}(x)$$

Proving one fact can trigger the proof of another, and so on. Modus Ponens is the most well known rule of inference and dates back to the writings of Plato. There are several others which form the basic set of tools for logical deduction. As they are not directly relevant to this work, they will not be discussed in detail. (See Russell & Norvig for a more detailed account). The important point to note is that the rules themselves are independent of the meaning of any sentence. Instead, they are executed on symbols which have certain properties, creating new relations as a function of those properties.

By way of contrast, backwards chaining, while using the same rules of inference, proceeds in the opposite direction. Thus, presented with a goal in the form of a conclusion, the task is to search the database for premise which will justify it.

Using the same example database as before, then given the query:

$$\text{cat}(x)?$$

the system can show that it is true by locating the sentences which are necessarily entail $\text{cat}(x)$. In this case the entire set is needed. Some premise might not be justified in which case the rules of inference can go back an indefinite number of levels to prove the premise upon which the conclusion must be based. If, for example, $\text{furry}(x)$ was not in the database, the system could search back for other relations which would logically entail it, to support the notion that $\text{cat}(x)$.

Forward and backwards chaining are different ways of using the same knowledge base. The former spends resources up front to ensure that all logical consequences of its sentences are made explicit even though some conclusions may not be useful in the future. The latter only uses resources at “run-time” to

derive needed conclusions but as a consequence it is usually slower for any given query. To use the current example, the forward chaining method will have already precalculated the existence of $\text{cat}(x)$ (or lack thereof) whereas backwards chaining must prove it in response to the query.

The use of transformational rules applied to sentences expressed in a formal language is at the heart of symbolic AI. Deductive inference is a powerful tool which, as will be discussed later in this chapter, is slow to be developed in neural systems.

2.2.4 Production Systems

The extension of logical representation to systems capable of acting in an environment led to the first production systems (such as Xcon which was a component configuration tool designed for use at DEC (McDermott, 1982). As shown in the diagram below, such a system consists of a knowledge base of rules (each rule stating the actions to perform when the given conditions are fulfilled) and a working memory (containing propositions which are true at any given time).

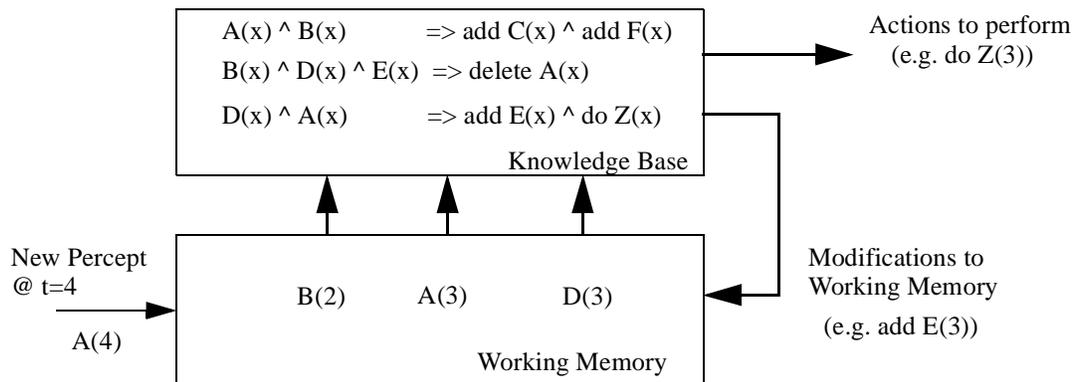


Fig. 2-1 An example production system.

After the arrival of a new percept, ($A(4)$ in the example a time $t=4$), the contents of working memory are applied to the left hand side of all of the rules in parallel, in what is called the match phase. If any rules have all of their conditions fulfilled, the right hand side of that rule is marked for execution. Execution can involve either changes in working memory or actions performed in the environment. In most implementations, there is a phase of conflict resolution after the matching phase in which any conflicting actions are resolved (such as two firing rules which collectively want to both add and delete the same item in working memory). The rules used to resolve conflicts depend on the implementation. In the final phase, the remaining actions are performed.

New rounds of rule firing can occur at this point, even without the arrival of a new percept, due to the changes made to working memory as a result of the previous actions. The match, conflict resolution and act phases are repeated until no new rules are fired during the match phase. The system is then tranquil until a new percept arrives.

Production systems are a form of state machine, in that there are state variables (the working memory), input (in the form of new percepts), a mapping from input to output (the rules) and the generation of both a new state and output signals.

As an implementation of an intelligent agent, the production system design offers a number of desirable features. Firstly, the rules are evaluated in parallel at each time step. From an implementation point of view such an approach is amenable to execution on a parallel processing system. Secondly, each rule can be written without reference to any of the others. Thus, the knowledge base can be built up in incremental fashion rather than be fully specified in advance. As a related point, the rules can be written without reference to how they will be used (i.e. they are a function of a particular *knowledge domain* rather than of a particular *problem*).

The disadvantages of the system stem both from the parallel nature of the rules and the restriction to purely logical representations. All of the rules are evaluated in parallel, so it is possible for many rules to be activated which are not relevant to the problem being solved. The algorithm used in the conflict resolution phase to prune redundant or conflicting actions seems somewhat arbitrary (for example, some applications give priority to rules in which the items in their conjunctions arrived most recently in working memory).

One possible approach to reducing the number of rules that are tested when they are irrelevant is to collect several rules together into a ‘block’ and permit whole blocks to be either active or inactive. Inactive blocks would be ignored and so would not consume resources. Clearly, extra rules would be needed to active and deactivate these blocks.

There are several drawbacks to such an approach. First, the behaviour of the system becomes harder to predict since its response to a given stimulus may change as a function of which blocks of rules are active and which are not. Furthermore, any errors in coding of the meta-rules responsible for activating each block could lead to shifts in behaviour out of proportion with the degree of error. The whole notion of having some rules which are more critical than others is intuitively unsatisfying, as is the coarse grained nature of the control. It is unlikely that such granularity is optimal in most environments.

As an overall comment on production systems, the restriction to purely logical representations implies that they suffer from the same fragility as any purely logical AI system, although it might be possible to extend the rules to include probabilistic or fuzzy representations. (See “Semantic Networks” on page 23.)

At a higher level, one might question the philosophy behind the production system approach. The global use of parallelism eliminates the need to decide which rules and data are relevant and which are not. The only approach adopted to deal with this is perhaps too coarse grained. What appears to be missing is a fine-tuned guiding system which applies rules (or even parts of rules?) when they are potentially involved and suppresses them when they are not. Essentially, there is no formal way of efficiently channelling system resources into the useful (or at least *potentially* useful) pathways.

Chapter four provides a more detailed review of a number of the implementations of production systems in the literature. For example, a production system forms the heart of the Soar architecture (see section 4.1.1, page 86). A neural implementation of a production system are also considered.

2.2.5 Semantic Networks

At the same time as much of the development of first-order predicate logic (FOPL), a parallel branch of research was under investigation. This approach, called semantic networks (Quillian, 1968), was viewed by its proponents as providing a more flexible and hence more powerful formalisation of the relationships between objects and sets. Whether or not this is indeed the case is now a matter of some debate. However, the use of graphical notation, with nodes representing concepts and objects, connected to each other through links representing the relations, certainly set it apart from the purely textual form of first-order logic.

The essence of semantic networks is to provide a framework for logical inference, but using the modelling of class membership as a mechanism for inheritance of parent properties. The example below, taken from Russell & Norvig, (1995), illustrates the concepts.

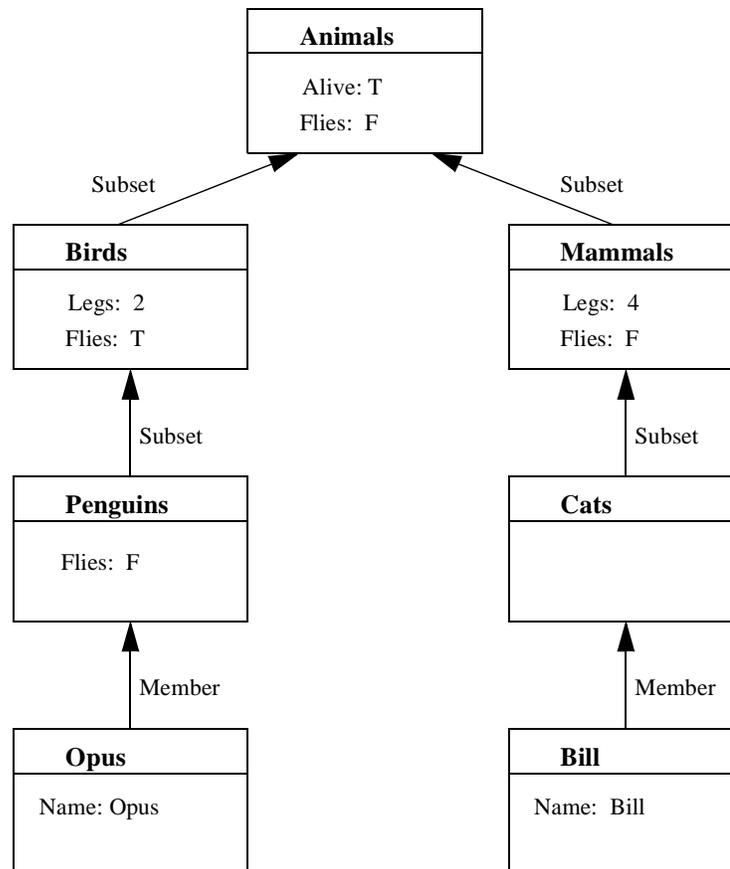


Fig. 2-2 An example semantic network.

Each box (usually referred to as a *node*) represents a concept or object. The arcs connecting nodes indicate relationships between them. Such relationships can

indicate membership of a object or class within a higher level (and hence more general) class. Finally, within each node are properties associated with it, and the appropriate value of that property for that node. Taking, for example, the node 'Birds' representing birds, we see that birds have two explicitly assigned properties, that of having two legs and that of flying. Furthermore, by virtue of the links between nodes, we see that birds are a subset of animals and that penguins are, themselves, a subset of birds.

The power of semantic networks lies in the way that properties are managed. If a property of a given node is not specified explicitly, its value can be passed down, or *inherited*, from a parent class. Thus, even though the node for birds does not state that birds are alive (with property *alive* = True), it can inherit a default value from its parent class 'Animals'. In contrast, the fact that we are told that Birds do fly (property *flies* = True) takes precedence over the less specific information from the parent category (for 'Animals', property *flies* = False).

The overall effect is that properties are only stated explicitly for a node if the values of such properties do not conform to those of the class(es) in which it is a member. Information is managed much more efficiently than in a network in which every node carries a full set of all properties.

Another important aspect of semantic networks is that they are regarded as a good framework in which to perform non-monotonic logic. Such logics differ from standard FOPL in that the addition of a new fact to the database can cause other 'facts' to be invalidated and thus removed. The size of the database is no longer a monotonically increasing function of time.

Most of the problems associated with semantic networks are well known and many potential solutions have been proposed in the literature. The principle problem occurs when a class or object has multiple parent classes. It is entirely possible for the property values inherited from the parents to be incompatible. How does the system decide which value to accept? Several strategies have been developed and are discussed in chapter four which considers semantic networks in more detail.

A second problem with semantic networks is the rather vague semantics associated with the subset relation. In the figure, we see that the node 'Birds' is a subset of the node 'Animals'. This may be true, but in what respect is it a subset? In what way is it different from its sibling category 'Mammals'?

One method of deciding this could be to compare the properties and values for the two sub-classes with respect to each other and also with respect to the parent class. Birds are stated as flying. This is a general differentiator for birds. But penguins, a sub-class of birds, do not fly. So in general it is not possible to give precise definitions of sub-classes with respect to parent classes and siblings. This can make the task of assigning a new object to the correct category a resource-consuming process.

A less discussed issue is concerned with the usage of semantic networks in the wider context of intelligent system design. Most research in semantic networks

is concerned with the issues involved in the correct inheritance of properties from parent to child nodes. Such networks can answer questions about properties of objects when the information has not been given explicitly, and thus are performing useful computation in their own right. However, one must suppose that once such implicit properties have been established, the intention is that they be used in processing other relations between objects and categories. (For example, having decided that penguins are indeed alive, this result might be needed to decide that penguins need to be fed). Often this wider purpose is forgotten. This is unfortunate since the constraints of the overall system would no doubt influence the way in which the inheritance mechanism functions.

One final point of note is that each time the system needs to establish the properties associated with a particular node, it must assemble them by searching the network of connections and dependencies between nodes. In principle, it could store all of the potential properties with each node as it calculates them, but there are two problems with this.

First, making all properties and their values explicit for every node would, in general, consume much more memory than the original tree. Second, any changes to the network (either a property value in a parent class or a new link) would require a phase of network updating since a single change could have an impact in many parts of the network. Depending on the network and the rate at which changes occur, this might consume more resources than the local regeneration of properties for a few nodes at run-time.

We can compare this run-time behaviour with the backward/forward chaining of first-order predicate logic systems discussed earlier. By reconstructing the set of properties of a node when needed, the system is performing a form of backwards chaining. In contrast, forward chaining would be achieved by making all properties explicit and re-evaluating them at each change in the network.

The issues involved in semantic networks are presented in detail in chapter four, along with some implementations as neural networks.

2.2.6 Description Logics

Description logics were created to address the problem of vaguely defined subsets, as mentioned above for semantic memory. In essence, they allow the notion of classes and membership, but force the knowledge base designer to make explicit the definition of each category.

An example given in Russell & Norvig, is taken from the CLASSIC language. Here, the definition of one sub-class, Bachelor, is defined explicitly as the conjunction of three others:

Bachelor = And(Unmarried, Adult, Male)

This makes it easy to assign new objects to classes or to decide if a given class can be subsumed by another (using their respective definitions). However, they lack a means to express negation and disjunction, as Russell & Norvig note.

2.2.7 Uncertainty, Belief and Expert Systems

One area which will not figure strongly in the work presented in this thesis is the modelling of uncertainty and belief. Uncertainty allows the system to model its expectation of the frequency of an event over multiple trials. The mathematical discipline of Bayesian statistics exists independently of AI and provides a solid foundation for making predictions about the probability density function for certain events given prior data (Bishop, 1995).

Belief allows the system to express its degree of certainty in the truth of a given proposition. The body of theory behind belief is much more recent and less well developed than that of probability. Some work on belief (the Dempster-Shafer rule) seeks to differentiate between uncertainty (where an event is yet to occur but has a certain probability) and ignorance (where a fact is already established but is not known by the reasoning agent) (Shafer, 1979).

The use of utility theory allows a planning agent to quantify the desirability of goals and sub-goals and hence make quantified comparisons of its actions based on their expected payback. This quantified decision making is the subject of decision theory which embodies both utility theory and probability theory. Decision theory itself lies at the heart of expert systems which must provide quantified answers to queries in such domains as medical diagnosis, and financial planning (Patterson, 1990).

Even though the inability to model such concepts as uncertainty and belief would render an intelligent system fragile in many real world applications, the issues involved add a new level of complexity to the architecture and implementation. Their omission is therefore a result of limiting the scope of the research, not of denigrating their importance in the design of robust symbol systems. It was envisaged that the architecture would need to be extended at some later time to add the notions of belief, uncertainty and utility. How that might be achieved is left for future work.

2.2.8 Case-based Reasoning (CBR)

In the last few years a new discipline has emerged in the AI research arena which is not only a product of the disillusionment felt towards formal logic problem solving but is also a step towards the neural networks camp in its use of pattern matching and approximation as primary tools. Case-based reasoning seeks to solve new problems in a particular domain by calling upon stored knowledge about previous, related cases and adapting the known solutions to fit the problem's novel aspects (Leake, 1996).

The process of case-based reasoning when applied to a new problem are roughly as follows (from Kolodner & Leake, in Leake (ed.), 1996): First, a ballpark solution is proposed based on a case already stored in the database which is

similar, in certain respects, to the problem at hand. Next, an iterative loop begins in which the proposed solution is first criticised to identify any aspects which fail to achieve the desired goals in the new problem, and then an adaptation phase is executed which is intended to correct any problems encountered with the proposed solution (again, drawing on the database). The loop is exited only when the system believes it has a solution which will resolve the problem, or can come up with nothing better. The next step is to apply the candidate solution to the new problem and observe the results. Finally, the results are processed and the database updated accordingly. Both positive and negative consequences of the systems actions are a viable source of new data.

The intended benefits of CBR are manifold. Applying precalculated results from previous experiments is potentially much less compute intensive than recalculating or re-proving a result from scratch every time. Thus, there is a potential to reduce resources for the same level of performance relative to an expert or production system. Also, the ability to fill in the gaps of unknown information by analogy with stored cases is intended to help the system to produce reasonable results while acting in uncertain environments.

Case-based reasoners differ from expert systems in that the basic unit of currency in CBR is the *case* rather than the *rule*: a case is essentially an experiment which produced a result in the past which can be adapted and re-applied in the future. Even though the rules used by an expert system may have been built up by previous experiments, the contexts in which they were used in the past are not retained. So on the one hand we see the benefit of working only in terms of rules; it is easy to know if a rule should fire or not as there is little context information necessary.

But we can also see the added benefit which CBR brings to such systems, the property which expert and production systems essentially lack; the ability to use higher level information to decide on the relevance of knowledge and rules and avoid wasting computing resources on exploring irrelevant side branches (such as evaluating rules which will serve no useful purpose in the context of the current problem). Thus CBR represents a step towards a higher level representation of the problem and its solution.

2.2.9 Summary of Artificial Intelligence

In this section we have briefly reviewed many broad classes of AI systems. Several key ideas formed a recurring motif and will be drawn on during the architecture development in chapter five.

First, the idea behind the symbolic approach itself. Elements in the world of interest can be abstracted as symbols. The relationships between these symbols can be expressed in a formal language which permits them to be manipulated to derive new knowledge and to make true statements about the world. It is straightforward to add a new piece of knowledge to the knowledge base without disturbing existing data (as long as the new data does not conflict with existing data).

The second key idea to emerge is the re-usability of relations and knowledge. A relationship derived in one place can find application in another unrelated area. Symbols can represent small parts of an given situation, parts which may occur again and again in different positions and configurations. The symbol system becomes more efficient by recognising regularity and modularity in the world and exploiting it. This idea is also the basis for case-based reasoning in which the entities involve are entire episodes of earlier experience, applied to novel problems in a modified form.

Finally, the fragility of purely truth maintaining systems was highlighted as a major problem of early AI. By branching out into systems which were better able to express and reason with belief and uncertainty, more robust frameworks were created which permitted a forms of generalisation to novel data and situations by extrapolation from given examples. By moving in this direction, AI systems designers were, perhaps, tacitly acknowledging that the quantification (of membership, probability and belief) inherent in neural network architectures is an important property for the robust design of even symbol systems.

2.3 Artificial Neural Networks

The study of artificial neural networks as a mathematical discipline arose from a belief that it would be possible to create brain-like machines, capable of learning by example and deriving their own internal representations to model the problem rather than having it formally described. The sub-sections that follow describe those neural network architectures which were relevant to the work reported in this thesis, along with a discussion of their advantages and disadvantages.

2.3.1 Basic Concept of a Neural Network

There is no single network which could be described as a ‘typical’ neural network. The term is rather loose and can be applied to a great variety of structures which, on the surface, appear to be very different. This section reviews the core of features common to most neural networks.

The building blocks of network are often referred to as ‘neurons’, despite the occasional complaint that they are not, strictly speaking, realistic models for biological neurons. (They are sometimes called either ‘units’ or ‘nodes’ to avoid this problem but these conventions will not be followed here since it should be clear that the use of the word neuron in terms of a mathematical model does not imply a true representation of all neuronal processes).

Perhaps the most general network structure is the multi-layered perceptron (MLP) in its recurrent form, which embodies most of the concepts that are ubiquitous in the neural network approach (Rumelhart, Hinton & Williams, 1986b). Such a network is illustrated in figure 2-2, overleaf.

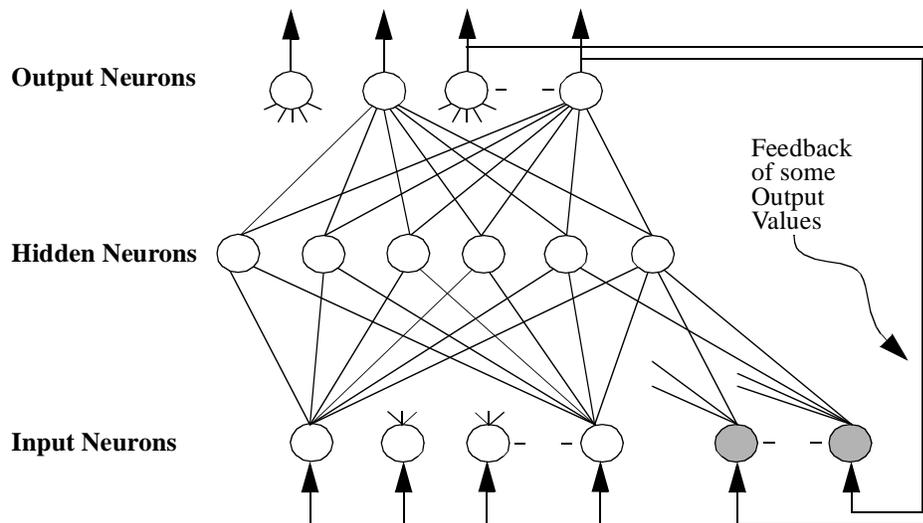


Fig. 2-3 Multi-layered Perceptron, with recurrent output.

In the figure, the network has three layers. The number of ‘hidden’ layers can be increased without limit. Note the shaded neurons at the input which represent fed-back output values. This feedback allows the network to act as a state machine.

Within each neuron, there are two major processes. The first is the potential function, which usually consists of the weighted sum of the inputs. At time, t , the potential of neuron, i , is given by:

$$U_i(t) = \sum_{j=0}^N O_j(t-1) \times W_{ij}$$

where U_i is the potential of neuron i , O_j is the output value of a neuron, j , on the previous level of the network and W_{ij} is the weight value of the connection from neuron j to neuron i . This is a simple mathematical model of the functioning of the dendritic trees of real neurons. A number of researchers have speculated that this model is too simple and that a significant proportion of the information processing occurring in the brain happens in the dendritic trees themselves (Shepherd & Koch, 1990). Such ideas will not be pursued here.

The second process in each neuron is the output non-linearity, or thresholding function. This can take many different forms, depending on the requirements of

the application. The two most common threshold functions are the sigmoid and the binary threshold.

$$\begin{aligned} O_i(t) &= 1 \text{ when } U_i(t) > T_i && \text{Binary Threshold} \\ &= 0 \text{ otherwise} \end{aligned}$$

$$O_i(t) = \frac{1}{1 + \exp(-(U_i(t) - T_i)/K)} \quad \text{Sigmoid Threshold}$$

where K is a constant which defines the slope of the function. The sigmoid function is often used in applications since it is easily differentiated. This permits the network to learn using a credit-assignment learning scheme such as error back-propagation (Rumelhart *et al.*, 1986).

The action of each neuron is essentially to calculate a dot product between its input vector and its weight vector. When both vectors are of known length, the dot product can be used to calculate the angle between them:

$$\cos \theta = \frac{X \bullet W}{|X||W|}$$

where X and W are the input vector and weight vector, respectively and θ is the angle between the vectors. When either vector is of unknown length then the potential of the neuron, and hence its output value, do not provide an exact indication of the similarity between the two vectors. Many applications for neural networks therefore insist that both vectors be normalised (e.g. Kohonen, 1982a).

2.3.2 Unsupervised Learning

The lack of an external source of the network output is the hallmark of unsupervised learning (also called self-organised learning). It is left to the network itself to “decide” how it will represent the probability distribution of the input data. This definition permits a wide range of interpretations which differ according to their criteria for what constitutes “good organisation” and the complexity of the processing carried out by each neuron. Examples of this paradigm include Principal Component Analysis (Jolliffe, 1986) and the modelling of retinal receptive fields (Linsker, 1986).

Competitive learning is a non-linear form of self-organising system. Usually the network is arranged as a set of identical neurons. Each neuron receives the same input vector and calculates its potential as the dot product of the input vector, X and its weight vector, W_i . A subset of the neurons are allowed to “win” the competition to decide which is fittest to represent the vector. The winning vectors are those that have the highest potential. In many applications, the subset of winning neurons is only a single neuron.

The competition phase can be executed either externally, by examining the potentials of the neurons directly, or using the dynamics of the neurons themselves. This latter method uses lateral inhibition between neurons such that the neurons with the highest potential tend to dominate the group. (Such a scheme has implementation advantages for a self-contained neural system).

After each competition, network weights are updated in a manner which depends upon their status: winners or losers. Usually, winning neurons increase the weights by an amount proportional to the input vector, X . By doing so, they are more likely to win the competition for this input vector next time it is presented. To keep the weight vectors within bounds, they are usually normalised so that learning corresponds to a change only in the angle of the winning neuron's vector. Losing neurons either maintain the same weight vector or are increased by only a small fraction of the input vector.

A particular example of this genre is considered in more detail next: that of the self-organising feature map proposed by Kohonen (1982a), building on earlier work by Willshaw and von der Malsberg (1976).

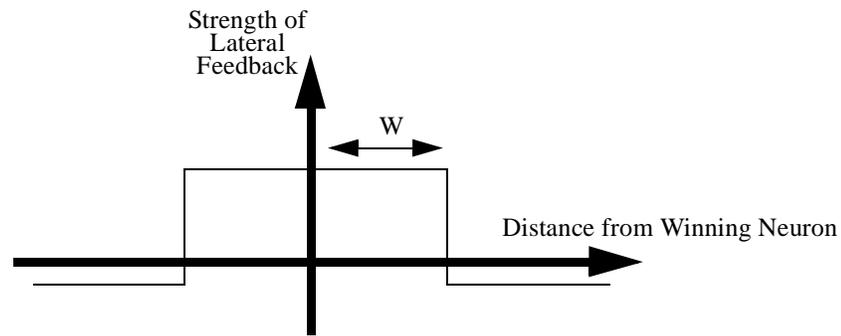
The advantage of self-organisation is clearly the fact that no teacher input is required to specify how the neurons should arrange themselves. The general principle is that each neuron is as important as any other in terms of its representational significance. To ensure that this is the case, each neuron must be capable of measuring its 'importance' in terms of the quantity of information it represents compared to its peers and to take individual action if it is either under- or over-representing. If each neuron is to be able to make such a decision alone (a desirable property for a network with a large number of neural elements) then there must be certain assumptions about the statistics of the representation scheme and the input data to facilitate this.

In self-organising systems this is the case because within each group of neurons there are a fixed number of winners. Each neuron can track how often it is a winner, and make adjustments to its behaviour if it wins either too frequently or too infrequently. But in networks in which any output vector is possible such as an MLP, it may not be so easy for each neuron to determine how they fit into the scheme of things. The *credit-assignment problem* is in evidence once again. In these cases, information external to the neuron is needed.

2.3.3 Self-Organising Feature Maps

As a special case of the self-organising networks described in the previous section, that proposed by Kohonen is an important example of the competitive learning sub-genre. The aim is, once again, to model the probability distribution of the input data; in this case the medium in which this modelling takes place is a lattice (usually in two dimensions) of neurons. Each neuron is fed a copy of the input vector, as before.

The important difference with this network is the complex form taken by lateral inhibition. The neighbourhood function is such that neurons very close to the



winning neuron receive positive feedback from it, while those further away receive negative feedback. In theory, this function takes the form of the so-called ‘mexican hat’ function but for ease of simulation it is approximated by zones of constant value, as shown in the figure overleaf.

The width of the positive feedback zone, W , is usually a decreasing function of the learning trial, t . Thus at the beginning almost all of the neurons are involved strongly in learning, while later on the function becomes much narrower and the network undergoes local fine tuning.

Fig. 2-4 Kohonen network: Lateral feedback vs. distance from winning neuron

Globally, the behaviour of the network is intended to arrange the vectors in the array so that two conditions are fulfilled. First, that the proportion of vectors in a particular region of pattern space is proportional to the probability density function, $p(x)$, of the input population. The second condition is that neighbouring vectors tend to develop similar directions such that, if the “correct” neuron does not win the contest (perhaps because of noise in the input vector) a vector very close to the correct one will. This lends the representation scheme a level of robustness.

However, there are several cautionary points to note. First, the algorithm as it was proposed by Kohonen does not truly model the probability density function of the input vector. It was only with the addition of a “conscience mechanism” (DeSieno, 1988) that the network could achieve this goal. This conscience, as discussed earlier on general self-organising systems, allows each neuron to track the amount of information it is representing and to make adjustments to its participation in each contest to ensure that it is neither dominating nor under-achieving relative to its peers.

The second drawback is that during some learning trials the lattice can develop in a twisted fashion and cannot properly fill the representation space. This can happen when the physical neighbourhoods as defined by the layout of the lattice are at odds with the relative directions of the vectors of the neurons themselves.

But this begs the question of why the physical lattice was present at all. The physical arrangement of the neurons is needed to apply the lateral inhibition function. This function, in turn, ensures that neighbouring neurons had similar vectors which is necessary because the 1-from- N representation scheme has no degree of

redundancy: if the correct neuron fails to fire (due to noisy input, for example) then the resulting output pattern would have no overlap with the desired pattern. The lateral inhibition forces correlations between neighbouring neurons in an attempt to compensate for this.

The addition of conscience within each neuron seems like an appropriate measure for any self-organising system which aims for robustness. Avoiding the problems associated with a pre-defined physical neighbourhood is a more interesting area for discussion. One possible solution is to eliminate the need for a lattice altogether. How could this be achieved?

A scheme in which more than 1-from-N neurons win each competition provide a level of redundancy automatically: consider a system in which K neurons are selected as winners after each competition. If one of the neurons which should fire fails to do so, there are still K-1 others which could fire correctly. Thus for large K the vector of firing neurons is almost exactly as it would be if all neurons fired correctly. This approach removes the need to build in redundancy by way of physical proximity and hence avoids any problems of net 'twisting'. These ideas will be pursued in chapter six when the data coding scheme is discussed.

2.3.4 The Multi-Layered Perceptron

The MLP represented a turning point in the fortunes of neural networks as a subject of serious study, even though the algorithm it embodies (Rumelhart, Hinton & Williams, 1986) was first proposed by Werbos more than a decade earlier (Werbos, 1974). Its power derives from the learning algorithm employed, usually a variant of the error backpropagation algorithm. Using this algorithm, it is possible to train a network containing layers of so-called *hidden* nodes whose outputs are not specified by the external environment during learning. It is the role of the network to select an appropriate output for these nodes and to ensure that they attain them. Error backpropagation is one practical solution of the *credit-assignment problem*, which was first identified as a serious issue in the context of AI research (Minsky, 1961).

The error backpropagation algorithm itself advocates the assignment of blame to a hidden node in proportion to the contribution it has made to each erring output. The assessment of this contribution relies on the use of a differentiable threshold function for the neurons themselves.

Since the algorithm is so well known, its details will not be repeated here. However, several key properties need to be mentioned to provide a contrast with other networks:

Unconstrained mapping. The network can, in principle, learn to associate any pairs of patterns, without restriction except that no two output patterns can be associated with the same input pattern. The linear separability constraint of the single layered networks no longer apply (Minsky & Papert, 1969).

Multiple presentations. The network learns by incrementally adjusting network parameters over multiple presentations of a data set, guided by an energy function. As such, it is not suited to so-called ‘one-shot’ learning in which a pattern can be reliably recalled after a single presentation and learning event.

Convergence problems. The presence of local minima in the energy function can prevent correct network convergence or at least make it extremely slow. A considerable amount of effort has been expended in the research community to resolve (or at least contain) the problem, resulting in a wide variety of MLP variants including the use of a momentum term and individual learning constants for each weight (Jacobs, 1988).

Generalisation issues. The ability of an MLP to generalise has also received much attention in the literature. It is widely accepted that the size of a network relative to the complexity of the mapping task is important in determining its generalisation performance (Barron, 1991).

The MLP, although a powerful and much applied network, is unsuitable for the task to be undertaken here. The reasons for this are largely those given above. The convergence problems and necessity for multiple presentations of a pattern during learning are in conflict with the philosophy which motivates this work: the demand for reliability of recall after even a single presentation.

2.3.5 Non-Holographic Associative Memory

While appearing quite a few years before the multi-layered perceptron, in the sixties, this type of network has been left until late in the discussion because of its relative obscurity. In spite of this, it is an important contribution to neural network theory. The original concept of such networks is due to Steinbuch (1963), but the approach seemed to make little impact even after it was re-interpreted by Willshaw *et al.* in 1969. It is this latter paper that will be considered here. In this context, the network was first put forward as an optical system in which beams of light (or their absence) are the carrier of information. The concepts that it embodies can be extracted and applied to a more conventional (in this case neural) network.

The network has two important properties. First is the ability to learn a new association after a single presentation of a pattern-pair. Second, assuming the patterns are evenly distributed, the capacity of the network (expressed as the number of pattern pairs that can be stored) tends to M , the number of output nodes, under certain circumstances. How these properties are achieved and the side-effects of the learning scheme will be briefly described.

The network learns to associate a set of input-output pattern pairs. Legal patterns are constrained, however. Each input pattern is N bits in length and consists of n ‘1’s and $(N-n)$ ‘0’s. Similarly, each output pattern is M bits in length and consists of m ‘1’s and $(M-m)$ ‘0’s. These restrictions are essential to the processes of learning and recall.

The memory of the network, in which the patterns are stored, is a binary matrix of size $N \times M$. Every entry in the matrix is initialised to zero. During learning, the input patterns, X_i , and the output patterns, O_i , are applied to the input and output nodes, respectively. Entries of the matrix, W_{ij} , are updated according to the following prescription:

$$W_{ij} = 1, \text{ when } X_j = 1 \text{ and } O_i = 1$$

$$= \text{unchanged, otherwise}$$

The figure below shows the learning of a single pattern pair made up of a 5-from-8 input pattern and a 4-from-8 output pattern.

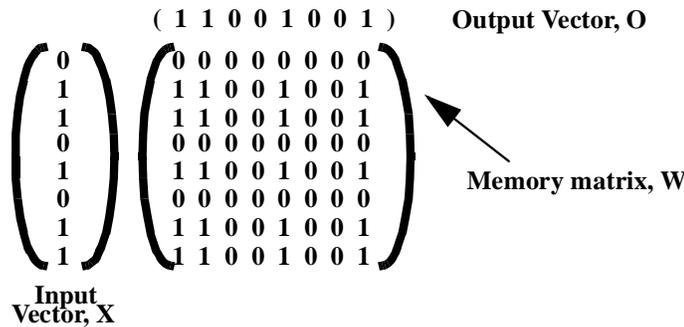


Fig. 2-5 Single pattern learning in the non-holographic associative memory.

During recall, the test input, X' , is applied to the matrix, W , producing a potential in each neuron, U , as given by:

$$U = W \cdot X'$$

The value of the output units is a thresholded version of this potential:

$$O_i = 1 \text{ when } U_i \geq n$$

$$= 0, \text{ otherwise.}$$

where n is the number of '1's permitted in an input pattern.

If the input pattern exactly matches one of the learned patterns, it is clear that each output unit which produced a '1' in the learned output pattern will receive a potential equal to exactly n . It can thus be identified exactly. When the number of learned association is low, it is possible to learn a new pattern in a single update of the matrix, W , which can be immediately recalled without error in this way.

The only limiting factor is noise from other learned patterns. When an input unit produces a '1' it evokes potential into every output unit with which it ever

made a connection during learning. Clearly, the more patterns that are stored in the network, the greater will be the noise during recall.

An output error can occur if any of the output units which should not produce a '1' achieve a potential equal to n . This can happen only if a connection was made to every input node which is itself producing a '1'. But each such connection is made with probability h , as given by:

$$h = 1 - \left(1 - \frac{nm}{NM}\right)^Z$$

where Z is the number of learned patterns. Thus, the chance of all n of the '1'-valued input units making such a connection with a '0'-valued output unit is hn . Clearly this probability tends to 1 as Z tends to infinity.

For high Z , it is necessary to keep the fraction of '1's in the input and output vectors low. These ideas will be quantified in a later chapter, since the principles of this network form an important basis of the work described in this thesis.

The network has a number of significant advantages. Most importantly, the network will learn a new pattern pair after a single presentation. Secondly, the capacity of the network is high compared to many other networks (such as a Hopfield network, see later). For a network of M output neurons, the capacity approaches M . This is clear when one realises that each output neuron is calculating, in its potential, the correlation it has with all of the input patterns for which it was supposed to produce a '1' output. The network will only saturate when every one of its synapses is set to '1' which, for large M , corresponds to the case when the number of learned associations is also M . Hence, the storage capacity of such a network tends to M .

However, the network also has some drawbacks. First, the fact that weight values are set to one during learning rather than being increased in an additive manner means that the frequency of presentation of a pattern is not recorded by the network. Thus, a pattern pair presented 1% or 50% of the time would produce a network with the same weights. The second drawback is that, since the input/output pairs use a representation scheme with a few '1's and the rest '0' (known as *sparse coding*), the information content of each vector is usually less than that of an unconstrained binary vector of the same length.

Thirdly, there is no hidden layer in the network. As a consequence, the network will suffer the same inability to learn linearly inseparable mappings than other single-layered networks (Minsky & Papert, 1969).

Overall, the inherent controllability, predictability and high storage capacity of the network make it a strong candidate for the work ahead. The ideas it embodies were developed to try to avoid the disadvantages described here.

The next chapter extends this review by considering developments in correlation associative memory (see section 3.1, page 47).

2.3.6 Hopfield Networks

All of the networks considered so far performed pattern-mapping. The input vectors were either given or an input probability distribution specified. The output vectors were either given or the system was allowed to generate its own representation scheme according to some criterion such as maximising the information content or modelling the input distribution.

The network architecture proposed by Hopfield is of a very different kind. It is an auto-associative network with a theory couched in the statistical physics of spin glass systems. It has been applied to a range of cost-minimisation problems, one example being the classic ‘travelling salesman problem’.

In his original prescription, Hopfield used noiseless binary neurons (Hopfield, 1982), but this was later extended both to the noisy case and to the continuously-valued neural output case (Hopfield, 1984a). The basic structure is essentially the same, however. The N neurons in the network are fully interconnected. A new pattern is learned by adding to each weight an amount proportional to the product of the outputs of each pair of neurons. For binary neurons, whose outputs are either $+1$ or -1 , this means that weight is added between neurons which produce the same output and is subtracted between neurons that produce complementary outputs. One special case is that no neuron is permitted to make a connection with itself. The resulting weight matrix is always symmetrical which is important in the study of its properties.

During recall, a noisy or partial pattern is set up on the neurons and the network is allowed to iterate alone. In the noiseless case, the network will always make changes in its state that reduce the energy of the system.

Empirically, Hopfield found that for zero-mean, uncorrelated N -bit patterns, the total capacity of an N neuron network was less than $0.15N$ which was very low. Analysis by Amit on the noisy case showed explicitly by way of mean-field theory that the actual figure was closer to $0.138N$ (Amit, 1989).

Despite the seeming low capacity, the Hopfield network is important for several reasons. First, it has applications in non-linear constraint satisfaction problems which are NP-complete. Second, a pattern can be learned with a single modification to the weight matrix. Next, an associative memory can be made by using one half of the neurons as the input and the other half as the output which are initially ‘blank’ but develop appropriate values during computation. Finally, its properties of learning and recall are well understood, using concepts from statistical mechanics.

In addition to its low memory storage capacity, the Hopfield network suffer from other problems (described by Amit, 1989). First, the basins of attraction around each minima are not of equal width in each direction. Thus, a network which starts off in a state close to that of a stored vector is not guaranteed to move towards that vector even in the noiseless case. Second, even though noise may be added to help destabilise the local minima, this reduces the overall speed of convergence of the network and makes it impossible to guarantee that the network will

end up at the lowest point of the well in which it began. This results in unreliable recall. Next, during recall the network can get caught in local minima. These can be superpositions of stored memories but in some cases may be unrelated to any stored pattern, so called ‘spin-glass states’.

The review of Hopfield networks is extended in the next chapter to consider the issues and limitations of such networks as well as to present more detailed analyses of their capacity (see section 3.2, “Recurrent Memories”, page 60).

2.3.7 Summary of Artificial Neural Networks

This section has discussed a variety of neural network architectures, but which make up only a small proportion of the population. Each embodies a number of central ideas.

Most fundamentally, in neural networks information is stored by connections between neurons and is represented by the firing of the neurons themselves. Learning consists of some form of synaptic modification between neurons, to develop neurons which are tuned to identify useful features in the input and to make stronger the connection between features which are statistically correlated.

Next, it was noted that in feedforward type networks, the learning processes often demands that every pattern be repeatedly shown to the network over many learning trials to achieve convergence.

In general, learning a new pattern tends to erode previously stored data, unlike symbol systems where the storage of new data tends not to erase old data during the storage process itself. The interdependence of stored patterns is much stronger in a neural network than it is in a conventional AI knowledge database such as was described earlier.

2.4 Artificial Intelligence & Neural Networks

All the discussion so far has centred around one or other of the major disciplines of connectionism and AI, describing some key concepts in each field and the problems left unsolved. Even from the limited selection of examples presented in this chapter, it seems clear that far from being competitive in nature the two disciplines in fact complement each other well; each provides possible solutions to remedy the drawbacks of the other. This fact has not gone unnoticed in the research community at large and this chapter continues with a review of the debates that are moving towards a unified theory of neurons and symbols.

2.4.1 The Proper Treatment of Connectionism?

It is likely that the idea of merging the AI and ANN approaches has been around for as long as the individual disciplines themselves. However, progress towards such a goal seems to have begun when Paul Smolensky produced his *magnum opus* (Smolensky, 1988) in which he tried to establish connectionism as the

only true medium for a constructive theory of how mental representation could work. The scope of his attack was more than just the true nature of human thought processes, but rather the whole philosophy of knowledge representation and manipulation by ‘intelligent agents’.

The irony of the debate is that Smolensky’s aim at that time does not seem to have been one of proposing a merging of the disciplines but rather one of defending the whole concept of neural networks against certain prominent members of the AI community who claimed that connectionism (the cognitive science branch of neural networks research) was fatally flawed and would only be capable of producing an intelligent machine by essentially implementing a Classical AI machine using neural elements.

But by proposing that a neural network was far more than a medium of implementation for a symbol system and indeed offered the *only* true explanation of mental computation, Smolensky seems to have prompted a critical re-examination of the differences between AI and ANNs which (if it occurred at all) had been a far less common activity before the opening volleys of the debate.

The loudest voice of Smolensky’s opponents belonged to Jerry Fodor, author of ‘The Language of Thought’ (1975) and a cognitive psychologist with a long and distinguished track record in the symbolic computing arena. In his reply to Smolensky’s work (Fodor & Pylyshyn, 1988) he was unswerving in his belief that the classical AI approach offered the only currently plausible explanation of thought and cited four properties that (he claimed) any mental representation must have in order to qualify as a candidate for a language of thought. Not one of these properties (he claimed) were possessed by any neural architecture yet proposed.

Despite some glaring simplifications made for the sake of argument, the Fodor & Pylyshyn paper effectively set the standard by which any future neuro-computational architecture would be assessed. Smolensky has continued to refine his ideas of sub-symbolic computation but it is always to this same yard stick that he returns for re-assessment. Since it is relevant to the architecture development, the points made by both sides will be briefly reviewed.

2.4.2 Smolensky’s Argument for ‘Sub-Symbolic Computation’

In writing this paper, Smolensky stated that while insufficient research had been carried out into the connectionist approach to prove its capabilities as the substrate for mental computation, nevertheless the stage *was* set to state what the goals of such research should be. In every sense, he was staking a claim, but with the caveat that there was little supportive evidence at this time to justify it. His argument could be summarised thus:

Knowledge can be encoded in a wide variety of ways, as befits the problem at hand. In verbal and written communication, knowledge is encoded as strings of punctuated words: language. Due to our intrinsic ability to formalise knowledge in this way and to manipulate it mentally, we could regard the human mind as a ‘conscious rules manipulator’. Indeed, it is in this way that AI has traditionally modelled all aspects of human cognition.

However, in addition to the notion of mind as an entity for explicit manipulation of rules, there is second class of ‘thought’ which does not (at the conscious level) appear to involve such formal methods. Examples of such processing are animal behaviour and non-verbal processing (such as being able to reach a conclusion without being able to describe the mental processes which led there). This level, using Smolensky’s nomenclature, is the domain of the ‘intuitive processor’.

Where Smolensky and traditional AI begin to diverge is in the handling of this intuitive processor. By the AI account intuitive processing is just another example of the application of formal rules. For Smolensky, this was not acceptable. He drew a line between two levels of knowledge representation and then between two styles of processing which describe the computation performed at each level. First, he defined the *conceptual level* at which individual units in the machine may represent an object or entity in the outside world and to have the semantics of natural language. This has traditionally been regarded as the domain of symbolic computation.

Next, he defined the *sub-conceptual level* in which a unit can represent only a part of an entity in the task domain; each entity is represented as a distributed pattern of activity over many units. Computation is defined as sub-symbolic and is characterised not as the manipulation of strings by formal rules but as the evolution of the state of a dynamic system. Essentially what he describes are just the standard descriptions of distributed representations and processing.

Smolensky asserted that this lower, sub-symbolic level of computation is the only way of accurately describing the mechanism of the intuitive processor. Indeed, he went further: extrapolating its capabilities and aiming to promote it as the best account for the rule-based symbolic level too, since it would account not only for rule-following but also for the apparent rule breaking seen in a century of psychological experimentation. The basis of this attack was that a rule based approach often fails to exactly capture the full computational picture in any given modelling problem from psychology. If the symbolic account is only approximate then it can hardly be an explanation of how the mechanism actually works, he argued. When re-interpreted as a set of interacting sub-symbolic units, however, the representation can accurately describe the processing and thus is an adequate explanation of it.

Smolensky provided no proof of his theories, instead going beyond the mere refutation of the claim that the Classical approach is the only valid one by postulating that the *neural* approach is the only valid one, which is also perhaps too strong a claim. However, when taken as a statement of where we intend to go with the neural paradigm it surely has considerable merit. The error of judgement in rejecting Smolensky’s argument for the significance of the sub-symbolic nature of mental computation would be comparable to the dismissal of neural networks in general on the grounds that they were incapable of solving the XOR problem (Minsky & Papert, 1969).

The reply to Smolensky’s paper casts considerably more light on the nature of the problem itself.

2.4.3 Fodor & Pylyshyn's Requirements for Mental Representations

The bulk of Fodor and Pylyshyn's argument against Connectionism was summarised in their paper presented in *Cognition* (Fodor and Pylyshyn, 1988). In essence, they argued that there was no current defined connectionist architecture that could be made to account for mental representations and their manipulation. They posited four properties that any representational scheme must have to be a valid candidate as a medium of thought. Once again it is reasonable to extend these arguments beyond the single mechanism of thought itself, to any mechanism for general knowledge representation and manipulation.

2.4.4 Productivity

This refers to the ability of the representational encoding to handle relations of arbitrary complexity. Formal grammars can be used to generate strings of unlimited length from a finite set of terminal and non-terminal characters and a finite set of rules to concatenate and manipulate them (Chomsky, 1965). The only limit on string length is the finite memory of the machine which, as highlighted by Chomsky, is a question of *performance* rather than of *competence*.

By way of contrast, Fodor & Pylyshyn argue, current connectionist models are extremely limited in their productive capability. Productivity arises because it is possible to build complex things out of simple parts which is something that they claimed could not be done with any known Connectionist network.

In fact, they are really making two assertions. First, that to be productive there *must* be a combinational syntax to establish relationships between the elements in a complex expression. Second, that if a neural network were to be designed which was capable of creating complex expressions out of simpler ones, it would merely be implementing a Classical architecture and that the neural aspects were mere detail.

It is reasonable to agree with the first statement; that combinational syntax is important for a generic symbol system. Thus, the network should embody a combinational syntax. However, the second assertion is not acceptable since, as Smolensky argues quite reasonably, there are many aspects of the mechanism of mental computation which are not explainable at the symbolic level. Chapter five will attempt to clarify many of those aspects.

2.4.5 Systematicity

Here the argument is that the ability to represent a certain state of affairs should also imply an ability to represent other states. To quote Fodor & Pylyshyn's own example, the ability to represent:

'John loves the girl'

should reasonably imply the ability to represent:

'The girl loves John'

This is the case with data structures which have a compositional syntax but is not necessarily true for a neural network in which the ability to represent ‘John’ as the subject of the sentence implies nothing about its ability to represent ‘John’ as its object.

At the heart of this argument is the ability to perform *variable binding*. In this case, the token John is bound either as subject (in the first example) or object (in the second). Fodor & Pylyshyn are claiming that variable binding is essential for knowledge manipulation, that neural networks are not capable of it and that if a neural network could perform variable binding then it would be merely implementing a Classical architecture.

It is reasonable to admit that systematicity is an important property that a network should display. However, to claim that variable binding is not possible in neural networks is an unreasonable assumption. Several examples now exist (for example that of Sun (1994), which show that variable binding is possible, although examples were perhaps harder to find at the time that the critique was written.

2.4.6 Compositionality

This issue is closely related with that of systematicity. It states that the individual symbols retain their intrinsic meaning regardless of how they are arranged. So, to quote their example once again, the meanings of ‘John’, ‘loves’ and ‘the girl’ are the same in both sentences:

‘John loves the girl’ and

‘The girl loves John’.

The difference in semantics is due to how each symbols stands in relation to the others, rather than any changes in the meaning of each symbol alone. Having fulfilled the requirements for productivity and systematicity, a neural network would almost certainly meet the requirements for compositionality.

2.4.7 Coherence of Inference

This is the weakest of Fodor and Pylyshyn’s four arguments since it is based largely on the attack of a fairly non-representative member of the connectionist population, a network for logical deduction, an example of which is shown below:

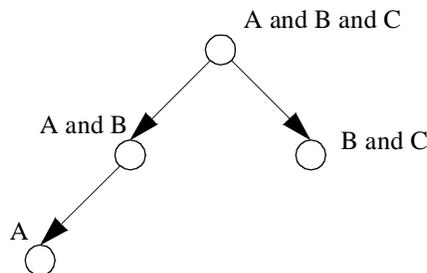


Fig. 2-6 Fodor & Pylyshyn’s argument for Coherence of Inference.

A firing node represents a logic deduction, as indicated by the node label. An arrow indicates a positive connection that will cause the child node to fire if the parent node fires. Thus, activation of the node labelled 'A and B' will cause the node labelled 'A' to fire, indicating that the fact that 'A AND B' is true also implies that A is true.

Fodor and Pylyshyn's argument here is two-fold. First, that the deductions which can be made using this network depend only on the way it was constructed. For example, no node labelled 'B' appears from the node labelled 'A and B' even though B would be a logical conclusion of 'A and B'. It would seem that the conclusions are only exhaustive if the network was exhaustively specified at the beginning.

Their second argument is that the causal nature of the network is, in reality, independent of the labelling. If the label 'A' on the lowest node was changed to 'D' then the network would behave the same, even though it were making an incorrect deduction, logically speaking. The problem is that the meaning of each node, as given by the label that it carries, does not *causally* affect the processing itself.

Clearly, the example they use is not a neural network in its more general sense. It is the explicit tokening of variables and expressions using variables in an arbitrary manner which permits this problem to occur.

Despite the flaws in this argument, the idea of inferential coherence does illustrate a number of key concepts. First, that the meaning of a pattern of activity of neurons can only be specified and understood in terms of the causal effect it has on the evolution of the state of the system. Appealing to labels or other baggage which is not *causally related to processing* is to add meaning where it is not. Second, that by assigning meaning to a particular pattern produced in one step of processing, one has made an obligation to ensure that this assignment is consistent in every other interaction it may take part in. This is a consequence of the fact that its meaning lives or dies by the pattern itself, which was assigned in one context but may be used in others. This crucial point strongly influences the symbol encoding chosen for this work, which is described in chapter five.

Since 1988 there have been developments. The whole debate of how to use neural networks in a symbol environment is now receiving a great deal of attention in areas outside of philosophy and psychology (see for example Honovar & Uhr, 1994).

The next two chapters will discuss in depth the major areas of research into neuro-symbolic systems (section 3.3, "Coding and Representation of Symbol Structures", page 67 and section 4.0, "Introduction", page 85).

2.5 Conclusions

This chapter has presented a brief overview and critique of the existing ideas which are relevant to this work. These ideas were drawn from the areas of neuroscience, Artificial Intelligence, Artificial Neural Networks and hybrid architectures.

It emerged that in the area of unified neuro-symbolic systems there exists a potential for significant progress towards systems that combine the best features of both the Classical and the Connectionist approaches.

From the Classical AI account, we gain the ability to construct and manipulate knowledge structures of arbitrary complexity and size; to decompose problems into manageable sub-problems and handle the intermediate results; to explore possible solutions by rearranging sub-parts and then evaluating the resulting whole; to perform deductive inference; and to learn new data in a single trial without disrupting the ability to recall other, unrelated, data.

From the Artificial Neural Networks account we gain the ability to statistically extract relevant features in the input data without supervision; to address stored data by content rather than by index; to have graceful degradation in performance given noisy or incomplete data; to generalise from existing data (inductive inference by interpolation); to store correlated data efficiently; to use noise and non-linear optimization techniques in the evaluation of potential solutions to a problem.

In combining the two approaches we hope to eliminate (or at least reduce) the effects of several key problems which persist in either solution alone. In the Classical AI account the key problems are the prohibitive amount of resource needed to search a huge knowledge base for relevant data or problem space for a valid solution; the prohibitive amount of processing resource needed to keep a very large database consistent; the fragility when faced with new situations, due to poor generalisation capabilities.

In the Artificial Neural Networks account, the key problems are the inability to represent data structures of arbitrary complexity; the inability to manipulate data structures using control strategies as complex as those found in symbolic algorithms; the inability to focus on one part of its database at a time, bringing only a subset of its knowledge to bear on a problem at any one time; the lack of a good solution to the variable binding problem; the degradation of stored memories with each new learning cycle.

In a qualitative sense, one might say that the problems associated with AI systems are based on the fact that data items are too static: each has a pigeonhole and does not change unless something changes at the symbol level. This makes it more difficult to use the information; for a piece of information to be used in any situation it must be explicitly accessed. Keeping the knowledge base coherent thus requires an investment of resources; the system could hold two contradictory facts and must expend 'effort' to find and eliminate such inconsistencies.

But the ANN approach has the opposite problem. There appears to be relatively little stability for its stored knowledge (every learning event helps to degrade the storage quality of previously stored patterns) and it is difficult to control how (and if) such knowledge is applied to each recall event. However, as each new pattern is learned the network is able to make minute adjustments to the way it encodes all stored patterns so that there exists the potential to develop an underlying unity which can be exploited to keep the database unified.

In short, a hybrid architecture should not be thought of as an unnatural mixture of symbols and neurons. That is a superficial detail. The assertion which underlies this work is that the fundamental issue is one of finding the right balance between the *separation, stability and flexibility of association* of data items on the one hand, and their *computational integration and mutability* on the other. Furthermore, the philosophy presented here is that an architecture which achieves the right balance, be it implemented in 'neurons' or PROLOG, is the architecture which will form the basis of systems that can finally rival the intelligence of humankind. The results to be presented in later chapters represent steps towards finding that balance.

The next two chapters continue the review by considering in more depth the details of key areas: correlation associative memories and the implementation of reasoning systems in neural networks.

Detailed Review I: Coding and Storage Issues

3.0 Introduction

The survey presented in chapter two represents an overview of the topics that were considered at the outset of the work presented in later chapters. This review now continues with a more in-depth examination of particular elements that are most directly related to the contents of the thesis.

The in-depth review is divided into two chapters due to its length. The first of these looks at neural coding issues, including associative memories and approaches to coding symbol structures as neural representations. The second chapter of the detailed review will examine the topic of reasoning systems, particularly architectures that can be implemented as neural networks.

This chapter, then, is divided into three major sections. The first section looks at the current literature on connectionist associative memories and issues in pattern encoding for such memories. Important properties such as retrieval and storage capacity are considered and will provide a basis for comparison with the network to be developed in later chapters.

The second section in a similar vein to the first, is dedicated to the recurrent associative networks, principally the Hopfield network but also some variants such as the bi-directional associative memory and the Boltzmann machine.

The third section returns to the theme of coding in neural networks, but addressing the more complex issues involved in representing symbol structures. It examines a variety of approaches and attempts to identify the benefits and limitations of each.

A final section presents a summary of the chapter, highlighting the key points that will be drawn upon in later chapters.

3.1 Connectionist Associative Memories and Coding Issues

In this section, the literature on connectionist associative memories will be reviewed in greater depth than the cursory review of chapter two (section 2.3.5, page 34 and section 2.3.6, page 36). This review begins with a discussion of the different ways of categorising such memories, before considering individual contributors who have provided key results in this area.

The review will begin with the simple feedforward approaches typified by Kohonen and Anderson. Several memories based on the non-holographic associative memory of Willshaw *et al.* will also be considered as well as the sparse memory approach of Kanerva. The next section will consider memories with feedback such as the Hopfield network, the Boltzmann machine and Kosko's BAM network.

In all cases, the objectives will be to consider the parameters which are important for characterising the memory, how each memory compares in terms of those parameters, and what trade-offs were available to the designer to achieve his or her goals.

3.1.1 Categories of Associative Memory

All associative memories share a common basic structure and set of aims. In each case the goal is to associate an input and an output vector together using a matrix of connections, or weights. The values of the weights are such that when the input vector is re-applied to the network, the associated output is produced. Normally many such input-output pairs are stored within the same matrix so that on any given recall the output calculation is affected not only by the desired pattern but also by other unrelated activity coming from the other stored vector pairs: cross-talk. A major part of the design of such memories is to extract the desired vector from the memory while suppressing or even eliminating the crosstalk.

We can classify the many types of associative memory according to a number of categories, the most important of which (from the point of view of this work) will be briefly summarised.

Feedforward or recurrent? In feedforward memories, the input pattern elicits a single wave of activity through the network and an output vector is produced once. In recurrent networks (such as the Hopfield and BAM networks to be described in the next section) the output vector is fed back to the input and influences future network activity.

Iterative or non-iterative recall? During the recall process, some networks allow the output to develop over time by an incremental scheme. This iterative scheme contrasts the alternative approach where the outputs immediately achieve their final values. The former method is more compute intensive, but (by the same token) makes more complex recall operations possible.

Hetero-associative or auto-associative? In hetero-associative networks, the input and output patterns are permitted to be different. They may even have differing dimensionalities. In the auto-associative case, a pattern is associated with itself.

Binary or bipolar vector encoding? The choice of either $\{0, 1\}$ or $\{-1, +1\}$ as the set of valid outputs for any neuron impacts the signal to noise analysis and the learning algorithm.

Vector encoding? The rules which define what constitutes a valid vector. Some memory architectures are more restrictive than others. Choices for vector encoding include 1-from-N, M-from-N and the so-called 'standard' encoding (which allows roughly 50% of neurons to be 'on' and the other 50% to be 'off').

Layer structure? Some memories have only a single layer of neurons, while others permit multiple layers. This choice is a trade-off between required resources and flexibility since adding more layers allows more complex transformations to be made with a single network.

Matrix element? The designer may choose either binary elements for the weight matrix or continuous. An intermediate, quantised, element has also been used to allow the memory capacity as a function of matrix element information capacity to be assessed.

Synapse modifications during learning? Here we distinguish between learning algorithms that set the synapse to a pre-specified value (such as '1' for the Willshaw memory) and those that add to or subtract from the current value (such as for the Hopfield network).

Learning algorithm locality? Some learning algorithms make synaptic modification based on locally available information (such as occurs in purely Hebbian learning). Other algorithms draw on non-local information, perhaps requiring the whole pattern set to decide on each synaptic value (such as the pseudo-inverse prescription). Usually it is the latter type that produces the best results.

Pattern modification during learning? A small number of proposed algorithms actually modify the patterns that they are storing to make them easier to store and retrieve.

These are the most important distinctions that will be made in this review. The rest of this section will look at correlation matrix memories. Discussion of the recurrent memories is left until the next section.

3.1.2 Correlation Matrix Memories

While the idea of associative memory has existed since antiquity, it was perhaps the work of Hebb which triggered the search for a mathematical treatment of memory (Hebb, 1949). It took about twenty years for work in this area to really start to bear fruit, leading to the Steinbuch memory model called the 'Lernmatrix', in the early sixties (Steinbuch, 1963). This, in turn, led to the Willshaw non-holo-

graphic memory a few years later. A different approach to the learning scheme led to the correlation memories of Anderson and Kohonen in the early seventies and we continue by considering these models (Anderson, 1972; Kohonen, 1972) in more detail.

In both models, a set of vector pairs, P , are to be stored in a matrix of memory elements. Using Kohonen's naming convention, the set of output or data vectors, $\mathbf{x}^{(P)}$ are associated with the set of input or key vectors, $\mathbf{q}^{(P)}$. This is achieved via a matrix of elements, $\mathbf{M}_{\mathbf{xq}}$, constructed using the sum of outer products relation:

$$M_{ji} = c \sum_{p \in P} x_j^{(p)} \cdot q_i^{(p)}$$

where c is a constant. Recall of a particular data vector is achieved by applying its associated key vector thus:

$$x_j^{(r)} = \sum_i M_{ji} \cdot q_i^{(r)}$$

If the key vectors are orthogonal, the data vectors can be recovered exactly. In the case where the key vectors are non-orthogonal there is interference between the output vectors, producing noise at the output. From this common starting point Kohonen and Anderson's investigations took different directions. Kohonen considers the possibility of incomplete connectivity by introducing sampling coefficients, S_{ij} , into the matrix, permitting a fraction of the matrix elements to be set to zero at random. He showed for number of general situations that the number, s , of matrix elements not set to zero was proportional to the number of components (i.e. the dimensionality) of the output vector.

Finally, Kohonen analysed the case of essentially moving the position of the s non-zero elements in the matrix by an iterative process in order to maximise the mean signal term during each recall. This idea is interesting but it is difficult to see how it could be used in a large network such as will be developed here, especially one in which connectivity will be specified in advance.

Anderson's model was slightly different in that he had two groups of neurons, with each neuron in the first group making a fixed number of connections, M with neurons in the second group. The learning prescription was once again the sum of outer products. He showed that the signal to noise ratio for his network was proportional both to the number of neurons per group, N , and to the number of connections per neuron, M , but inversely proportional to the number of stored patterns, K .

The common failing of these linear networks is well known: the lack of non-linearity and (as a consequence) the inability to add a useful second layer of neurons needed to implement more complex mappings. Even though the means to address non-linearity were first developed around this time (Werbos, 1974) it was more than a decade before the principles involved were brought to the attention of the neural networks community (Rumelhart & McClelland, 1986).

The outer product learning rule is local since it is based on the correlation between pre- and post-synaptic activity, something that is plausible and straightforward to implement. This point should be noted for comparison with later networks which often used more complex, non-local learning rules that can produce better results but with the cost that they are difficult to implement for large networks (see later in this section).

In the early seventies, perhaps because the ideas involved were so new, there seems to have been a number of approaches to the modelling of associative memory that differed in the level of neurological realism incorporated into the model. The Kohonen approach was to strip away almost all of the biological aspects, leaving a problem that was essentially one of extracting signal from noise. Anderson's model was similarly constructed, although he was at pains to discuss the biological realism of his proposed architecture. At the other extreme were models typified by Little & Shaw that took into account the functional implication of many of the biological processes involved in real neurons and their synapses such as the probability distribution for the release of vesicles and the modification of neuron thresholds (Little & Shaw, 1975).

The Little and Shaw model was couched in the statistical physics of spin glasses and seems to be a forerunner of Hopfield's auto-associative network. Their paper is particularly interesting because it tries to deal with multiple levels of memory permanence. As well as the short-term memory (represented by the reverberating pattern of activity) and the long-term memory (permanent changes in synaptic strength and number) they also model an intermediate level of memory based around temporary changes in synaptic value. The idea of multiple levels of permanence in the memory is a key concept in later chapters in this thesis, but seems to have received little attention after Little & Shaw's work.

By the early eighties, more varied models of associative memory began to appear. One important model was of course the recurrent Hopfield network, first presented in the last chapter and of which more will be said the next section. Another was the multi-layered perceptron of Rumelhart *et al.* (1986), the limitations of which act as a catalyst for much of the work in this thesis.

We turn now to some of the work that has appeared on variants of the Willshaw memory, analysing capacity and considering the recall and learning trade-offs for a range of different coding schemes and architectural modifications. The first of these by Baum, Moody and Wilczek, seems to be motivated by the issues in implementing an associative memory in VLSI, although it provides a thorough review of some of the storage issues in such networks (Baum *et al.*, 1988).

They considered a number of possible internal representations for a two-layer network, with N-bit input and output vectors and an intermediate layer of G bits. Like the Willshaw model, the network was composed of binary synapses, which all begin as zero but are set to one whenever both the pre- and post-synaptic neurons fire simultaneously. Their network used polar inputs (+1 and -1) which permitted a zero input to act as a don't care signal. Their analysis of storage capac-

ity could therefore include recall from a partial input, consisting of b bits out of the total of N .

They also considered the precision of each synapse (a major concern for a VLSI implementation of such a network, where fabrication tolerance of resistors could introduce significant errors in the potential of each neuron).

They show that for 1-from- G encoding in the intermediate layer, such a network can recall any stored pattern given a unique fragment but that fan-out and fault intolerance place limits on the capacity and reliability of such a scheme. While they state that a neuron only needs to connect to $\log M$ other neurons (where M is the number of stored pattern pairs) to recall any of M patterns, this leaves no room for error or for system faults, so should be treated as an absolute lower limit.

For distributed representations they looked at s -from- G encoding, once again with N -bit input vectors in which only b bits were specified during recall. They showed that the number of bits, b , that must be specified to activate a particular pattern in the hidden layer is:

$$b \geq \log M + \log \frac{1}{\epsilon} + 2\alpha \sqrt{bMs/G}$$

which must be solved iteratively for b . Again, M is the number of pattern so the first term is the number of bits required to uniquely identify the input pattern. The quantity $1-\epsilon$ is the probability that the b bits of the pattern are unique, so the second term represents the number of bits to add to b to attain that expected probability. The third term takes into account the fact that extra bits may be required to attain the signal to noise ratio, α , demanded for the desired level of confidence in the result.

They go on to show that the number of stored pattern pairs can exceed the number of intermediate neurons, G , for sparse coded vectors. Turning to binary vectors, they show that the synaptic efficiency (defined as pattern bits stored per synaptic bit) reaches a maximum when half of the synapses are set to '1', which occurs when the number of bits set to '1' in each vector, s , is given by $s \approx \log \frac{1}{\chi}$ where χ is the fraction of firing neurons in each pattern and the log is base two.

Moving on, the work of Nadal & Toulouse was to consider a single layered Willshaw memory, and its recall characteristics both below and above the saturation limit (Nadal & Toulouse, 1990). They also put forward the idea that the capacity of the network should be measured not only in terms of the total number of patterns stored but also in terms of the information stored in each pattern. This is important for sparse patterns where the actual information stored per pattern can be low compared to the number of bits in the vector.

After deriving the information density of a sparse coded vector in terms of the fraction of '1's, they go on to show that for a given proportion, q , of synapses set to '1', the number of bits stored per binary synapse, i_w is given by:

$$i_w \ln 2 = \ln q \ln(1 - q)$$

which has a maximum at $q = 0.5$, in line with the result of Baum *et al.*. Furthermore, as the number of stored pattern pairs is increased beyond that point, the error rate of each stored pattern increases, but there is no catastrophic failure of the memory as is seen for the Hopfield network. This region of operation they call the error-full regime since there are almost certainly some errors in every recalled pattern. A graph of information stored in the network versus the proportion of synapses set to '1' is reprinted below.

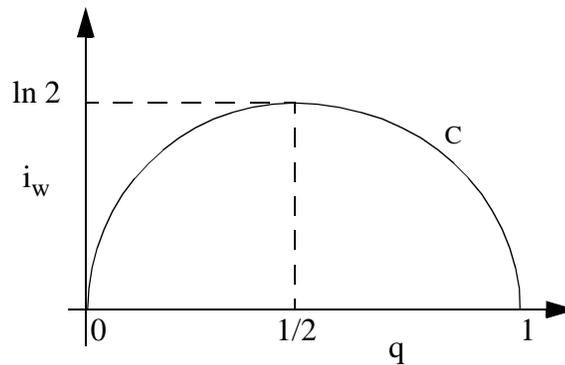


Fig. 3-0 Storage density vs. proportion of active synapses, Nadal & Toulouse (1990)

They show that even when operating in the error full regime, where some output bits have errors, that the information storage per bit, i_w , is still given by

$$i_w \ln 2 = \ln q \ln(1 - q)$$

so there is an appealing symmetry in the rise and fall of the storage density as q varies from zero to one. The optimal capacity, P_c , of such a network (if a single bit error in any pattern is allowed) is shown to be:

$$P_c = (\ln 2)^3 N^2 / (\ln N)^2$$

As a further line of inquiry, Nadal & Toulouse investigate a Hebbian learning rule, where synapse values are incremented when there is simultaneous pre- and post-synaptic activity. The binary synapses of the original Willshaw model, are therefore replaced with multi-levelled synapses with no upper bound on their value. An extra complication is that the threshold, while fixed for the Willshaw model, is now a non-trivial function of the number of stored patterns. They show that the synaptic efficiency for the Hebbian scheme in the error full regime tends to the constant value $1/\pi$ bits, a very different result to that of the Willshaw model

where the value tended to zero in the limit as q approached one. They attribute the variable threshold as the probable cause of this result. A more likely explanation is that the unbounded range of values for the synapses means that the values do not saturate as they would in the binary Willshaw network.

A third example of further developments to the Willshaw memory is the ADAM network (and its descendents) (Austin & Stonham, 1987; Austin, *et al.*, 1991). The ADAM network is based on the Willshaw associative memory but has been enhanced to address the practical use of the memory in a number of environments (principally object recognition, in which the data can be noisy or the object occluded). The authors made two major enhancements to the Willshaw network.

The first change was to the threshold procedure, which in the Willshaw network consisted of setting the threshold for all neurons equal to the number of active bits in the key vector. The Willshaw procedure is shown to lack robustness when faced with noisy key vectors. ADAM uses an ‘L-max’ procedure, in which the L neurons with the highest potential are permitted to fire. This turns thresholding from an absolute into a relative procedure, with all neurons competing against one another for the right to fire. Austin and Stonham showed that this approach is more robust than the Willshaw memory in a noisy environment.

The second enhancement in ADAM relative to the Willshaw memory was to introduce a second layer to the memory and to introduce a set of intermediate key vectors between the two stages. These so-called ‘class patterns’ effectively clean-up the signal inherent in the noisy key vectors. During learning, each key vector is associated with a randomly generated K -from- N class vector in the first memory and this same class vector is associated with the output (or teach) vector in the second memory. Austin and Stonham demonstrated that a small, two stage memory can produce the same results as a much larger single stage memory, representing a significant increase in the efficiency of resource usage.

Further developments of ADAM resulted in an architecture for symbolic reasoning. This is discussed later (see section 4.4.1, “Austin’s Associative Memory for Reasoning (AURA)”, page 108).

As a final example of further work based on the Willshaw associative memory, we consider work by Casasent and Telfer(1992), who compared a range of coding methods and learning algorithms for associative memories. The first layer of a multi-layered system was used to investigate the properties of maximum storage and recall accuracy using the target algorithms. This layer was made up of N -bit input vectors mapping to K -bit output vectors via an $K \times N$ matrix, \mathbf{M} . In all cases, the algorithms assigned values for the matrix coefficients of \mathbf{M} in the equation linking the set of input vectors \mathbf{X} to the set of output vectors, \mathbf{Y} , which is $\mathbf{Y}=\mathbf{MX}$. As well as the storage capacity of the network, the performance in noise was also evaluated to assess the robustness of each learning method for real world applications.

The direct storage nearest neighbour algorithm (DSNN) acted as the baseline algorithm for comparison purposes and is based on a similarity metric together

with a winner-takes-all threshold permitting a single neuron to fire. Rows of the matrix correspond to stored patterns, and the synaptic coefficient matrix was defined as $\mathbf{M}=\mathbf{X}^T$. Since each output pattern is signalled by the firing of a single neuron in the output, the maximum number of stored patterns is equal to the dimensionality of the output vectors, K . This was the best the algorithms in terms of storage capacity and scored highly for robustness in noise. The disadvantages are that a winner-takes-all threshold is required over all of the neurons and in practical systems there is no fault tolerance so a single neuron failure causes the total loss of the stored pattern.

The ubiquitous sum of outer products algorithm, $\mathbf{M}=\mathbf{YX}^T$, was considered but with a reported storage capacity of only $N/4\ln N$ this proved to be the weakest of the algorithms in the investigation. This result is surprisingly low. Though it is not clear from the text, it may refer to the capacity of the recurrent Hopfield network (which does use outer product as its learning algorithm) and thus should have been considered apart from the other, purely feed-forward networks. The algorithm also performed worst in the noisy case.

The pseudo-inverse algorithm was shown by simulation to produce a higher storage capacity than the two previous approaches for $K=N$ (the DSNN algorithm can produce better results but only for $K>N$), being equal to the dimensionality of the input vectors, N . But the learning procedure itself, involving the minimisation of an energy function for the complete set of patterns, is both non-local in character and computationally intensive. Both properties make it unsuitable for the type of networks that will be studied later in this work. Also, for a system continually learning in a real-world environment it is not possible to know the entire vector set before any training can begin.

The next method they compare, the Ho-Kashyap (or HK) algorithm (and several variants), has a potential capacity of $2N$ patterns. This is an impressive result (compared with the Hopfield networks $0.135N$, for example). However, this algorithm is not only more compute intensive than the pseudo-inverse, it also modifies the output vectors (using an iterative procedure) to make them easier to store. Such an approach may be neither practical nor desirable in some problem domains. However, the notion of modifying the stored vectors themselves is one that was considered in the work reported in this thesis, both in the architecture (chapter five) and network development (chapter ten).

In terms of noise performance, Casasent and Telfer note that while the basic HK algorithm has poor noise performance when the number of stored patterns, M , is near the input dimensionality N , variants of this basic algorithm (although more compute intensive to execute) provide good performance in noise even for $M>1.4N$.

As a final method in this part of their work, Casasent and Telfer used more complex codes as the output of the first layer (rather than 1-from- K codes as they had done previously). These codes were intended as addresses into the second layer or an external memory containing the associated output patterns. This scheme, they referred to as Content Addressable Associative processors (CAAP)

and they restricted its use to the pseudo-inverse and HK algorithms, only. In both cases, the absolute storage capacity was not changed. However, the use of a more compact encoding for the output vectors permitted a reduction in the output dimensionality, K . This, in turn, led to an improvement in storage density due to the reduced total number of synapses. The results of their storage capacity analysis are shown in the tables below:

Table 3-0 Comparison of Network Storage Capacities in the noiseless case. From Casasent & Telfer (1991)

Memory Type	Storage Capacity	Memory Size	Storage Density
	M	NK	M/NK
DSNN	$M \rightarrow \text{infinity}$	NM	$1/N$
Correlation	$N/4 \ln N$	N^2	$1/4N \ln N$
Pseudoinverse	N	N^2	$1/N$
HK	$2N$	N^2	$2/N$
Pseudoinverse CAAP	N	$N \log_2 N$	$1/\log_2 N$
HKCAAP	$2N$	$N + N \log_2 N$	$2/(1 + \log_2 N)$

For networks operating in noise the results they obtained for the ratio of stored patterns to neurons for the case where at least 95% of the patterns are recalled perfectly is summarised below:

Table 3-1 Stored patterns per neuron, M/N , for several network types and several levels of noise during recall. From Casasent & Telfer (1991)

Noise, σ	Network Architecture			
	DSNN	Correlation	Pseudoinverse	HK
0.00	1, 2	0.12	1.04	1.52
0.05	1, 2	0.12	0.88	0.88
0.10	1, 2	0.12	0.72	0.72
0.20	1, 2	0.12	< 0.6	< 0.6

Overall, a trend is clear. The simplest training methods (such as the vector correlation or outer product algorithm) give the lowest storage performance and poorest performance in noise. Better storage requires more complex training methods, even to the extent of modifying the output patterns themselves. Generally, obtaining good noise performance involves training with noisy vectors, which adds to the training time.

The only exception to these observations is the nearest neighbour classifier (DSNN), which provides impressive storage capacity and noise performance with virtually no training (each input pattern is assigned to one output neuron, as described above). The problem of fault intolerance and the fact that the number of stored patterns cannot exceed the number of output neurons are the only drawbacks and even these may be acceptable penalties for some applications.

In a second part to their work, Casasent and Telfer consider alternative, denser encodings for the recollection vectors (the outputs to the first network layer). Four encodings are considered: binary (or standard) coding, Hamming codes, BCH codes (a variant of Hamming coding) and L-max encoding, in which only the L neurons with the highest potential fire, all others being silent. The properties of L-max encoding were first demonstrated by Austin & Stonham in their ADAM network, discussed earlier in this section.

Over a series of tests using a variant of the HK algorithm to train the weight matrix, it was shown that the L-max scheme offered almost the best performance in the noiseless case but proved to be superior (i.e. most robust) in the noisy case, with performance degrading slowly with increasing input noise. The binary encoding, while performing best in the noiseless regime, also showed the steepest performance degradation with increasing input noise.

Some of Casasent and Telfer's results are shown in the table below which shows the maximum ratio of stored patterns to neurons (i.e. the capacity per network processing element) for two different output encodings: the binary encoded output and the L-max encoding with $L = 2$. The capacity was measured for a range of error rates, P'_c , defined as the percentage of fully recalled vectors.

Table 3-2 Storage capacity, M/N, for different output encoding schemes and different recall accuracies, P'_c

	Binary Encoded Outputs				L-max (L=2) EncodedOutputs			
	Binary Biipolar Keys (number of errors)				Binary Encoded Keys (number of errors)			
P'_c	0	1	2	3	0	1	2	3
90%	1.52	0.80	0.60	< 0.60	1.68	1.36	1.04	0.92
95%	1.32	0.68	< 0.60	< 0.60	1.44	1.16	0.96	0.72
99%	0.92	0.60	< 0.6	< 0.6	1.24	1.04	0.72	< 0.6

The results show that the L-max scheme gives superior storage capacity for an equivalent number of errors and suffers a lower rate of performance degradation with decreasing tolerance to errors (i.e. increasing P'_c).

On a different topic, the point made by Nadal and Toulouse to the effect that the information stored per pattern is also a significant factor in determining information storage capacity was not considered by Casasent and Telfer. Perhaps they tacitly assumed that ignoring the second network layer (which encodes the output pattern) would render such considerations irrelevant.

A final point needs to be made concerning their definition of correct recall. They used the metric of 98% perfectly recalled patterns, so two in every hundred patterns were permitted to have errors. This, they argue, is because a single error in the output of the first layer causes the wrong memory to be addressed in the second. While this may be true, for many practical systems (including the one to be defined in this work) we require that *every single pattern* be recalled correctly. Allowing the odd pattern to fall outside of the defined error bounds is simply not

acceptable since it could lead to total confusion in an ongoing symbolic computation. Having said that, it is much easier to build-in a level of redundancy in each pattern so that each input vector can contain a non-zero (but bounded) number of errors and still be correctly transformed by the network. These issues are discussed more fully in chapter six, which discusses the pattern encoding methodology that will be used (see section 6.4.2, page 164).

3.1.3 Further Work on Correlation Memories

Here we consider some key points in some other contributions to the understanding and exploitation of associative memories. Much of the more recent work has been to examine variants of earlier networks or to apply them to particular domains.

Both a review and a different interpretation of Willshaw memories and their variants was provided by Palm (1980). He also speculated on their suitability as models for associative memory in humans and other animals, although he did not arrive at any definite conclusions.

Graham & Willshaw investigated more complex variants of the original Willshaw model by including partial connectivity between the input and output neurons in the calculations (Graham & Willshaw, 1995a, 1995b). Such an approach was designed to address more neurologically plausible models where fully interconnectivity between cortical cells does not occur.

The complexity that occurs in sparse, randomly connected networks is that the signal arriving at each output neuron is now subject to variation due to the missing connections. Graham and Willshaw address this problem by modifying the threshold as a function of the input activity and the output unit usage. They show that lower connectivity can lead to higher storage capacity and that the optimal information storage efficiency (stored bits per synaptic bit) is not only a function of the connectivity but also of the noise level in the input patterns, the required level of connectivity rising for increasing noise level. At a noise level of 40%, they show by simulation that the optimum connectivity is slightly below 20%, which they claim is a possible explanation of the low connectivity of the human neocortex. These results are extremely interesting and deserve further work at a later time.

As a rare example in work on the tailoring of an associative network for a given mapping problem, Turner & Austin (1997) provide a probabilistic framework for estimating the minimum size of a multi-layered associative memory for a given level of system performance. Interestingly, the network allowed both partial matches and multiple simultaneous valid outputs. To facilitate this, the first layer of the network used a threshold that was lower than the number of '1's in the input. Subsequent layers were used to refine and separate the patterns.

3.1.4 Kanerva's Sparse Distributed Memory

Although not a matrix associative memory, the novel approach to memory drawing on the power of high-dimensional vectors put forward by Kanerva is a related topic and so it will be included here (Kanerva, 1988).

Kanerva's memory architecture was built around a small number of high dimensional vectors. He used one million binary vectors each of 1000 bits, which thus could cover only a small fraction of the pattern space. The value encoded in each neuron, he called a hard location. The values of the all of the hard locations were distributed randomly so as to uniformly cover the 1000 dimensional input space.

The mechanism behind the memory is to allow any input vector, x , to be stored by first locating the set of hard locations that are within a Hamming distance r of it (defined as the number of bits to change to transform one pattern into the other).

The selection is done by applying the pattern vector in parallel to the one million neurons and selecting those with a potential above a value that is a function of r . (In the example, r was set so that the number of neurons had a mean of one thousand). Once the set of vectors has been located, several possible schemes can be applied to store the associated word. In the simplest, the target word is merely written to every hard location in the set, either overwriting the old value (if any) or being stored along with it, in the case of the multi-set (which allows an unbounded number of independent writes to the memory).

When attempting to read from the memory, the input vector is applied to all one million neurons, as before and the one thousand with the highest potential were selected. The value read was either the average of the values associated with the winning vector set or the most common value.

In an extension to the basic model, a series of reads was used to converge on the correct answer. The input vector was used as an initial guess and the output was used as the next guess. Under certain circumstances, the series of vectors generated in this way will converge on a better result than that obtained from a single memory read.

In fact, the mechanisms of the Kanerva memory have much in common with those of associative memory and of competitive learning. Many neurons in parallel compete to represent a given pattern and many patterns are superposed. The correct vector can be extracted from the noise generated in the overlap because its signal is statistically the strongest.

An extension to Kanerva's memory has recently been proposed (Hely *et al.*, 1999). The authors point out that the Kanerva memory is only optimal when the input vectors are evenly distributed and the statistics of the vectors do not change over time. Thus, for correlated vectors the memory is not optimal.

They propose a variation in which the storage locations are not specified in advance. Instead, they are created location by location as each new input pattern is presented, each new storage element taking the exact value of the input vector. Once all of the storage locations have been allocated, they begin to compete for new patterns. As in the earlier model, the Hamming distance is used to find the set of locations closest to the new pattern, but instead of the hard threshold of the Kan-

erva model (in which a particular location is either within the given radius of the input vector or not), a mechanism based around diminishing signal strength as a function of distance is used. Locations further from the input vector are modified less than those lying near to it.

To ensure that the storage locations follow the changing distribution of the input vectors, periodically there is a purge in which the locations which are used the least are pruned and new locations are initialised to replace them in areas of the input space that are more inhabited.

It is clear that this variation takes the model even further towards a competitive learning network. Locations effectively move into areas that are populated with data points, so the competitive element that existed before has become a fight for life for each storage element.

It is less obvious how such a memory compares with the standard competitive learning scheme such as a Kohonen network. For example, in the latter scheme the input vectors stored in the memory elements are moved in order to track the input statistics, whereas here this mechanism is performed using a kind of neuronal 'death and rebirth'. Whether this leads to better or worse performance is hard to tell and further analysis is required.

3.1.5 Summary of Associative Memory and Discussion

This section has presented some of the key contributions and results in the area of associative memories. It has been shown that using binary associative memory it is possible to achieve storage levels in excess of N , the number of neurons in the memory. When permitting errors to occur in the associated output pattern, the storage capacity can be increased still further.

Various storage prescriptions have been compared, leading to the not unsurprising result that non-local, computationally intensive algorithms using the entire pattern set lead to more optimal storage than the simple sum-of-outer products approach. The best method (in terms of both storage capacity and noise performance), called Ho-Kashyap encoding, requires that not only the weight vectors but the output patterns themselves be modified during network training. This process involves iterating the vectors in a noisy environment, increasing the computational load still further. For the type of network to be described in this work (one that must continue to learn new patterns) there is no opportunity to execute an algorithm based on the entire pattern set. This type of learning is clearly unsuitable.

It is interesting to examine the underlying goals of all the associative memories considered in this section. In each there are pattern pairs to be memorised and an ideal memory would allow any one to be recalled without interference from any other. The way that each network designer tries to achieve this goal is to encode each memory trace in a way that minimises the noise contributed by other memories. This mindset is at odds with that applied to multi-layered perceptrons where the hidden layer is trained specifically to locate features of commonality between the patterns and exploit them, thereby reducing the cost of storage. Once the under-

lying regularities of the data distribution has been extracted, an MLP is capable of generalising in the face of previously unseen patterns, giving an *effective* capacity far beyond anything demonstrated in any of the networks considered in this section.

In the work presented in this thesis a hybrid approach to memory will be taken, drawing on the most appealing aspects of each type of memory: the one-shot, reliable learning of the Willshaw model and the efficiency gains of the feature extracting MLP. This will be discussed in detail in chapter ten.

3.2 Recurrent Memories

This section will present and discuss the main contributions to our understanding of recurrent associative memory. Such a history tends to revolve around the Hopfield network, although non-Hopfield based recurrent networks do exist.

The basic structure of the Hopfield network was presented earlier (see section 2.3.6, “Hopfield Networks”, page 36). In this section, the analysis is presented in more depth, reviewing the main pieces of work which have appeared, covering such properties as storage capacity and issues in pattern recall.

Hopfield’s original paper provided empirical data on the storage capacity of N fully interconnected neurons, the output of each being hard thresholded, taking on values in the set $\{-1, +1\}$. As described in chapter two, Hopfield notes an empirical result for the maximum number of patterns stored before catastrophic failure of the network as $0.15N$ (Hopfield, 1982). It was several years before this result was shown theoretically to be nearer to $0.138N$, using a technique from statistical mechanics called mean field theory (Amit, 1989).

Before the analysis using mean field theory, McEliece *et al.* provided an interpretation of the Hopfield network as an information channel, ascertaining its capacity using information theory. In a well known paper, they showed that allowing a small number of patterns to contain errors permits a storage capacity, m (defined as the maximum number of patterns stored) of $m = n/(2\log n)$, whereas demanding that every pattern be recalled without error reduces the capacity to $m = n/(4\log n)$ in the limit of N tending to infinity (McEliece *et al.*, 1987).

The analysis using statistical mechanics performed by Amit and co-workers explained in terms of phase transitions why the performance of the network can undergo an abrupt transition from a working memory to total failure by merely adding an extra pattern or raising the statistical ‘temperature’ slightly. However, from the point of view of this work, we accept the value of $0.138N$ as an empirical result and will not look further into the statistical mechanics that lie behind it. A readable account of the mean field analysis of Hopfield networks is to be found in Hertz *et al.* (1991).

Vidyasagar analysed a class of dynamical systems called continuous-time, continuous-state (CTCS) networks, of which Hopfield networks are only a sub-

class (Vidyasagar, 1993). CTCT networks use a single layer of neurons that are fully connected, but each uses the sigmoid as the output threshold function, allowing continuous output values. The output state, H , can be modelled as a continuous position in n -dimensional space, where n is the number of neurons in the network. One important generalisation from the Hopfield model is that the interactions between neurons no longer need to be symmetric, so that in general, $W_{ij} \neq W_{ji}$.

The analysis of the model cannot be done using an energy function, since this can only be defined for symmetrical connections. Instead, Vidyasagar uses matrix theory to show that for small perturbations of the weights away from the symmetric case, the equilibria of the network are slightly perturbed, but not destabilised. Furthermore, the stable equilibria of such networks always exist in the corners of the output space, that is when the individual neurons have saturated at '0' or '1'.

These results are interesting since they lend credibility to the idea of building a non-symmetrical network (as will be done in chapter eight using dynamic patterns) and still produce a stable output in response to external stimulus. Though Vidyasagar's analysis has not yet been applied to the network defined in chapter eight, this would be an interesting topic for future research. (One adjustment to the dynamic patterns network would be the replacement of the hard threshold with the sigmoid or other differentiable, high-gain function, which would be required for the analysis to proceed).

The storage of hierarchical patterns using a Hopfield-like network, called a cascade associative memory (CASM), was recently investigated by Hirahara and co-workers, seeking for more efficient methods for storing correlated patterns (Hirahara et al., 1997). They generated a set of n -bit parent patterns by random selection of vectors from the set $\{+1, -1\}^n$. From each parent vector, a number of child vectors were created by randomly perturbing a number of bits of the parent. Two separate memories were used, ASM1 and ASM2. Parent patterns were trained into ASM1, while the differences between the parent and each child was stored in ASM2. The structure is shown overleaf:

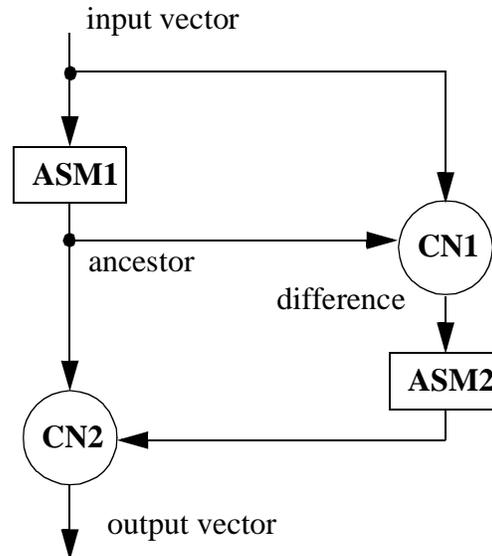


Fig. 3-1 Cascade Associative Memory (CASM), Hirahara et al. (1997)

Note that while the parent patterns use a biased encoding, with unequal chance of a bit being +1 or -1 characterised by a bias parameter, a , the child patterns stored in ASM2 had even greater bias, to the point of being sparsely encoded with signal bits set only for those few bits which differ between parent and child. This principle of increasing bias could be applied to a many more levels of the hierarchy, leading to whole chains of ASMs, each level acting as a delta the level above.

During recall, the input vector is first characterised by ASM1 to determine the ancestor pattern. This is done using an iterative update, as in the Hopfield network. When this has converged, the identified parent pattern is removed from the input using CN1, leaving only the difference which are then used as the cue for ASM2. This network iterates to converge on the pattern corresponding to the nearest set of differences for that parent. The selected parent and difference pattern are put back together by CN2 to give the nearest child pattern to the original input. In both CN1 and CN2 the operation performed is the same: bitwise multiplication.

The authors show that the storage capacity of their network (measured as the maximum number of stored patterns per neuron) is a function of the bias level of patterns in that network, with higher bias corresponding to increasing correlation in the vectors. They showed empirically that for bias $b=0.96$, a network with $N=1000$ neurons approached a maximum storage capacity $\alpha_c=0.6$, well above the Hopfield limit of 0.138 patterns per neuron.

However, they did not consider the storage capacity as a function of the total number of neurons in all layers of the system, something which might change the efficiency results considerably. Also, the fact that the number of layers and their composition are predefined in advance is a disadvantage for a system in which

there is no foreknowledge of the pattern set that will be applied. Furthermore, pattern sets in which some branches of the pattern tree are deeper than others would not be optimally stored in such a network. It would be much more desirable to have a system that could detect the structure in the patterns itself and store the whole hierarchy in a single network, with adjustments for different branch depths being accommodated at the synaptic level rather than at the level of separate networks as we see here.

Overall, the principle of hierarchical organisation of dependent patterns is an intriguing one, and one that will be pursued further in this thesis. In chapter ten the learning hierarchies principle will be described, which has similar aims to Hira-hara's, that of improving storage efficiency by drawing on correlations in the stored patterns. Where it differs is in how it tries to organise the hierarchical data, using only a single network but with more complex synapses.

3.2.1 Gardner-Medwin's Recurrent Memory Model

In a piece of work that for some reason, did not spawn the further work that it deserved, Gardner-Medwin looked at a variant of the original Willshaw memory from the point of view of a physiologically plausible account of how memory might function in real cortex as a means of storing sets of associated events (Gardner-Medwin, 1976).

His network consisted of a set of N two-state neural units, each of which could potentially receive excitation from R others ($R < N$). While bearing striking similarity to the Willshaw model, it differs in that the neurons are interconnected. The fact that, at the outset, each neuron was hardwired with a predefined subset of possible connection that it could make was intended to reflect the initial connectivity in mammalian neocortex; each pyramidal neuron in such regions makes connections with only a subset of the surrounding neural population. Only the strengths of the existing connections could vary as a result of learning.

An event to be memorised consisted of W active and $N-W$ inactive neurons. Whenever there was an existing 'potential connection' between two firing neurons, the connection was made between them (thus synapses were binary, once again following the Willshaw model).

During recall, a subset of a learned event consisting of, say, w_0 of the W firing neurons was applied to the network. The excitation from these activated input cells then spread to other (initially silent) cells via those synapses that had been activated during learning. If the potential of any neuron exceeded a threshold (the level of which will be discussed later) then that neuron would become active too, thereby recalling more of the memory given a fragment.

Gardner-Medwin defined two types of recall: simple and progressive. Simple recall allowed a single round of updating to take place, so that the final set of firing neurons consisted of the original input set of size w_0 plus those that were directly activated by them. In progressive recall, further rounds of updating were

allowed, so that neurons becoming active at the second and subsequent cycles of recall did so not only due to the original fragment, but also due to newly recalled parts of the event.

The most complex part of Gardner-Medwin's analysis, and the weakest part of the model, related to the setting of the threshold for the neurons. First, for optimal results the threshold needed to start with a low value and be increased in a non-linear manner as a function of the number of firing neurons at each time step.

Second, the recall performance of the network at each timestep was very sensitive to the accuracy of the threshold. If the threshold was set too low, the probability of a neuron firing spuriously increased dramatically, corrupting the recall. (The quoted example is that a 10% error in threshold can lead to a tenfold change in the expected number of spurious cells). Conversely, too high a threshold led to a collapse in the support for neurons that had previously started firing, causing them to stop firing.

If the threshold was set correctly at each timestep, Gardner-Medwin found that the network could store vectors of associated events with a bit efficiency (defined as the number of pattern bits that could be recalled per synaptic bit in the network) of about 7%, not as good as for the Hopfield network. As in the Willshaw memory (and for similar reasons) the highest levels of storage were obtained for sparsely encoded vectors, i.e. with only a small fraction of the N neurons firing for each event.

Overall, this network differs from both the Willshaw model and the network to be presented in this thesis in that, during recall, the number of firing neurons is not constant. Rather, it increases as the active set grows. It shares the property of recurrence with the network to be presented later, although the premise of Gardner-Medwin's approach is to start with a small set of 'correct' events and only admit new items to the set when they are statistically justified as belonging. This leads to the need to vary the threshold during recall.

An alternative (and arguably better) approach will be presented later in the thesis: that of allowing a fixed number of events to be asserted (rightly or wrongly) and letting network iterations revise the active set to find the most mutually satisfying interpretation of the event. This scheme will make it possible for any errors which creep into the active set to be removed by later iterations.

3.2.2 Bidirectional Associative Memory

Bi-directional associative memories are a hetero-associative variant of Hopfield network, first proposed by Kosko (1988). Rather than generate a symmetrical $N \times N$ matrix from auto-associative case with one input pattern, Kosko considered pattern pairs, \mathbf{A} and \mathbf{B} . Learning took the form of summing the outer products $\mathbf{A}\mathbf{B}^T$ leading to a weight matrix, $\mathbf{M} = \sum_i \mathbf{A}_i \mathbf{B}_i^T$, which is in general neither symmetrical nor square.

The single pool of N neurons in the Hopfield network is replaced by two pools of neurons, A and B. An input pattern is applied to the neurons of A and passes through the matrix in the forward direction. After thresholding, the resulting output pattern is represented by the neurons in B. For the second iteration, the pattern held on B is used as an input to the inverse matrix, \mathbf{M}^T . After thresholding this produces a new pattern on A, designated A'. This process can be repeated until no further changes occur in either A or B.

Kosko showed that iterating on any matrix \mathbf{M} in this way will result in stability in the limit as time $t \rightarrow \infty$ though the capacity, defined as the maximum number of pattern pairs that can be stored and recalled exactly, is less than the minimum of the dimensions of the storage matrix, \mathbf{M} .

Further work by Simpson extended the original result of Kosko by considering several different variants of the bidirectional network, including higher-order connections between neurons Simpson (1990).

For the second order case, there are two matrices defined between pools A and B. The first is the same as in the BAM network, where a matrix element corresponds to the correlation of a single neuron in pool A with one in pool B. The second matrix holds the higher-order correlations, so that a single element corresponds to the correlation of two neurons in one pool with one neuron in the other. Thus the signal term in each sum is the product of output level of three neurons.

Simpson shows that an energy function can be established for the higher-order networks and that the resulting network does have stable states corresponding to stored pattern pairs. By simulating various network configurations he shows that the storage capacity of the second-order network is roughly twice that of the equivalent first-order network (in terms of maximum number of patterns stored). However, as he points out, the number of synapses in the second order network is more than twice that of the first-order network, implying a reduction in storage density (the number of patterns stored per synapse).

Zhang and co-workers re-analysed the bidirectional associative memory from the perspective of matched filters (Zhang et al., 1993). By replacing the energy function to be minimised with an exponential function of the pattern vectors (a technique that comes from the theory of optimal receiver design) they claim to improve the storage capacity and stability characteristics of the original BAM. To do so they introduce a new parameter, γ , into the learning and recall equations that is a function of the total number of memories stored and is connected with the 'noise' in the communication channel between the input and output vectors.

Leung shows that a complex learning algorithm, called the Householder encoding algorithm, leads to better performance than the sum-of-outer products originally used by Kosko (Leung, 1993). The algorithm itself is based on finding the pseudo-inverse of the interconnection matrix, requiring full knowledge of the pattern set in advance and non-local computation of each synaptic weight. Though the results of comparative simulations by Leung show that this algorithm is much

better than the local correlations proposed by Kosko, for the purposes of this thesis it is rejected for further study for all of the reasons just mentioned: non-locality and computational complexity during learning.

In later work, Leung applies the perceptron learning rule to the training of the weight matrix (Leung, 1994). The matrix is constructed over multiple presentations of the pattern set, with adjustments made whenever a pattern has been wrongly classified by one of the output neurons. Like Rosenblatt's original perceptron learning rule it is local in character but not guaranteed to find a solution (Rosenblatt, 1958). Also, the fact that training the matrix requires an unspecified number of repeats of the whole training set makes it undesirable as a learning rule for the network to be developed in this thesis.

Finally, we consider work, again by Leung, into the theoretical capacity of second-order bidirectional associative memory (Leung et al., 1995). Simpson's earlier work did not include a theoretical analysis of the capacity of second (or higher) order BAM networks. Leung and co-workers show that the capacity (in terms of the number of stored patterns with zero tolerance to errors during recall) is

$O\left(\min\left(\frac{n^2}{\log n}, \frac{p^2}{\log p}\right)\right)$ where n is the dimensionality of the input vectors and p is the that of the output vectors.

Since the number of connections for a second order network is $n^2p + np^2 + np$ (where the first and second terms are the second-order weights between the n input and p output neurons) we see that for $n > p$, the storage density measured as the number of patterns per synapse is $O(p/(n^2 \log p))$. This shows that the storage efficiency is reduced for increasing n .

3.2.3 Other Work in Recurrent Networks

Several areas of recurrent networks, although interesting and important in their own right, will not be considered in depth since they are less relevant to the work described here. One such example is the Boltzmann machine variant of the Hopfield network. Here, the addition of noise in the output function of each neuron of a Hopfield network allows a form of simulated annealing to take place (Hinton & Sejnowski (1986). The noise offers the network the opportunity to escape from local minima with a probability that is a function of the statistical 'temperature'.

The approach has several drawbacks, however. The extra computation this implies for each iteration of the network makes it much slower to simulate than the equivalently sized Hopfield network. Even if this problem can be removed by implementing the system in fully parallel hardware, there are few known methods for the selection the annealing schedule (which is a description of how the statistical temperature changes over time) leading to a waste in computational time due to overcautiously cooling the system at too slow a rate. The system is expected to explore the space of possible output states, driven by a combination of energy minimisation and random injections of energy due to the thermal component. Even with an optimally chosen annealing schedule, such an approach is a computation-

ally expensive solution to even fairly simple optimisation problems, as the authors admit.

Kothari and co-workers developed a synchronous Hopfield network variant in which the weights are modified by adding the outer product of the current state to the weight matrix after each iteration (Kothari et al., 1998). They show by simulation that this improves capacity and noise performance though it is not clear why this should be so.

3.2.4 Summary of Recurrent Memories and Discussion

This section has presented many of the key contributions and results in the area of recurrent memories. The memory capacity of the Hopfield network has been derived using both information theory and mean field theory. The results are different for the two analyses, but agree that the maximum number of patterns that can be reliably stored and recalled is a fraction of the total number of neurons.

This is lower than the result from the feed-forward associative memories described earlier where a capacity equal to $2N$ could be achieved for certain learning algorithms. However, it should be noted that none of the work on Hopfield networks in this chapter drew upon learning algorithms of the complexity of the Ho-Kashyap algorithm that produced the $2N$ result. Perhaps there is room to improve even the Hopfield result using such algorithms.

An area that has not been studied in depth is the application of K-from-N coding (as used in purely feedforward associative memories) to Hopfield-like networks in which the neurons are fully connected and iterate asynchronously. The Harahara network used delta encoding and its neurons used +1 and -1 outputs, so the results may be the same if we used pure K-from-N coding and neuron outputs from the set of $\{+1, 0\}$. This topic will be addressed in chapters six and seven.

3.3 Coding and Representation of Symbol Structures

The debate between the AI community and the connectionist as to the appropriate framework for the development of intelligent systems was presented in the previous chapter (see section 2.4.1, page 38). One of the key assertions made by Fodor and Pylyshyn in their 1988 paper was that connectionist systems were simply incapable of representing the complex relationship between entities typified in the syntactic structures used in traditional AI, relationships that they felt were essential for a reasoning architecture.

A number of workers in the neural networks camp attempted to refute this allegation by addressing the issues of representing symbol structures in a neural substrate. This section presents a number of attempts to do so.

To provide some background for the issues involved in coding syntactic structures, the first part of this section discusses the way in which symbol structures are stored and manipulated in 'Classical' AI systems.

3.3.1 The Non-Neural Coding and Manipulation of Symbol Structures

The figure below shows a typical tree structure, taken from linguistics, representing a phrase that has already been processed to extract its constituent structure. In this case, the raw input phrase “John loves the girl eating the sandwich” has been processed using a set of re-write rules to identify the individual verb and noun phrases (VPs and NPs, respectively) and the relationships between them. Note that hierarchical decomposition is possible, as in the noun phrase “the girl eating the sandwich”, which acts as an object to the main verb ‘loves’ but also contains its own subject noun, verb and object noun.

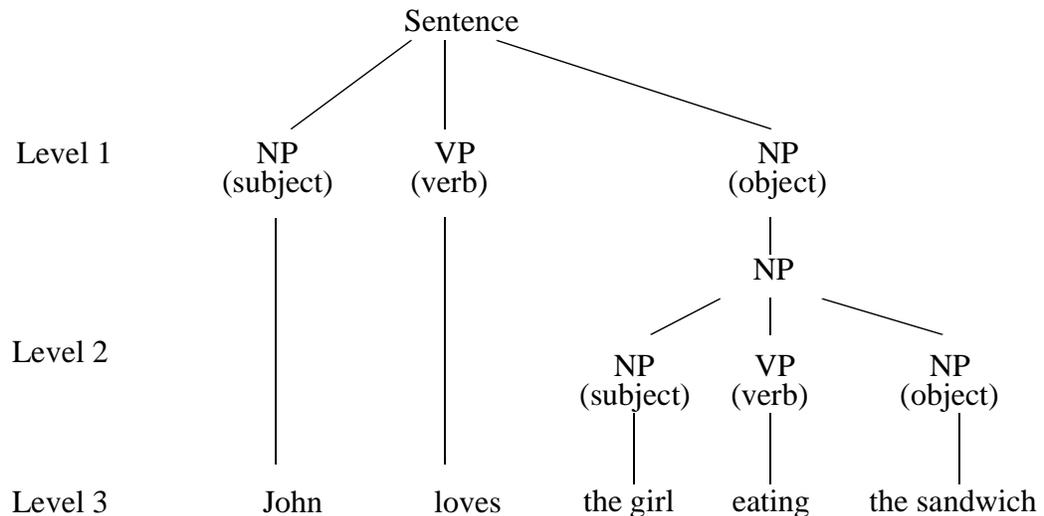


Fig. 3-2 A syntactic tree with branches of different depths.

In what is termed ‘Classical’ symbolic processing, the various manipulations that can be performed on such a tree consist of re-write rules such as:

NP VP NP --> NP

which indicates that the sequence made up of noun-phrase, verb-phrase and a second noun phrase can be replaced by a single noun-phrase. Processing thus consists of pattern matching between the data (as represented by a string of symbols) and the set of re-write rules (that are also usually represented as sets of strings).

In this case, the rule cannot be applied to the raw input (at level 3) since the rule is expressed as a relationship between types of word rather than the words themselves. Other rules are required to categorise the individual words as nouns, verbs, adjectives, etc. Simple, so-called context free rules could be written thus:

John --> NP, eating --> VP, sandwich --> NP

to provide the necessary categorisation. After this initial mapping the phrase (now represented at level 2) will match the re-write rule stated earlier and permit the representation of the entire phrase to be further reduced. Note that by replacing the sub-phrase “the girl eating the sandwich” with a single NP the information that

the sub-phrase contains is not lost, but is not directly accessible by the pattern matching algorithm. In this way, detail that is not relevant to the application of a particular rule is suppressed so that it does not interfere with that rule's operation. The benefit of this approach is that each rule can be written in the most general terms possible. It is not necessary to have an exhaustive list of rules to cover every case of parameters that do not contribute to the nature of the mapping. An example of this using the re-write rule given above will illustrate this point. The rule:

NP VP NP --> NP

states that for every phrase in which a verb phrase is flanked on either side by a noun phrase we can treat the whole as single noun phrase. We could spell this rule out for every possible verb, but this waste rule resources: the result will be the same for every single verb. Hence the abstraction from a given verb to the VP token allows compact, efficient representation and facilitates generalisation.

The drawback with such schemes, ironically enough, is also one of efficiency and is linked with the control issue. Correct transformation requires the inputs to be in the correct form, i.e. in the form envisaged when the rule was created. One of the major challenges, therefore, for the design of such schemes is to manage an efficient set of rules that can quickly lead to the transformation of a given input phrase into the required output phrase. In the general case there is no known algorithm that can optimally guide the transformation process (i.e. to select the minimum subset of transformations that will produce the required output). In many cases an exhaustive search over the rule-space produces many possible recordings of the phrase that can, themselves, be used as the input to further transformations.

It will be argued throughout this thesis that the inefficiency with Classically-represented symbol structures is amplified by the fact that the detail of the contents of a symbol are so completely hidden from the rules operating at a given level that this hinders the optimal transformation of the structure. In other words the totally arbitrary link between a symbol and its constituent structure make it necessary to expend more computational resource than is really necessary (although discussions on how one might measure this are left for future work).

The rest of this section looks at the issues in symbol structure encoding that are particular to a neural network implementation. It is envisaged that the use of neural networks as the medium of implementation will allow some of the restrictions inherent in the Classical implementation of symbol structures to be overcome. Attempting to justifying this view is a central thread of this thesis.

3.3.2 Three Key Issues in Representing Symbols in a Neural Network

While there are many problems to be overcome in the translation of symbol systems to a neural substrate, three key areas stand out as being most important. This section presents these issues and later sections describes some attempts to address them.

The first key issue is the encoding of the symbols themselves and the means by which they are connected into syntactically meaningful structures. Traditionally, a symbol can have any encoding (provided that it is unique). Its purpose is only to act as a pointer to some other structure and thus the encoding is unimportant. In a neural network this is no longer the case: representation is often the most important issue in the design of a new network. A good scheme can speed learning convergence and provide good generalisation capability. Conversely, a bad scheme could prevent the network from converging at all or make generalisation highly error-prone.

The second key area is that of variable binding. First-order logic permits the representation of generic relations such as:

$$\forall x, \text{dog}(x) \Rightarrow \neg \text{likes}(x, \text{ice-cream})$$

so that for any given value of x , if the predicate on the left is true, then the predicate on the right is also true. Variable binding allows a system to operate on individual values of x , using generic rules such as the one above, and possibly to hold multiple instances of such rules in memory simultaneously, each with a different value of x .

The problem in a neural network implementation is how to do this binding. If dedicated 'hardware' is required for each rule, how can multiple copies of a given rule exist each with a different value of x ?

The third key issue is that of stability of symbols and their relationships. A symbol can represent something highly significant to the agent, perhaps because it represents something which occurs frequently in a variety of contexts, or which has a significant impact on the agent's ability to solve a problem when it does occur.

In either case, a symbol may be a permanent feature of the knowledge base, whereas the relationships in which it appears may be varied and each may occur with much lower frequency. Entire symbol structures, composed of permanent symbols, may be quickly created, used and then discarded in a short space of time.

The problem with this way of working is that most neural architectures are not good at differentiating between more or less permanent information. Usually fairly crude mechanisms are employed such as dividing the memory at a high level into separate long and short-term blocks, perhaps derived from the simplified models of mind used in psychology (Gross, 1996).

When networks that one might describe as *monolithic* are used (for example a single MLP network) learning is usually achieved at a single rate that does not distinguish between temporary and permanent links between data items. Any new learning risks to disrupt existing associations. Clearly, more complex learning schemes are needed to address this issue.

The rest of this section presents work in the literature designed to address these and other complexities which arise when trying to map complex symbol structure onto a neural network substrate.

3.3.3 Mapping Whole-Part Hierarchies in Connectionist Networks

With the critique so eloquently provided by Fodor and Pylyshyn (see section 2.4, “Artificial Intelligence & Neural Networks”, page 38), it was up to the connectionists to refute the claims that, among other things, neural networks were incapable of representing symbolic structures in a way that is both systematic and productive. The first such attempt at a rebuttal to be considered here is that of Hinton (1990).

The work was an extension of his earlier development of coarse coding (Hinton, 1986) and was made up of two parts: a general exposition on the key issues and three example implementations with increasing levels of sophistication.

In the general exposition, he makes a number of important points. For example, he emphasises the need for the symbol encoding to carry useful information as well as pointer data. A symbol, in his view, is ‘a “reduced description” of an object’. This is a key point that will be developed further in later chapters (see section 5.4, “The Symbolic Principle”, page 123).

Later, he differentiates between the mechanisms of rational and intuitive inference (the behaviours of which were debated in Smolensky (1988) and Fodor & Pylyshyn (1988)). An intuitive inference is arrived at by applying constraints to the network and allowing the state to evolve until it has ‘settled down’. The nature of this settling process (while not stated) is implicitly that described in his work on Boltzmann machines (1986).

More complex computation, which might be unachievable in a single settling might require multiple steps, arranged serially with the results of each feeding into the next. This sequence, he refers to as *rational inference*. Thus, to Hinton, even the logical progression of logical thought is, in fact, an assembly of intuitive steps. There is no physical separation between the processors for each type of operation. In this respect, he is agreeing with Smolensky.

Each intuitive inference in a sequence might itself be relatively complex. Hinton suggests that the completion of a ‘schema’ (a framework in which one can assemble several relationships involving a number of roles and their fillers) would be a plausible candidate for a single intuitive step.

As a separate topic, Hinton discusses symbol encoding and the importance of allowing the network to develop its own encodings to suit the mappings being made. This is an important theme in this work, and one that will be explored in greater depth in chapter five. The figure below shows one scheme he considered. The network below is used to associate two symbols via a relation. The boxes at the bottom represent the symbols to be bound, which are coded using 1-from-N (localist) approach. The middle three boxes, one per role, have an encoding which

develops during the course of learning (via back-propagation), Thus, the encodings can be shaped to meet the requirements of the representation rather than being artificially imposed from the exterior.

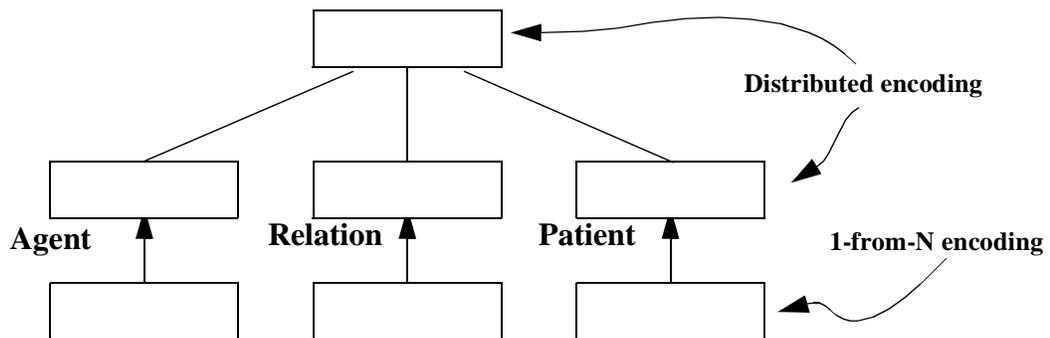


Fig. 3-3 Defining relationships between two objects. From Hinton (1990).

Three connectionist models for encoding and resource sharing were considered, which Hinton called *fixed mapping*, *within-level time-sharing* and *between-level time-sharing*.

The *fixed mapping* technique, typified by the McClelland Rumelhart word recognition model (in McClelland & Rumelhart, 1981) uses a localist representation and allocates one neuron for each role-value pair. In this case, each combination of letter and position within the word is represented by the firing of a single neuron. As Hinton notes, the neural circuitry needed to recognise any letter in one position must be duplicated for all of the positions. While appropriate for problems in which the range both of roles and of possible fillings is limited, such a scheme is too inefficient and restricted for a general purpose system.

Next the *within-level time-share* technique, which attempts to reduce the degree of parallel resources by transferring the letter by letter analysis from the spatial domain (with repeated circuitry) to the time domain. Now each letter requires one set of detectors and it is scanned along the words, letter by letter. While more efficient in terms of resources required, it places extra burden on the support structure of the system (required to move the input sequence along one letter at a time and to store intermediate results).

Finally, the *between-level time-share* technique, which assumes that regularity exists all many levels in the hierarchy of a symbol structure, permitting the system to sequentially process such a structure one node at a time, as shown overleaf. With each processing step, a different node becomes the “whole” and the structure directly beneath can be mapped into the finite processing resources of the network.

In the diagram, we see that either node A or node D in the symbol hierarchy on the left can act as the “whole”, with the constituent structure of each being mapped to the finite resources shown on the right. When looking at node, the dotted mapping are used, whilst for node D it is the solid mapping that apply.

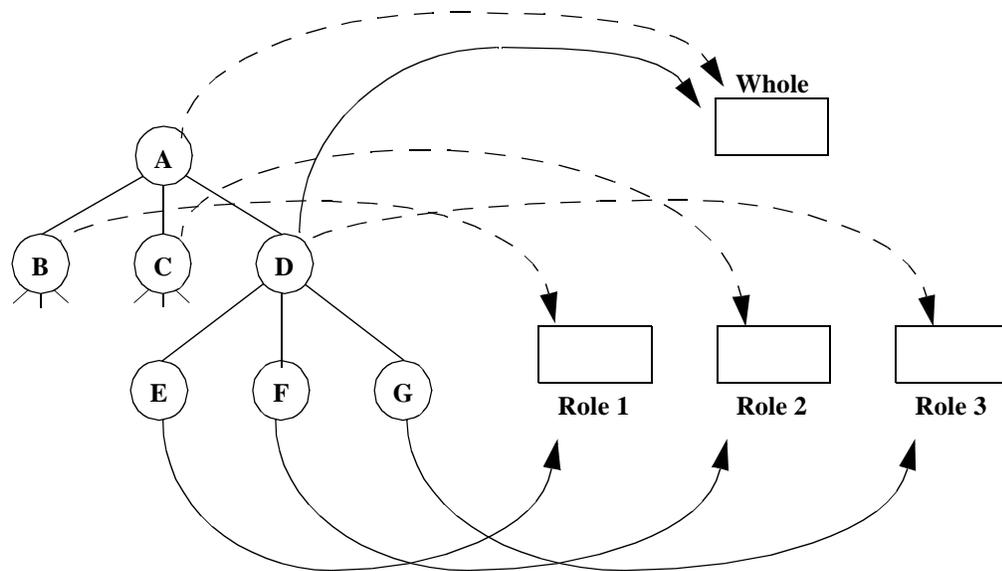


Fig. 3- 4 Between-level time-sharing. From Hinton (1990).

This third scheme is by far the most general, and could easily be applied to the parsing of any symbol structure. However, the fact that very little in the way of the problem space has been represented explicitly as parallel hardware means that the time element is now increasingly dominant. The network can handle structures of arbitrary complexity but must do so serially. Some external guidance must be applied to decide how the hierarchy is to be traversed (which is in stark contrast to be deterministic or linear styles of the *fixed mapping* and *within-level time-share* methods, respectively).

As a final insight, Hinton briefly discusses the use of short and long-term weights, with the short term providing a means for recent learning to override the long term when appropriate. His usage of such weights is to allow a return-from-procedure-call to occur in a procedural network. Later in this thesis an independently developed technique called *learning hierarchies* will be presented which also uses parallel weights with differing characteristics, but with a different purpose, that of addressing the stability-plasticity dilemma.

3.3.4 Tensor Products

Smolensky himself proposed a representation scheme based on tensor products (Dolan & Smolensky, 1989; Smolensky, 1991). Here, a form of variable binding is proposed whereby an entire tree structure made up of roles (such as subject, verb, object, etc.) and their values (“John”, “loves”, etc.) can be coded together into a single tensor. Each role is assigned an n-bit vector which is independent of any of the fillers (the value assigned to a role in any given expression). Similarly, each filler is assigned an m-bit vector which is independent of any role.

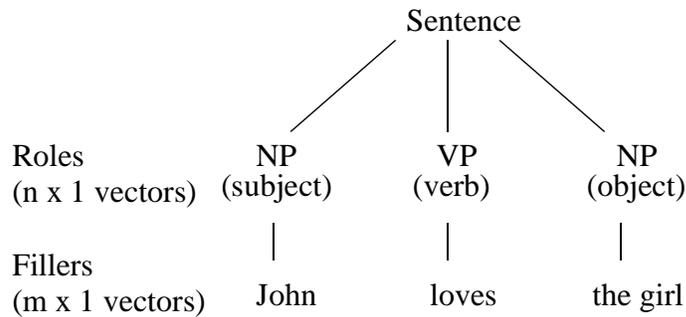


Fig. 3-5 Tree structure with roles and fillers.

To create a particular tree, each role-value pair is first combined using the tensor product operation and then the pairs are combined by vector addition.

Thus for the example linguistic tree shown above, we can construct the tensor T as follows:

$$T = \sum_i R_i \otimes f_i$$

where R_i is an n -bit Role vector, f_i is an m -bit filler vector and the \otimes operation is the vector outer product. T is thus an $n \times m$ matrix. The summation is over the number of components (three in the example). The role vectors are linearly independent of each other which permits any value to be extracted from the composite vector using the role vector as a 'key'.

Finally, it is worthy of note that recursion is possible, allowing tensors to be embedded inside each other, creating a matrix of increasing dimension with each added component. A tensor defined thus:

$$U = \sum_i S_i \otimes R_i \otimes f_i$$

would produce a three dimensional matrix, U .

Smolensky's expectation was that structures of knowledge encoded in single vectors could be fed in parallel to a neural network permitting true parallel processing of a structured representation. He notes that, while defenders of the purely symbolic view claim to have nothing against parallel processing *per se*, there was no evidence of the existence of a parallel processing *symbolic* architecture that could act as a benchmark for his tensor product scheme. While this might be true, it hardly constitutes an argument to justify the validity of his proposal.

There are two principle advantages to the tensor product approach. First is that it permits symbol structures to be represented as distributed representations

across a set of identical neurons. Second, it incorporates variable binding and thereby provides a solution to the systematicity problem put forward by Fodor & Pylyshyn. This, even on its own, is a feat of some merit.

But the method is not a rebuttal of all of Fodor & Pylyshyn's objections. Three problems with it are worthy of discussion. The first concerns the philosophy behind the encoding scheme itself. As long as the elementary vectors that are to be combined have the same dimensionality (e.g. all roles are n -bit vectors and all fillers m -bit vectors) then it is easy to see how the combination could be executed to produce a single matrix. Addition is well defined for vectors or matrices which have the same dimensionality ($n \times m$ in this case). But in a general symbol structure, we could ask to combine structures of different depths such as might be required for the tree shown in Figure 3-2 on page 68.

By creating the NP "the girl eating the sandwich" we have already formed a tensor of dimensionality $n \times m$ and must now combine it with other vectors with a different dimensionality.

The NP (object) "the girl eating the sandwich" role-filler pair would be a tensor of dimensions $n \times n \times m$. To match this, the NP(subject) "John" and VP(verb) "loves" role-filler pairs, each of size $n \times m$, would need to be combined with a dummy role, an n -bit vector. In effect, every sub-expression must be made to be the same dimensionality before they can be summed.

Using dummy roles is straightforward to do, but it adds one level of indirection to the affected sub-expressions. Presumably, the processes that must act on each such expression will be sensitive to (and compensate for) the extra level of indirection, but the dummy role is still somewhat of a fudge.

The second issue with this scheme is the variable length of the vectors. Handling vectors of different lengths is a task which few, if any, neural networks have excelled at. The reason is clear enough; the weights built up between one layer of neurons and another (be they input-to-hidden, hidden-to-output, or whatever) are sensitive to the position in which the bit patterns appear. Translating a pattern presented to a basic MLP, for example, is unlikely to result in the same output except in cases where translational invariance was specifically added. Similarly, weights built up with data vectors of one length are by no means guaranteed to produce similar results with the same data encoded in vectors of a different length. Noting that the length of a vector increases by a factor of n with each level of hierarchy, it is not clear how any practical system could be designed to cope with this variability.

The third issue is more fundamental, being concerned with the goal that we are trying to achieve. In Smolensky's scheme, all of the information which is present at the bottom of the hierarchy is carried through all of the layers of coding. In effect, the coding of an expression does not *refer* to its constituents, it *codes for them* explicitly. While this may be acceptable for a small set of constituents which can be coded in a few bits, extrapolating to the case of thousands or even millions of potential constituents forces one to reconsider the philosophy behind it. It is

partly due to the reluctance to shed any information of the lower levels which explains the explosion in the matrix size as a function of hierarchy depth.

Surely, what should be encoded in a symbol are *sufficient properties* of the level underneath it to permit operations performed upon it to execute correctly. In general there will be much detail hidden in the levels below any one symbol that is irrelevant for a given process and there is no reason to burden the process with this data. Thus the tensor product scheme, while representing the first serious attempt to model symbol structures in a neural network, is clearly only a partial solution.

3.3.5 Recursive Distributed Representations (RDR)

Following on from the work of Smolensky, other researchers developed the idea of encoding complex symbolic structures in a connectionist framework. First, we consider four such attempts, each building on the one before, which were particularly concerned with the idea of transforming strings of arbitrary length into connectionist representations of fixed length.

The first to be considered is due to Pollack (1990). His approach is illustrated in the figure overleaf. His aim was to train an MLP network that was capable of taking pairs of symbols, each of which was encoded as a K-bit vector and compressing them to a single K-bit vector. He did this by presenting the pair both as the input and desired output pattern, allowing the hidden unit layer to develop it's own representation over K-bits..

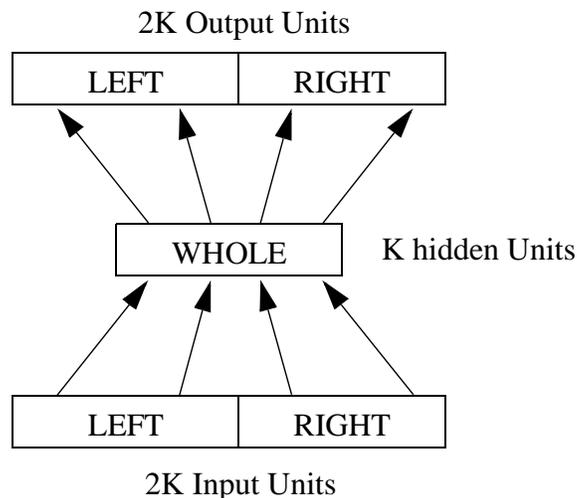


Fig. 3- 6 Symbolic Compression Scheme. From Pollack (1990).

To handle strings of more than two symbols, he allowed the result of one such encoding to act as one of the input vectors for a further association. This allowed a stack to be developed, such as could be applied to structures such as the LISP expression $((\text{Nil } X) Y) Z$, shown overleaf.

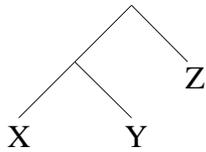


Fig. 3-7 A simple embedded tree.

In simulation, Pollack used ordinary back-propagation to train the network. As a consequence, many training cycles were required and there was no guarantee that the network would converge to a correct mapping. Also, he made the decision that rather than fully training one mapping before using its compressed form in the next, all pairs of associated vectors should be trained in parallel. Thus, the values of the hidden units, themselves evolving over time, acted as a ‘moving target’ for other learning.

Pollack found that the network, after hundreds of training epochs, was capable of correctly reproducing the given input expressions at the output, passing through the compressed intermediate format.

Furthermore, he was also able to demonstrate the generality of the internal encoding: by applying expressions that he had not presented during training he found a high proportion of them were correctly reproduced at the output. This implied that the hidden units were able to extract elements of the grammar and apply them to previously unseen sentences. This is an effective demonstration of systematicity and goes some way to refuting Fodor’s claim that connectionist networks were incapable of such behaviour.

As an extension, Pollack used his network to perform inference by associating an input expression with an implied output expression such as (from Pollack, 1990):

“If (LOVED X Y) then (LOVED Y X)”

The (LOVED X Y) acts as the input string and the (LOVED Y X) is the output that must be derived from the internal representation. He was able to extract the appropriate mapping even from input strings not previously seen. This is to be compared with the manipulation of symbol structures in Classical AI (section 3.3.1, page 68) in which a set of re-write rules is applied to an input string to transform it.

Overall, Pollack showed that compression of strings of arbitrary length was possible using connectionist methods. Moreover, the limitations of the model are also useful in themselves as launching points for future expansion. We now consider these drawbacks.

Firstly, the use of backpropagation as the learning algorithm is a problem. The need to train the network over many epochs, coupled with the lack of any guarantee of convergence make this an inappropriate model for a practical system.

Secondly, the ability of the system to generalise is not dependable. Pollack found instances where an attempt to code and then decode an expression not previously seen resulted in an output which was identical to a previously learned expression. He noted that this was caused by the coding of two noun tokens using bit patterns which differed by only a single bit. Thus, the choice of encoding of his original tokens was clearly critical.

He further notes that the systematic assignment of bit-patterns to tokens (such as similar patterns for all nouns) may have helped in the ability of the network to generalise relative to a network with random pattern assignment.

Thirdly, despite the ability to generalise in an (almost) systematic manner, the usefulness of the internal representations must be called into question. It seems that a compressed expression serves only as an efficient form of storage. No manipulation of the expression is possible in this form. The property of systematicity merely allows the compression abilities of the network to be applied to a wider range of expressions. It does not render them applicable to a greater range of manipulations.

Finally, note that access to any one symbol in a tree requires the sequential unfolding of the expression until the symbol in question is reached. Again, this is a consequence of the linear method of putting together an internal representation of each symbol pair.

An improved representation would not only permit systematicity but would permit manipulations other than merely compression and decompression.

How does this scheme compare to Smolensky's tensor product scheme? The obvious difference is that the compressed format in RDR is always a single vector, whereas the dimensionality of the tensor matrix increases with the number of combined elements. But Pollack's compressed format does not allow individual elements to be extracted based on a cue (such as a role vector), which is the essence of the tensor scheme. Instead, the whole expression in RDR must be unpacked until the desired element is revealed. So criticism of tensor products for producing larger representations would be unfair.

Another difference is the way that elements are combined. The matrix addition of tensor products can be executed in a predictable period of time. The results are guaranteed and the impact that each new learning event will have on existing knowledge is known, statistically. In contrast, the back-propagation learning of RDR (as noted above) requires many iterations. The exact number of iterations may not be predicted in advance, nor the degradation that each new learning event will imply. This lack of control is a major disadvantage of RDR.

We turn now to the second development on distributed representations to be discussed. Presented by Chalmers in his 1990 paper, it builds on Pollack's RDR scheme, concentrating on the possibility of applying transforms to the compressed format of the vector: in other words allowing computation on the symbol structure without unpacking it. He notes that being able to do this would provide functionality which did not exist for purely classical symbolic structures and would prove that connectionism was no longer merely concerned with implementation issues of symbolic architectures.

He quotes Van Gelder (1990) who made the distinction between *functional compositionality* (in which an expression is a potentially complex function of its constituents) and *concatenative compositionality* (in which constituents are merely arranged in a particular order but themselves remain unchanged). Only neural networks were suited to implementing the former, in his view.

The application that Chalmers chose was the association of an active sentence with its passive equivalent. To begin with, he used the same compression scheme as Pollack (except that he combined three elements at a time rather than Pollack's two). Two such networks were trained: one for auto-associating the active form of each sentence with itself, the other for the passive forms. The key idea he proposed was to create a third network (using one layer of hidden nodes) which could translate the vector from the compressed layer of the active network into the corresponding vector for the passive network. This would implement a transformation between the two compressed sentences without reference to their constituent structures. This is illustrated overleaf.

After 1500 training epochs using back-propagation the network was able to correctly transform all of the training inputs correctly and also to transform more than half of a set of previously unseen sentences. This indicates that, as observed by Pollack earlier, the internal representation was such that generalisations of the syntactic structure of both active and passive sentences had been made, but the rules themselves were not perfect as indicated by the performance on previously unseen test data. Thus, the network could not be described as performing true variable binding, for example.

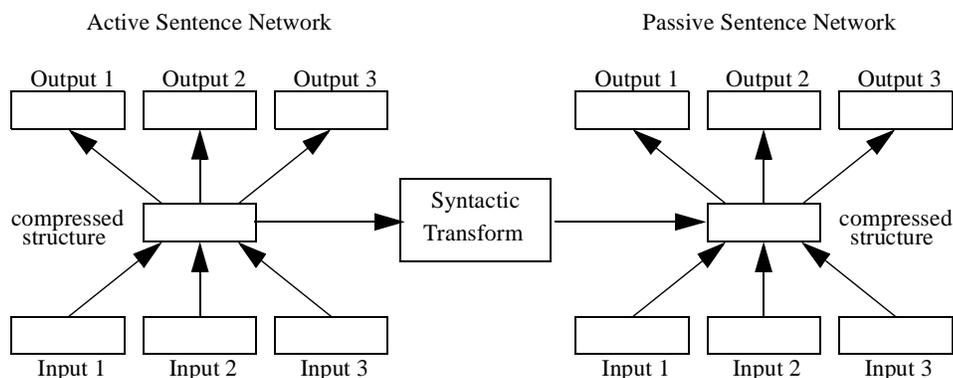


Fig. 3- 8 Chalmers' syntactic transform model.

While an important demonstration of the potential of neural networks to handle processing in a distributed fashion, the network is not practical when scaled up for real world applications. First, the training regime is artificial: the individual RDR networks are first trained (for several thousand epochs) and then the transform network is trained afterwards. Ignoring the inherent problems in using back-propagation for a moment, we see that the choice of micro-feature for the individual compressed vectors have no opportunity to take those of the other network into account. Thus the burden is on the central network that performs the syntactic transform to associate arbitrary vectors together. This may (or may not) be feasible for a given problem but is clearly not a good engineering choice since it is not predictable in advance.

Furthermore, there are problems in scaling up the scale of the task. We require one such transformation network for each pair of associated vector types. Each requires many training epochs and there does not appear to be a way to combine transforms (perhaps to extract a hierarchy of transformational steps which could be draw upon by many such transforms). However, nothing precludes such a possibility as future work.

The third approach on recursive distributed representations to be considered is due to Chrisman (1991) who built on Chalmers' *transformation inference* model, defining an alternative called *confluent inference*. This is illustrated overleaf. The essential difference is that rather than performing the transformation between them as a separate step, this network is trained to form a single compressed vector which is capable of representing both transforms simultaneously. This avoids the need for an explicit transformation network and allows translation in both directions.

Chrisman used this network to learn english phrases in encoder/decoder 1, and the spanish translation in encoder/decoder 2. After 5000 training epochs, the network could handle previously unseen phrases. 80% accuracy was achieved for auto-association(e.g. english-in, english-out) and 75% for a translation from english to spanish (or vice versa).

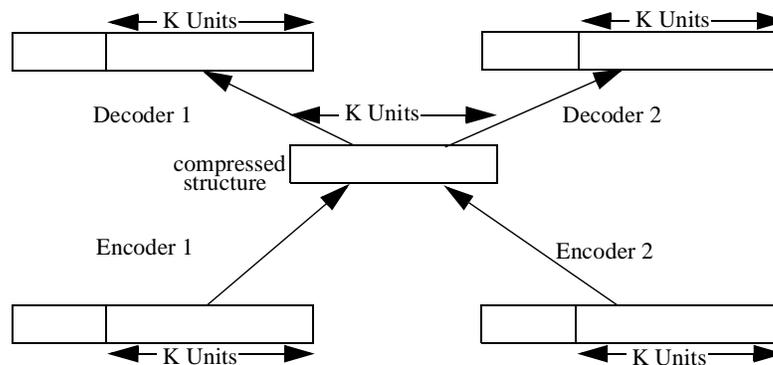


Fig. 3- 9 Chrisman's confluent transformation model.

These results show that the micro-feature generated across the K hidden nodes were extracting general properties from the two languages simultaneously. However, the natures of the errors showed that the performance was a sensitive function of the chosen encodings. Extending this scheme to a more realistic environment in which the input encodings cannot be so easily controlled would probably lead to much poorer performance.

In an extension to the basic idea, Chrisman suggests that this network could be extended to encodings of M arbitrary functions of an input vector x using M+1 encoder/decoder pairs (one to auto-associate the input vector x and one for each of the M functions). In principle this is true, but the danger here is that the more mappings that we try to squeeze through the same set of hidden units, the more complex the mapping task and the longer it will take to train. The generalisation performance would presumably be adversely affected, as well. None of these issues are addressed by Chrisman, but deserve further attention.

The fourth and final development of RDR to be discussed is the B-RAAM model, created by Adamson and Damper (1999). Here, the Chalmers' model of Recursive Distributed Representation is elaborated to try to correct for a number of the problems they identify in their analysis, specifically: the number of training epochs required, the physical number of hidden layer units required for a given encoding, the bias of the network to recalling recently learned patterns rather than older patterns and the intolerance to noise due to accumulated error during recall.

Their extension to the model is to use a history of recent hidden layer outputs as part of the vector to be auto-associated instead of just the previous value. A delay line provides the mechanism by which this is achieved. During training, the entire history of the growing tree can be explicitly represented in the input/output vector pair if the maximum tree depth is no greater than the number of state vectors that comprise the delay line. This allows the network to consider both recent and distant parts of the encoding vector equally during learning.

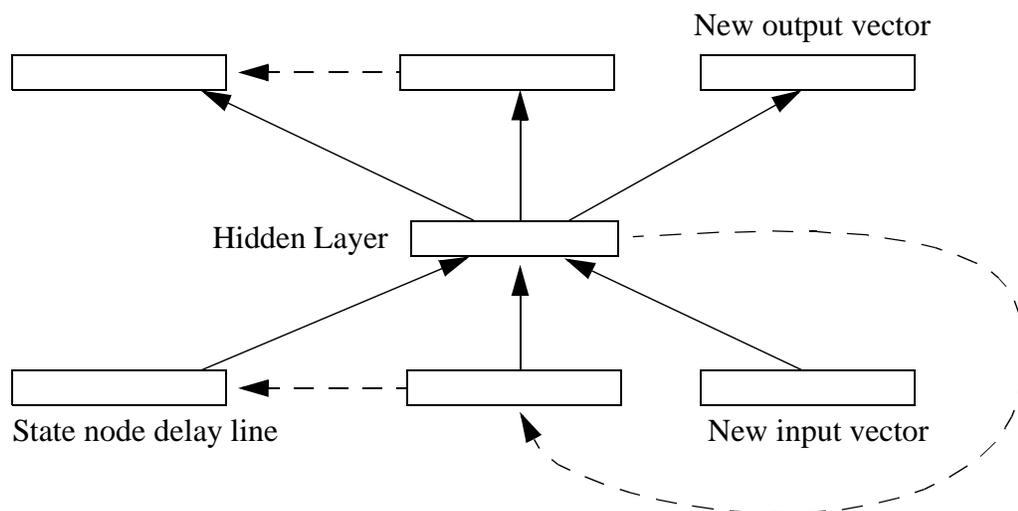


Fig. 3-10 Adamson and Damper's extension of RDR.

In the course of a number of experiments to compare their B-RAAM to Chalmers' original RAAM, they showed empirically that their network required fewer training epochs to learn the same task and that fewer hidden nodes were required. Furthermore, the bias towards recent learning was greatly reduced, as one might expect from the parallel representation of the compressed vector history.

Overall, this network represents an improvement over earlier RDR approaches but in the context of this work it is merely an interesting variation on the idea of compressed structures, which is itself the most important concept to note.

3.3.6 The Circular Transform in Symbol Structure Encoding

An alternative method of creating distributed representations of symbol structures was proposed by Plate who used circular transforms to combine role and filler information (Plate, 1996, 1997). For the case of vectors of length 3, he defines the circular convolution, t , of two vectors c and x thus:

$$\begin{aligned} t_0 &= c_0x_0 + c_2x_1 + c_1x_2 \\ t_1 &= c_1x_0 + c_0x_1 + c_2x_2 \\ t_2 &= c_2x_0 + c_1x_1 + c_0x_2 \end{aligned}$$

where the elements of each vector are drawn independently from a distribution of mean zero and variance $1/n$ where n is the dimension of the vector. This ensures that the mean vector length is one. Note that the transform represents a change of bases or, equivalently, a rotation. The vector t now codes for both the role (c) and the filler value (x) even though the size of the all three vectors is n . Thus, some form of compression has been achieved.

The inverse transform is achieved by multiplying by the transpose of the matrix, which represents a rotation through the same angle but in the opposite direction. In this way, given the circular convolution of two vectors, either one of the two vectors can be used to retrieve the other.

The next step of Plate's development is to generate more complex structures by combining several such vectors in what he calls a memory *trace*. Several possible schemes are considered, in which the central method of combination is simple vector addition. In order to combine and subsequently extract three vectors, a , b and c , a number of possible schemes are proposed. One possibility is:

$$d = \alpha_1 a + \alpha_2 a \otimes b + \alpha_3 a \otimes b \otimes c$$

where " \otimes " denotes the circular convolution operation. The constant, α_i , permits the relative strength of each component of the combined vector to be controlled independently of the others.

By making $\alpha_1 > \alpha_2 > \alpha_3$ we ensure that the base vector a is strongest in the mix and can be easily extracted. This in turn, can be used to extract b by applying the

inverse transform to d using a as the other input. Similarly, c can be extracted from d using $a \otimes b$ as the other input to the inverse transform. In each case, there will be some noise associated with the result, but a clean-up operation can be applied to remove it, subject to the non-saturation of the vector, d .

Embedded structures can be created by using a vector such as d as one input to a further circular convolution.

Overall, the circular transform offers a way to encode structures of variable depth using a simple matrix transform. Encoded values can easily be extracted using the associated vector as a key, similar to a correlation associative memory (see section 3.1, page 47). However, the method is not without problems.

First, note that separate traces cannot be overlapped. Each trace is the sum of several convoluted pairs. By combining two traces, information of which component is a part of which trace is lost. (In fact this is not a problem provided that no one vector appears in both traces but imposing such a restriction would be difficult).

The fact that traces cannot be easily superposed means that, in an implementation of such a system, a given input would need to be matched to each trace in turn (or each trace must be given its own hardware). This goes against the aim of associative memory which tries to overlap the storage of multiple memories in order to perform parallel search in a single set of units. (Note that this was also true of Smolensky's tensor product scheme where role/filler tensors were also summed for each such memory trace. Once again, summation of multiple traces would lead to ambiguity over which role/value pairs were associated with which others).

The next point of note concerns the clean-up process that Plate defines for the output generated by the inverse transform, when a single vector is extracted from the encoded vector. Here, the noisy output vector must be matched against the entire set of legal vectors. The closest matching legal vector is used in further operations.

This is not an issue for a limited vector set, especially in a system in which the traces are being compared serially (which seems to be the case in Plate's model). But in a real system this is a source of two types of limitation. Firstly it is a bottle neck on the speed of computation since each result must be put through a serialising step that does not lend itself to distributed processing.

Secondly, this is a restriction on the valid results that the system can produce: the only valid output from any operation on an encoded representation is one of the set of legal vectors defined before processing began. Any structural output (i.e. containing multiple elements or convolutions) would be cleaned-up to produce only one of the original set. Somehow, this restriction must be removed if this encoding methodology is to be put to use in more complex problem domains in which intermediate results are themselves still in encoded form.

As a final note on this type of representation, it is worth noting that the compression process, which on the surface takes several vectors of a certain length and produces a single vector of the same length, does not do so for free. The compressed vector may consist of fewer physical bits than the uncompressed vectors, but the noise margins of the stored data have been reduced. It is the continual erosion of these margins with each level of compression which results in saturation of the vector at a critical point, and the concomitant loss of retrieval fidelity. These noise margins are a resource that must be carefully considered as a part of the design process.

3.4 Conclusions

This chapter has presented two main areas of the literature in some depth: associative memories and the encoding of symbol structures using neural networks.

The review of associative memories divided the field into purely feedforward and recurrent memories. Key results on the capacity and performance of a range of networks was presented and discussed. It was established that the vector coding is a critical factor in determining the performance of the memory. In terms of the maximum number of patterns that can be stored and recalled, recurrent networks such as the Hopfield network appear to be much poorer than purely feedforward network such as the Willshaw model.

In the review of methods to encode complex symbol structures, starting with early work by Hinton, various attempts at compressing structures of arbitrary complexity using back-propagation as well as an alternative approach based on circular convolution were presented and contrasted.

One conclusion was that back-propagation, although powerful, is not the right algorithm for such work. A better algorithm must be found that is tailored for making reliable associations between symbol encodings. Also, it was argued that current techniques for compressing vectors ignore the fact that only part of the whole structure is relevant to the current process and this should be represented more prominently in the vector.

Finally, the importance of allowing multiple memory traces to be stored in the same set of neurons was emphasised. Without such capabilities, it was argued, the efficiency of the storage would be too low to be applicable to large knowledge bases.

The next chapter continues the in-depth review with details of existing reasoning architectures.

Detailed Review II: Reasoning Systems

4.0 Introduction

In this second part of the in-depth review, the highest level of the system will be considered: that of the information processing architecture itself. Such systems are characterised by the manipulation of symbol structures and rules, as discussed in chapter two.

For the purposes of this discussion, reasoning architectures will be divided into five types: *logical inference engines* using first-order logic and deductive inference to derive new facts; *production systems*, a form of symbolic state machine; *semantic networks*, which are a type of knowledge base which permit properties of classes to be inherited by their members; *marker passing inference architectures*, which facilitate parallel inference using a set of parallel rule units and their dynamic linkage through some form of marker passing. The fifth category is made up of architectures which fall into none of the above.

Logical inference engines were presented in chapter two in as much depth as will be required for the work to follow. This review begins with *production systems*, which were first presented in section 2.2.4, page 21. Here, two implementations are discussed in more detail; the first of these is the Distributed Connectionist Production System (DCPS) of Touretzky and Hinton, while the other is a hybrid between a symbolic production system and more traditional search methods, the SOAR architecture of Newell *et al.*

Following on from production systems, the properties of *semantic networks* are considered, a topic which is extremely important in the design of mechanisms for robust generalisation. The issues involved in inheritance are presented, along with attempts to resolve them.

The next section is concerned with marker passing inference architectures. Here, an number of alternative implementations are presented and compared, including work due to Shastri, Lange & Dyer and Sun.

Finally, we consider a set of architectures which fall into none of the main groups, including the commonsense reasoning architecture of Sun and the AURA architecture of Austin.

4.1 Production Systems

This review considers two production systems in more depth. The first is the general purpose problem solving architecture, called Soar, which uses a production system as a framework but can incorporate other algorithms as required by the problem domain. The second is the first attempt to construct a purely neural network version of a production system, called DCPS.

4.1.1 The Soar Architecture & Unified Theories of Cognition

In 1987 Allen Newell, pioneer of AI and leading exponent of symbolic computation, gave the William James Lectures at Harvard University in which he reported on the development of a 'candidate theory of cognition' called Soar. The content of those lectures subsequently appeared as a book "Unified Theories of Cognition" (Newell, 1990).

The philosophy of Newell and his co-workers was to examine the basic problem of problem solving with a view to producing a generic architecture capable of handling any kind of task domain. In this respect, it is similar to Newell's earlier work on general problem solving (Newell & Simon, 1963) though far more sophisticated. The resulting architecture could be viewed as embodying the fruits of more than thirty years of leading edge AI research.

The definition of the Soar architecture is sufficiently high level to avoid getting bogged down in task-specific detail but manages to be sufficiently detailed to be more than just an insubstantive thought-experiment. It provides insight at many levels as to how the cognitive process might be successfully modelled. The book is backed up with several case studies in which the Soar architecture is pitted against existing (application specific) expert systems with interesting and impressive results. The architecture itself is founded upon three key concepts.

The first concept is that the essence of intelligence is the ability to solve a new problem by applying a body of knowledge accumulated through previously encountering similar situations. This same idea underlies case-based reasoning which was only in its infancy at that time.

The second key idea is that when confronted with a situation in which there is no available knowledge that will permit the system to achieve its goal, Soar should define a sub-goal (whose purpose is to remove the current obstacle), solve this sub-problem and then return to the solution to the main problem. Sub-goals can contain sub-sub-goals, etc., *ad infinitum*.

In other words, the key is to break the problem down and bring all of the systems resources to bear on each part in series. This permits a problem of arbitrary complexity to be solved by a machine of finite resources an idea which has always been at the heart of symbolic AI (for example, the General Problem Solver of Newell & Simon, 1963).

The third key idea embodied in the Soar architecture is that the way to get around a impasse which cannot be resolved using the current body of system knowledge is to define a “space” which contains the required solution and then search that space until it is found.

Soar is implemented as a production system, similar in function to that described in chapter two but far more sophisticated in that it has a stack -based architecture permitting hierarchical decomposition of goals and tasks. The system also incorporates learning, which in this cases is essentially the generation of new productions (the term Newell uses is *chunking*, borrowed from psychology. The term *memoization* has also been used (Russell & Norvig, 1995)).

Newell draws attention to the basic difference between most AI and expert systems on the one hand and human cognitive performance on the other. It is the ability of a humans to store up and call upon the responses to previously encountered problems which is currently lacking in contemporary AI systems, he argued. He illustrated this with a graph of ‘Immediate Knowledge’ against ‘Search Knowledge’ the former being the product of previous experience (preparation) while the latter being the result of brute force search. A representation of this graph is shown below.

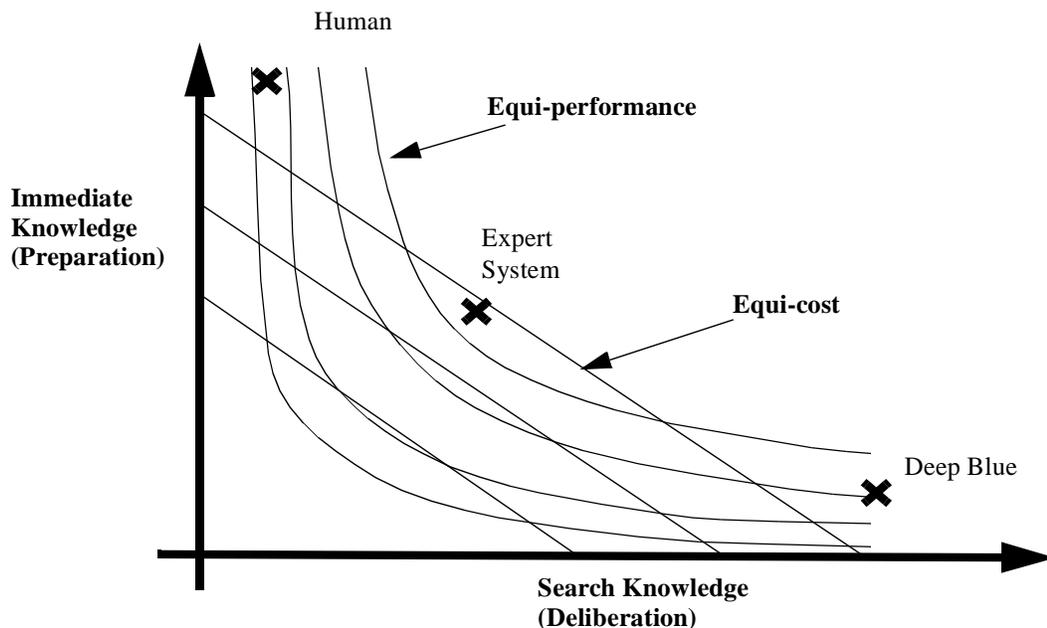


Fig. 4-0 Trade-offs in search and preparation (from Newell, 1990)

Brute force search engines, such as IBM's Deep Blue chess machine would be placed on the graph at the bottom right-hand side. While expert systems could be placed towards the middle of the graph, they still do not approach human cognitive skill, which, according to Newell, is almost entirely based on immediate knowledge.

When faced with a new problem domain about which it has no information, the expectation is that the Soar approach would mimic that of a human in a similar situation: at first only trial and error would be possible since initially there is no way to predict the response of the environment. But with each 'experiment' (whether successful or not) the system will have learned something new. In future, a sufficiently similar situation will permit the system to draw upon its knowledge of the earlier results and perform better. On the graph we see that, beginning at the bottom on the y-axis (possessing little immediate knowledge) and perhaps midway along the x-axis (having to perform a lot of search), with each new learning experience the Soar-based agent would progress upwards and to the left, depending less and less on search and more and more on its existing knowledge with the passage of time.

Unfortunately, Soar is not the answer to every question in cognitive science. It essentially glosses over a number of critical issues, as Newell himself was readily prepared to admit. The architecture has two principle failings, which are derived from the same source. First, it does not describe how to store knowledge structure in an efficient and coherent way. From an engineering perspective at least, this is a vital component of an efficient and practical system.

The second issue is the goal satisfaction itself. Newell describes how Soar would solve a problem by defining a sub-goal, establishing a problem space known to contain the solution and then searching the space until the solution is found. In principle this sounds like a reasonable approach to take. But how does one establish the problem space? What are the algorithms used to search it? Are they sufficiently general purpose to be applicable to any problem? If not, how is the correct one selected?

As an aside, in unrelated work, Minsky discusses the possibility of a mind as being composed of a large number of parallel agents (which he calls *demons*, perhaps in deference to Selfridge's Pandemonium Model (Selfridge, 1958)), each specialised in a different activity and applying themselves automatically to any part of a problem to which they are appropriate. Thus the decision as to which to apply in a given situation vanishes (Minsky, 1988).

The drawback of such an approach is that parallel specialists may not be an efficient use of resources. Certainly as Minsky describes them, demons seem to be far too specifically tuned to a particular function. How could such specificity develop? It could be argued that the specificity used in his examples is merely for pedagogical reasons and that a real system would use demons of greater generality leading to a more efficient use of resources.

Returning to the Soar architecture, we assert that the greatest impediment to its usage as a general purpose intelligent agent is its dependence on a rule-based representation of knowledge which is somewhat antiquated (essentially production system logic, although the notion of *chunking* is an improvement). The choice of knowledge representation must be such that large bodies of knowledge can be *efficiently, coherently and incrementally* stored and applied to new problems in an *efficient and coherent* manner. Though Soar sketches an outline which is sufficiently general to apply to any problem space, it is further asserted that it is the definition of the internal mechanisms of knowledge representation which present the greatest challenge to intelligent systems design and here Soar has little to add to existing methods.

4.1.2 Distributed Connectionist Production System (DCPS)

Around the same time as the debate on the proper treatment of connectionism was in full force, work was under way to map the basic idea of a production system into a neural network: the distributed-connectionist production system (DCPS) of Touretzky and Hinton (1988). A later extension of the model called BoltzCONS was applied to list processing (Touretzky, 1990). This latter model will not be considered since it adds little to the discussion.

At the time of its inception, Hinton and his colleagues of the PDP group were still in the process of discovering the properties and issues of distributed representations. The DCPS model is, arguably, more concerned with considering the issues of representation than of seriously addressing the problems of symbols-in-neural-nets, *per se*. The model is shown below in a diagram derived from the original paper.

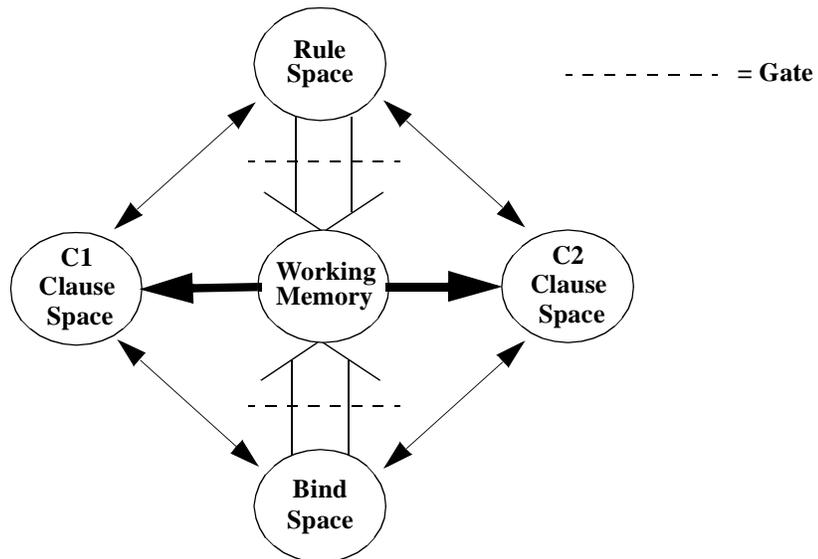


Fig. 4-1 The Distributed Connectionist Production System (DCPS).

Before considering the unique neural aspects of the model, let us first review the operation of the system from the symbolic standpoint. A set of 25 symbols was used, labelled **A** to **Y**. Like a standard production system DCPS had a working memory and a list of rules. The working memory was used to record which sets of three symbols were valid at any time. A triplet such as (**F A D**) is a means to encode a relation, **A**, between two symbols, **F** and **D**, and is thus sufficiently general purpose to be widely applicable in reasoning.

Rules were encoded in a separate rules space. The format allowed rules with two triplets on the left hand side, together with an unbounded set of triplet additions or removals on the right. This example is taken from the DCPS paper itself:

Rule-0: $(=x \mathbf{A} \mathbf{B}) \quad (=x \mathbf{C} \mathbf{D}) \quad \Rightarrow \quad +(\mathbf{G} =x \mathbf{P}) \quad -(=x \mathbf{R} =x)$

This rule, which includes a variable x (capable of taking any symbol as a value provided that the same symbol appears in each position of a given rule), will fire only if the two triplets on the left occur in the working memory (with the same first letter) and will cause one triplet to be added and another to be removed from the memory.

While the rules space and working memory are standard components of a production system, the clause spaces and bind space are unique to the neural model. Each of the two clause spaces is capable of taking on an encoding of one triplet that is active in working memory, focusing attention on it. These two selected triplets can then feed into the rules space, activating any rule which they match on the left-hand side. The selected rule then specifies modifications to working memory as dictated by its right-hand side.

Similarly, the bind space can represent a single symbol, which corresponds to the value of any variable that might appear in an expression. Thus, only a single independent variable is permitted in any rule (though the authors assert that this could be extended using further bind spaces).

From the network perspective, the architecture began life as a Hopfield network but evolved into a Boltzmann machine due to convergence problems. The neural populations of each representation space are fully and symmetrically connected with those of several others (as shown in the figure above).

During the normal operation the contents of working memory are fixed while the rest of the network is allowed to update its state as a Boltzmann machine. Beginning at high 'temperature' the cooling process allows the network to settle into a state consistent both with the rule which matches the active triplets and the appropriate binding for the variable (if any). At convergence, the working memory is updated and the cycle begins again.

Each type of representation space has its own hand crafted coding scheme. For example, working memory encoded triplets in a field of 2000 units. A single unit has a 'receptive field' of 216 possible triplets, since it is receptive to six possible symbols in the first position, six symbols in the second position and six in the

third, giving 6^3 or 216 combinations. Since the assignments were made at random, each triplet was represented by approximately 40 such units, and any two units were statistically unlikely to have even two triplets in common. Touretzky and Hinton present various scenarios, which showed that six or more triplets could be active in such a memory before the chances of interference between them become an issue.

The encoding of the clause spaces is similar since one-to-one connections exist between units in working memory and their counterparts in each clause space. However, mutual inhibition between all clause space units prevents more than a single triplet from being active at once, promoting the ‘focus of attention’ property. Interestingly, the rules space encoding used groups of units for each rule, but with no overlap between the firing sets for each rule. The authors claim that this prevents interference between rules that have similar LHS but very different RHS during the updating of the working memory.

Overall, the network acts as an existence proof of a neural network as a symbol processor. However, on the assumption that it should be possible to implement a Turing machine in a neural network, this demonstration was a somewhat moot point.

The real points of interest with this network are more concerned with the distributed encoding of the symbols and the energy that must be expended to overcome the problems that distributed representations bring with it. The encodings were necessarily hand-crafted for this network. If such care is always required to map a given problem then this method is clearly not appropriate to large practical systems. We assume, however, that it will be possible with further work to establish global rules of encoding that can be applied quickly to any problem without in-depth consideration of its content.

The potential benefit of the neural aspects of the architecture is that the serial search for ‘firing’ rules in a standard production system are replaced by a parallel search using energy minimisation in DCPS. As it is implemented here, however, there is a certain aspect of randomly trying combinations of triplets, rules and bindings until a consistent set is found. More careful tuning of the representation might have provided an opportunity to make a more focused (and hence faster) search possible.

The use of noise and an annealing schedule also rule this network out as a practical building block for the work to be presented in this thesis. Not only does imposing an external rhythm (which greatly modifies network behaviour) impose extra parameters on the system that may need to be tuned to each problem, but it also increases the total computation that must be performed by the system for any given step in the algorithm being executed.

DCPS is an important milestone in the development of neuro-symbolic systems but is perhaps best viewed as a foundation from which to address a whole myriad of barriers to the realisation of a practical system. Not least among these is the highly restrictive symbolic framework of the production system itself.

4.2 Semantic Networks and Inheritance Issues

The concept of a semantic network, both as a means to represent the hierarchical structure of class membership and as a framework for property inheritance, was presented as an overview in chapter two (section 2.2.5, page 23). This section describes the issues and limitations associated with semantic networks in some detail. Such details are important since at the heart of semantic networks lie a number of fundamental principles that will enable efficient common-sense reasoning. The work that will be contrasted is the NETL architecture (Fahlman, 1979), Touretzky's 'Mathematics of Inheritance Systems' (often abbreviated to TMOIS) (Touretzky, 1986), the evidential formalisation of Shastri (1988) and the method of 'Exceptional Inheritance Reasoning', (Al-Asady, 1995).

The major problem with the principle of inheritance occurs when a class may inherit different values for a given property by virtue of its membership of two or more parent classes. Two cases are identified, depending on whether or not the parent classes are ultimately both members of a third, common class, or not. When such a common parent exists, the problem is one of *redundancy*. Otherwise, it is a problem of *ambiguity*. Both cases are explained below, along with some of the proposed approaches to dealing with them.

The case of redundancy is typified by the 'Tweety-penguin' example, taken from Al-Asady(1995). (Note that the format for semantic nets in this section is different from (and more typical than) that used by the Russell & Norvig example of figure 2-2 on page 23. Instead, properties appear as nodes and it is often unclear whether a given node is a member, class or property. Despite this ambiguity, the format will be used given that it is the more standard notation).

In the network shown part (a) of figure 4- 2, we see that tweety, the penguin, is a member of the class of *bird*. Normally, by virtue of its membership of the *bird* class it would inherit the property 'flies'. However, the additional *negative* link from the subclass *penguin* to the property flies indicates that any member of the subclass *penguin* inherits a different value than the default.

In this case, the inheritance path is unambiguous, since the more specific information specified for the subclass of *penguin* overrides the less specific information inherited from the parent class of *bird*. We know that the information is more specific, since the path from *Tweety* to the property *flies* is the shortest (two links). Hence we know that Tweety does not fly.

In part (b) of the figure, extra information is provide that Tweety is a bird, creating a new link. This information is *redundant*, but still valid. However, it provides a direct path from Tweety to *bird* and hence to the property *flies*, and it does so in two links, the same path length as that which showed that Tweety does *not* fly. Hence, we have a potential conflict.

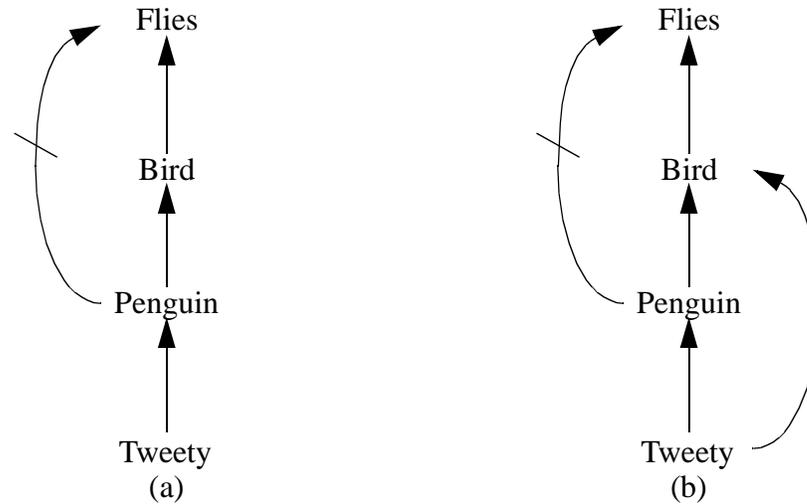


Fig. 4-2 An example of the problem of redundancy in semantic networks.

In trying to deal with such conflicts, a number of fundamental philosophies could be employed. Touretzky *et al.* (1987) distinguish between *credulous* and *skeptical* reasoners. To a credulous reasoner, if there are multiple paths from an object to a property (and/or its negation) then all such paths should be considered valid. The idea is that further processing might clarify the situation and prune away all but one path. Thus it is important not to ignore any potentially valid path. His own algorithm fits this model. In contrast, a skeptical reasoner assumes that if there are two paths leading to contradictory conclusions about the inheritance of a property, then neither path can be considered as valid. This idea leads to a reasoner that only deals in certainty, but would find limited application in most environments where handling ambiguity is vital.

To address the Tweety inheritance example, the NETL architecture, which acts by passing a marker and selecting the path that is the shorter, could give either inference, depending on which path it considered first. It is neither credulous nor skeptical, but merely overzealous, choosing the first result that it finds. As Al-Asady points out, its decision making process is then determined by details of implementation which are unconnected with the semantics. Such mechanisms are to be avoided.

Touretzky's approach which he called *on-path pre-emption* or *inferential distance ordering* (IDO) was to select one path over another based on the following principle. The path:

$$A \rightarrow \dots \rightarrow B \nrightarrow P$$

will pre-empt (take precedence over) the path:

$$A \rightarrow \dots \rightarrow C \rightarrow P$$

if we can establish the presence of a third path showing that B is itself a subclass of C:

$$A \rightarrow \dots \rightarrow B \rightarrow \dots \rightarrow C$$

thereby proving that membership of B provides more specific information than mere membership of C.

Off-path pre-emption (Sandewall, 1986, as cited in Al-Asady, 1995) allows the path from A to C to contain multiple links, whereas on-path pre-emption permits only a single such link. According to Touretzky this changes the semantics and can result in counter-intuitive inferences.

We now briefly present the alternative type of conflict in semantic networks, the case of *ambiguity*. This is often illustrated using the well-known ‘Nixon diamond’, which was presumably topical at the time when it was first used. This is shown below.

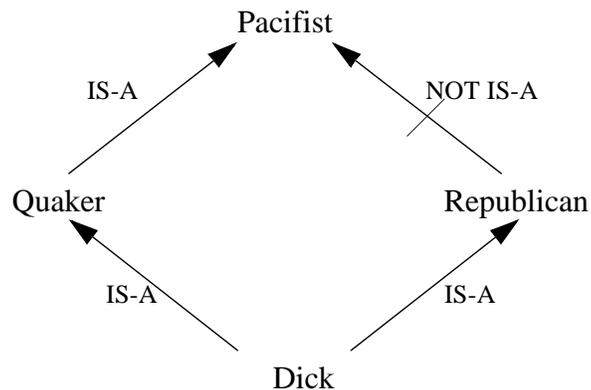


Fig. 4-3 A problem of semantics: The Nixon diamond.

In the figure, the object ‘Dick’ is a member of two different classes. Dick is both a *quaker* (and hence should inherit the property *pacifist*) and a *Republican* (and hence should not be a *pacifist*). In such conflicting situations, how does the system decide which (if any) of the property values should be assigned to Dick?

The new difficulty introduced here is that the two classes *pacifist* and *quaker* are unrelated. Nothing in the structure of the graph gives a means to decide which of the two candidate inheritance paths should be chosen and common sense would seem to indicate that extra information concerning the pre-disposition of one type of membership to dominate the other would be needed. Even then, opinion among individuals as to the relative strength of such affiliation is probably divided for many cases of interest (such as the quaker/republican case above).

To explain one type of failure of intuition AL-Asady defines what he called acquired properties. In one illustration (called the George example) George is both a Marine (and thus drinks beer) and a chaplain (and thus does not drink beer).

According to common agreement, even though being a marine-chaplain is a more specific class than merely being a marine, George would more likely inherit the property of drinking beer. But from Al-Asady's perspective, this is not a violation of the laws of inheritance because the property drinks-beer is a special, *non-inheritable* property and therefore falls outside the normal laws, requiring individual treatment.

But surely this belief is tantamount to an admission of failure of the whole formalism of semantic networks? If a mechanism admits common exceptions that fall outside those that it can deal with, then we must conclude that the mechanism is inadequate and must be re-structured.

Both Touretzky and Al-Asady point out that more complex examples (which might combine multiple cases of redundancy and ambiguity in a single inheritance path) are certainly possible and would require pre-empting several paths. They show that the order in which paths are pruned can influence the final result of an inheritance, which is surely another sign that the underlying mechanisms of semantic networks are flawed.

To try to standardise the mechanisms of inheritance, Al-Asady proposes a formalisation that he calls Exception Inheritance Reasoning (EIR). Here, he supposes that any sub-class that does not conform to its parent class is in fact conforming to a different 'virtual' class called the exception class, Ψ .

Thus the example semantic net above can be represented thus:

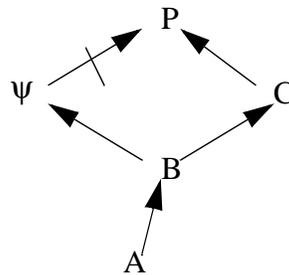


Fig. 4- 4 Exception inheritance reasoning (Al-Asady, 1995).

The exception class is a notional concept. It acts to regularise the rule of inheritance providing a uniform algorithm for dealing with inheritance issues of several types (not all of which have been detailed here). A further extension using a form of weights facilitates more intuitive behaviour. None of the proposed mechanisms corrects the problems of the non-inheritable property, or the multiple solutions based on the order of path pruning, both discussed earlier.

A parallel but distinct approach to semantic network is that of Shastri (1988). His principle strategy was to add a more complete evidential formalism to semantic inheritance allowing the strength of inheritance of a property by a given class to be quantified. This leads to far more intuitive results than were obtained in the all-or-nothing inheritance schemes already described.

In overview, Shastri's inheritance graphs include structure at each node (corresponding to an object or class) which list the properties possessed by that object/class, the possible choices of value for each property and the relative number observed for each value. These observed numbers can be used to form relative probabilities. For example (from Shastri, 1988), given the class of Apple, with one hundred examples observed, sixty of which had the property colour = red and forty of which had the property colour = green, it is possible to make more reasoned choices regarding the inheritance of the property by examples of the class, apple 'A-5'. using relative probability, the most likely value of property colour for object A-5 is Red, in the absence of more specific information to the contrary.

The methodology is based on conditional probability estimation, and thus has an established body of theory already in place. With such a framework it is relatively straightforward to make choices even in the previously ambiguous Nixon-diamond example, since it requires only foreknowledge of the conditional distributions for quakers and republicans with respect to pacifism and the individual *a priori* probabilities of those classes.

Overall, the principle of semantic networks is an appealing one since the goal is to reduce the amount of information that needs to be stored about every object or class in the network and facilitates generalisation to new examples. As such, it should form the conceptual centre-piece to an efficient knowledge processing architecture.

It is clear however, at least to researchers in the connectionist camp, that the all-or-nothing inheritance based on a careful pruning of the semantic tree for each case is not only a bottleneck (due to its serial nature) but also fragile and prone to failure. The evidential formalisation seems to hold far more promise since it has the ingredients for robustness and it should give a sound answer in circumstances where non-evidence based schemes fail.

Even so, the mechanism of inheritance in evidence based networks still requires a significant amount of computation for each inheritance decision. Shastri seems to avoid this problem by advocating a highly parallel network in which each class or object is allocated physical resources. Thus computation also occurs in parallel. While this may reduce the actual time required for any given inheritance (a fact much lauded by Shastri), it does not appeal as a practical solution for a very large knowledge base in which only a fraction of the stored knowledge is needed for any given problem. This issue will recur again in the discussion of marker passing architectures (section 3.4.4, next). Chapter eleven consists of a lengthy discussion on the issues of neuro-symbolic computation and will use the drawbacks of the Shastri system as one example from which to draw useful conclusions of the direction that we need to take.

As a part of the next sub-section, which discusses marker-passing architectures, Shastri's extension to his work on semantic networks to reasoning systems will be considered.

4.3 Marker Passing Inference Architectures

The next few pages will consider the last of the large classes of reasoning systems to be considered in this thesis: models based on marker passing as a means to achieve variable binding between rules during processing. Here, we consider three different networks. Unlike the work on recursive distributed representations presented in the last chapter (section 3.3.5, page 76) there is little sense of progression in this work, merely three different investigations into the same principles. Each led to networks which display properties worthy of discussion, and so each will be considered in turn.

4.3.1 Lange & Dyer's ROBIN Reasoning Architecture

The discussion begins with the work of Lange & Dyer, called ROBIN, which also serves to present the underlying concept of marker passing (Lange & Dyer, 1989a; 1989b). The basic principles of such architectures are taken from first-order predicate logic (FOPL) in that knowledge is encoded as a set of rule-like entities (called frames) which themselves consist of several variables (called roles), each of which can take on a value in any given instantiation, (the binding). The frame performs its inference by being active. It does this by two means: first it is activated by another frame whose activation could imply that this frame should activate as a logical consequence and second, that it is more successful at binding the values propagated forward by the activating frame than other frames which compete with it for dominance.

As each frame activates, it passes on to any and all frames which might be a logical consequence of its activation, a confidence value. This value is used to provide support for all logical consequences of the frame, with frames of higher confidence being chosen over frames of lower confidence when the system must select which is the preferred of two or more possible explanations of the given facts. This allows the chain of inference to preserve the relative confidence which is held in each of its intermediate results, which in turn makes it possible to resolve conflict between two competing theories which interpret any given set of data.

The example used by Lange and Dyer was the interpretation of potentially ambiguous phrases, such as the following:

P1: "John put the pot inside the dishwasher" and

P2: "because the police were coming."

where from P1 alone a normal person would infer that the pot is a cooking pot in a normal domestic scene, but with the addition of phrase P2 the word pot takes on the alternative meaning of marijuana and the interpretation of John's motives are to avoid the detection of an illegal substance by the police. The task that Lange and Dyer set themselves was to create a network capable of inferring John's motives either based on P1 or both P1 and P2. In the latter case, they point out that to do so requires a chain of inference based on knowledge not included in the sentence itself.

Their architecture proposes that the possible interpretations of facts are encoded as frames, which describes concepts as collections of roles and in any instantiation of the frame each role must be bound to a particular value.

An uninstantiated frame is thus a general rule and an instantiated rule is a particular assertion about the interpretation of a given set of entities. The figure below illustrates this with a portion of Lange & Dyer's network:

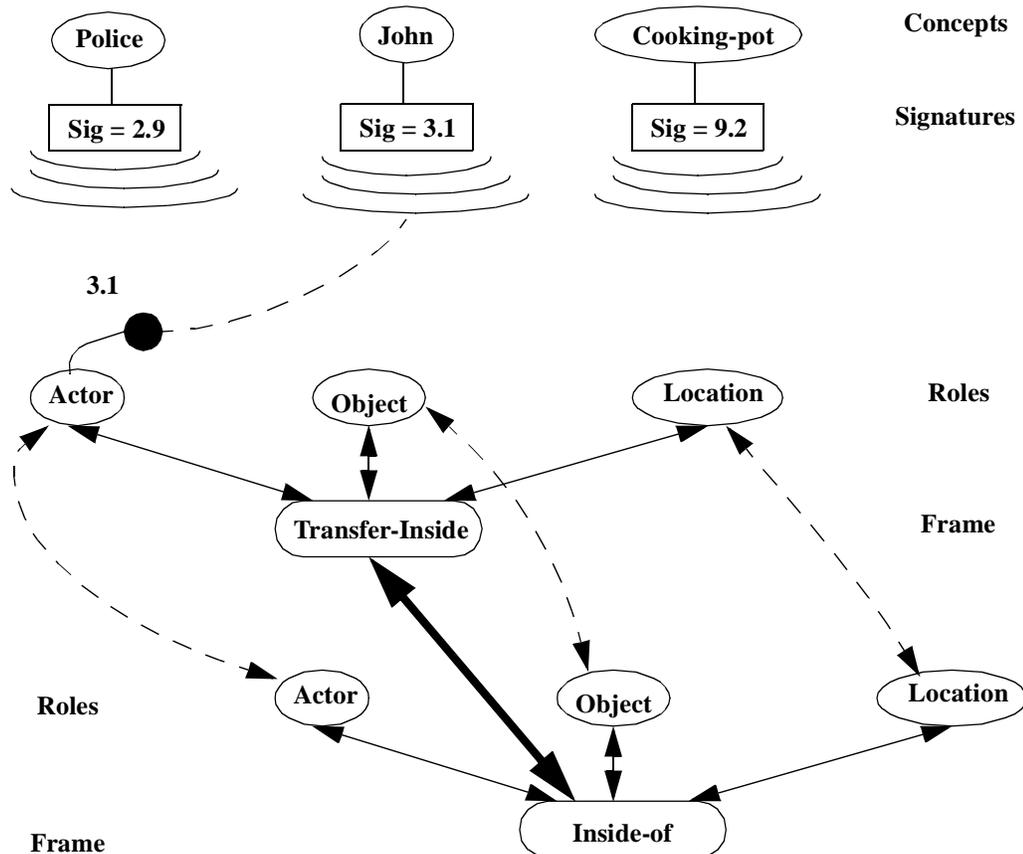


Fig. 4- 5 A fraction of Lange & Dyers' inference network

The figure shows that individual entities that play a role in the computation (such as 'John', 'the police' and 'cooking-pot' in this example) are each assigned a unique signature. This is the marker that will be passed to indicate that the entities is being bound to any given variable during processing. We see that the value representing John is 3.1. This does not change throughout the processing, but has no significance other than being a unique value so that we can identify the token for 'John' wherever it might appear.

A frame, such as 'Transfer-Inside' has an activation value, roles and bound values for those roles. Each role (such as actor, object and location in this example) is generic but for a given instantiation must be given a specific value (a binding) which is selected from the available set of entities.

The activation value indicates how much confidence the system has that the frame, with its given set of bindings, is a valid interpretation of the current scene. Closer fitting bindings tend to produce higher confidence, while badly fitting or missing bindings result in lower confidence.

When a frame is activated, it tends to activate more frames which are possible logical consequences of its activation. The frame's bindings (i.e. the signatures of objects that are bound to each role) are propagated forward to those frames that it activates, which is where the variable binding comes in. Chains of inference can be created in this way, linking instances of frames and propagating the signatures for which a given inference is true.

From any given set of facts many interpretations may be possible (as is the case in the 'pot' example used by Lange & Dyer). In such cases, multiple frames can become activated in parallel and must compete for the right to be the preferred interpretation. Only the preferred interpretation is permitted to propagate its bindings to other frames and thus new information which changes the preferred frame can invalidate a whole chain of dependent conclusions requiring recalculation.

The network itself was described a localist connectionist architecture. Individual frames are realised as parallel entities and computation is effectuated by the spreading of activity between them.

We note that much of the interpretation of the computation performed using this network is made by reading the labels attached to each frame. This is a criticism that applies to other networks to be reviewed in this section. Due to the commonalities of the networks (and the similarity of the analysis) criticism will be presented together in the next sub-section. For the Lange & Dyer architecture, it is worth noting the unique features that it proposes: the signature, used as the identifier for each object of instance, is a unique, real number.

In their papers, Lange and Dyer assume that each object known to the network could have a unique signature, fixed for all time. This seems unreasonable for a huge knowledge base since it requires the resolution of representation for each signature to depend on a property of the whole database not just on the size of the 'working set' of entities under consideration. Furthermore, it demands that each newly created object be assigned a number. How does the system handle the fact that objects may be just variants of an existing type? Does it create a unique identifier for every object that is presented regardless of its category? Despite the authors' assurance that fixed signatures for each object poses no problem, it is really an implementation weakness of the system.

4.3.2 Shastri's SHRUTI Reasoning Architecture

The second marker passing architecture that will be considered is that of Shastri & Ajjanagadde (1993), and later Ajjanagadde (1994) and Shastri (1996, 1999). These papers follow on from Shastri's work on semantic networks presented earlier (section 4.2, page 92). As mentioned in the earlier discussion, Shastri's variant of semantic networks was the most powerful at that time since it

combined not only the absolute values of membership or non-membership to a class, but a quantified evidence value based on a form of likelihood of inheritance. This allowed it to have something useful to say even in ambiguous circumstances that would confound less descriptive networks. It is unfortunate, therefore, that much of Shastri's work in the 1990's has been to lose sight of the evidential framework and to rely on all-or-nothing inference for his work on reasoning architectures.

The reasoning network proposed by Shastri and Ajjanagadde (hereafter S&A), which they called SHRUTI, is based on the same principles of marker-passing described for the ROBIN network. SHRUTI augments this with more powerful handling of roles, variables and variable binding. In addition to being able to reason with stored knowledge, SHRUTI permits questions of a certain form to be posed which is something not handled well (arguably, not at all) in ROBIN.

In their 1993 paper, S&A begin by stressing the need for *reflexive reasoning*, a look-up-like mechanism in response to most queries (even symbolically formulated ones) that is therefore not based on complex serial processing. This approach not only follows earlier work by Shastri (1990), but is similar in philosophy to the chunking principle of AI, embodied in the transition from search knowledge to immediate knowledge modelled in Soar (see section 4.1.1, page 86).

Furthermore, a key requirement of the performance that S&A demand is that the time required to make any inference must be independent of the size of the knowledge base. This is a stringent requirement and impractical for a network with finite resources due to the computational effort required to combine facts as the knowledge base is scaled up. We return to this point later.

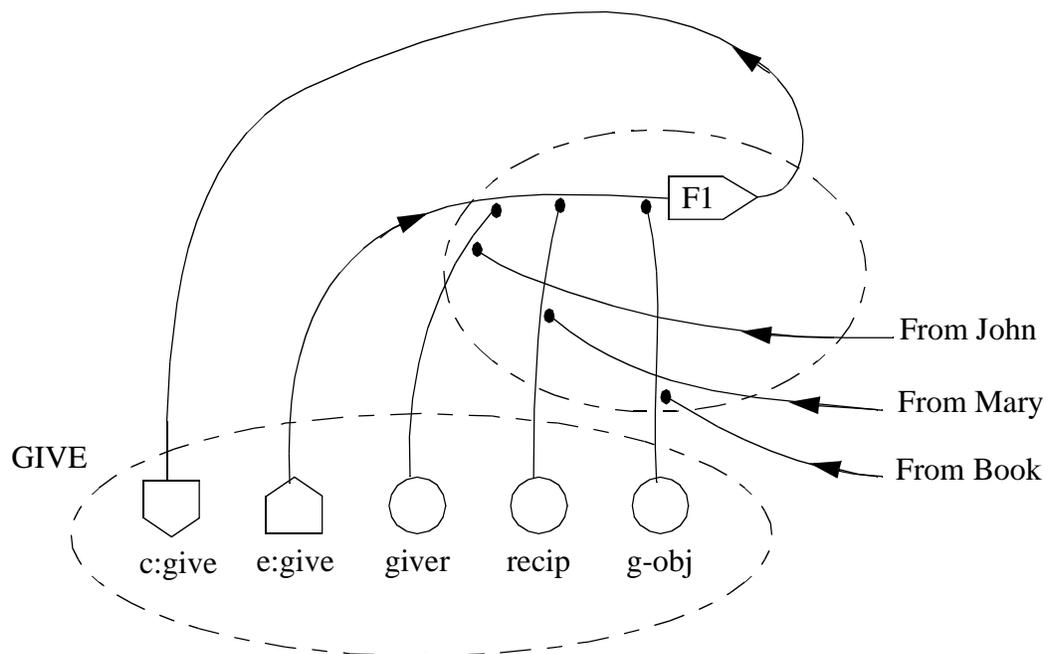


Fig. 4-6 An example of a predicate and a fact in SHRUTI

The architecture notionally assigns parallel hardware to each different part of the conceptual and variable binding process, as is illustrated in the figure above which shows the structure of both a predicate and a fact, as taken from S&A(1993).

In the figure, the lower oval is the predicate for *give*. It can take part in reasoning about many instances of giving simultaneously. Meanwhile the upper oval is the encoding of the single long-term fact “*John gave Mary the Book*”.

The predicate for GIVE has three roles, the giver, the recipient and the object and binds the entity appropriately. The node e:give is the enabler for the predicate *give*, used when in query mode (described later). The node c:give is the collector node which fires when any fact associated with that predicate is valid.

In each fact node there is a unit which is hard-coded for a particular fact. The fact F1 is tied to the predicate *give* receiving connections from the enabler node and role nodes of *give*. In addition, the output signal from the fact is sent to the predicate’s collector node, c:give. For a particular fact (and there can be many sharing a predicate) each input from the role nodes coming from *give* are gated (inhibited) by connections from the actual objects that this fact is binding. The signal gets through only if the phase of the role signal and the object signal are the same (the phase is described in a moment). In the figure, a connection from the node for *John* is physical made to the *giver* role of F1, so F1 encodes something about John as a giver.

To understand how these nodes work together during reasoning, it is necessary to explain the binding mechanism itself but first we must consider how multiple predicates are connected to form a potential chain of inference. An interconnected network of predicates is shown below, again taken from S&A (1993). It shows how the activation of one predicate can be fed forwards to other concepts that can be logically inferred from it. Note these connections form generic rules, since at this point no fillers have been bound to any roles.:

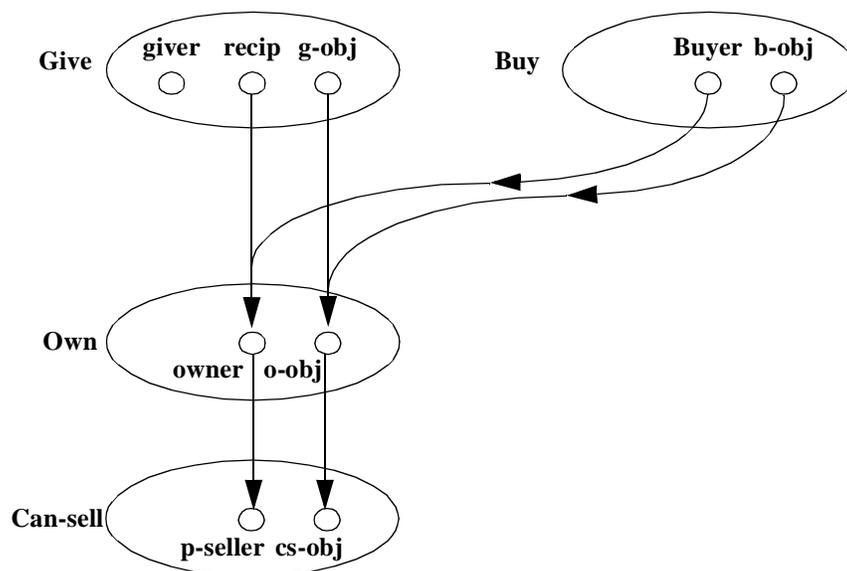


Fig. 4-7 An example network from SHRUTI

Like Lange & Dyers ROBIN model, computation proceeds by a spreading of activation between the concept along routes defined by the network's connectivity. The interesting difference is that rather than use the signature transmission system that Lange & Dyer have (with each entity represented by a unique real number), S&A opt for what they assert is a more physiologically reasonable system, based on phase-coded binary markers.

To do this, they impose a clock on the updating of the network and divide the cycle into a fixed number of phases. Each phase can carry either a zero or a one and the presence of a one indicates the signature of one entity. Thus the number of distinct entities that can take part in any one reflexive inference is equal to the maximum number of phases sustainable by the network.

A phase is allocated to each entity taking part in the computation and they begin sending out their respective signatures. To test a particular fact, we active the enabler node of the appropriate predicate. This in turn tries to activate all of the facts to which it projects (all facts about the predicate *give* in this example). If the phase signal from the role nodes of the predicate (i.e. *giver*, *recip*, *g-obj*) exactly match the signals coming from their respective entity nodes (i.e. *John*, *Mary*, *Book*) this means that a perfect match has been achieved. Thus the fact encoded by F1 is true and it activates, passing its signal back to *c:give*. If any aspect of the bindings do not match, the activation of node F1 is blocked.

In query mode, the system can answer yes/no type of questions based on predicates and stored facts. The example given in S&A(1993) is based on the predicate *can-sell* which has two roles, the owner *p-seller* and the object *cs-obj*. This predicate is a consequence of the predicate *own*, so the inference can be made that is someone owns something they can sell that thing. To ask the question: 'Can Mary sell the book?' we must see if the predicate *can-sell* is used in a valid fact with the bindings of *Mary* and *Book* to its roles of *p-seller* and *cs-obj* respectively.

To do this, the enabler node of *cs-sell* is activated, and two phases are required. One phase is fed to both *Mary* and *p-seller* (since they must represent the same thing) and a second phase is send to both *Book* and *cs-obj* (for the same reason). This has the effect of a query for that predicate. And the activity now spreads both to facts based on that predicate and to other predicates that act as antecedents to it. If the query matches any known fact for that predicate then the collector node for that predicate activates and the answer to the query is 'yes'. This would occur if we had explicitly stored the fact that Mary owned the book. In this case, the only stored fact is that John gave the book to Mary (fact F1) so *c:can-sell* remains inactive for now.

The enabler of *can-sell* enables predicate *owns* via its enabler on the next cycle because *owns* feeds in to it. This predicate tries to activate its associated facts in a similar fashion to *can-sell*, with the same results. It is only when the activity reaches *give* on the third cycle that the fact F1 activates, indicating that John gave Mary a book. This provides sufficient support for an answer to the original question, and this is propagated down the collector nodes from F1 to *c:can-sell*, answering the question in the positive.

In essence, the system is performing backwards chaining, looking for conditions that will enable a definite ‘yes’ answer to the question (by default the answer is ‘no’). At each predicate, if a fact exists which matches the search criteria then the search is over and this information is propagated back along the chain of reasoning to the start.

Rules with multiple terms in the antecedent such as P AND Q can be encoded in a similar way, only the collector node of the consequent needs a threshold of two (or greater, but equal to the arity of the rule) so that both predicates must fire to active the rule.

As S&A point out, the use of phase coding is superior to the signature scheme of ROBIN since phases can be allocated dynamically for a given computation. In contrast, ROBIN has a fixed allocation of values, one per entity in the system. For a large but realistic knowledge base with 50,000 such entities, they note, 16-bit values would be passed between nodes even if only a few entities were actually needed at one time.

The phase scheme allows a single bit wire between nodes to carry binding information. The limitation on computation is then defined as a function of the number of things that the system can consider in parallel rather than as a function of absolute knowledge base size. This is an important point and brings home once again the need to consider implementation as a central issue in architecture design. Another advantage is that the use of phases permits a single predicate to take part in multiple bindings simultaneously which implies at least some saving on resources. (But note that the facts themselves are encoded atomically, with one fact node per fact).

Since there are many common criticisms to make for all of the marker passing networks, further discussion of this network is left until the next sub-section. First, we consider the final presentation on marker passing network, that of Sun. His work in this area spans more than a decade and can be divided into three related elements. The first of these is an implementation of a reasoning system incorporating variable binding which will be considered here (Sun, 1992).

Later in this chapter, other elements of Sun’s work will be discussed. The first of these will be a reasoning architecture based on causal reasoning, which is essentially a formalisation of default reasoning (Sun, 1994). The other is an architecture to combine rules and a similarity measure into a single framework, which he applies to what he calls *common sense* or *robust* reasoning (Sun, 1994).

4.3.3 Sun’s CONSYDERR Reasoning Architecture

Sun’s approach to variable binding is similar in many respects to that of Lange & Dyer’s and thus will be considered only briefly. His reasoning unit (called an assembly) consists of a node, C, indicating the confidence that the unit has in its bindings and one node per free variable, X1, X2, etc. to store binding information.

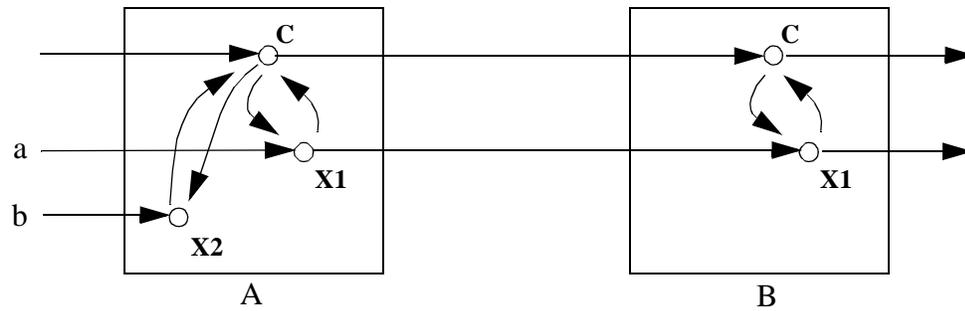


Fig. 4-8 An assembly from CONSYDERR

An assembly can represent one predicate. Rules between predicates are achieved by linking one predicate to another as shown in the figure for predicates A and B. The rule encoded in the figure is the equivalent of the following from first-order logic:

$$A(X1, X2) \Rightarrow B(X1)$$

If the rule fires when bindings a and b are applied (as in the figure), the resulting mapping is:

$$A(a, b) \Rightarrow B(a)$$

As Sun notes, a restriction on rules is that the variables on the RHS must be a subset of those on the LHS of any rule. This follows since the architecture only allows old information to be forwarded. This precludes the writing of rules such as:

$$C(X1) \Rightarrow D(X1, X2)$$

where the variable X2 is not passed on from the C predicate. In an extension of the basic model, Sun allows variables and constants to be generated inside a predicate but for variables such as X2 he states that the universal quantifier is assumed, that is that the variable X2 can take on any value. Is this what we would expect from such a system? What if we had:

$$\text{is_son}(\text{Fred}) \Rightarrow \text{father_of}(\text{Fred}, X2)$$

Surely, this inference implies that if it is true that Fred is a son, then we know that there is some (currently unknown person) X2 who is the father of Fred. This is an existential quantifier. The problem with this degree of freedom is that a normal feedforward reasoner such as CONSYDERR is not capable of then back-tracking or otherwise searching for evidence that will identify the appropriate value of X2.

This criticism also seems to be true of both Lange & Dyer's and Shastri & Ajjanagadde's networks although in all cases it would seem to be possible to introduce the concept using extra mechanisms.

The distributed nature of the binding nodes in CONSYDERR allow partial matching and generalisation which makes this network superior that those of both Lange & Dyer and S&A.

As a final point, we note that the bindings that are passed between predicate in CONSYDERR are unique identifiers, thus putting this model in the same category as ROBIN. Sun points out that the system could allocate, prior to processing, a set of values to any entities that will take part in that processing. So only active entities would have a signature. This is reasonable for the limited processing that will be executed on such a system, but is less appropriate for the type of system to be developed here in which there is no explicit 'start' and 'end' of processing and any entity could potentially be implicated in the processing at any time. These issues are discussed in chapter six.

4.3.4 Marker Passing Architectures: Discussion

The last sub-section presented a number of different architecture which share the common mechanism of marker passing as a means to perform variable binding and deductive inference. Each of the three sets of authors were able to demonstrate that far reaching inferences could be automated in a connectionist network (or at least a connectionist-like network). Without doubt, each represents a possible means of implementing variable binding in example domains such as have been presented in the last sub-section.

However, a considerable fraction of the benefits which appear to accrue from such networks are illusory, as this section will show. Examining the limitations of such an approach will be extremely helpful in identifying avenues for future work in the area of efficient, automated reasoning. Note that in this discussion the generic word 'frame' will be used to describe the computational unit (although both 'assembly' and 'predicate' have also been used by the authors). Any reference to a particular architecture will be clear from context.

The first criticism of marker passing network is that they are essentially an implementation of first-order predicate logic, with rules being applied to a knowledge set and the decision to bind a given value to a variable being conditional on the firing of those rules. The marker passing itself is thus an implementation detail. As such, examples of this kind are justification to members of the AI community such as Fodor & Pylyshyn that connectionist systems are merely implementations of purely symbol systems. Certainly, the neural level adds nothing to the computational account of the system's operation, which is based purely on the propagation of symbol values as a result of the firing of rules or predicates. The fact that there may be a quantitative notion by way of an associated activity value changes nothing in this regard.

Also, it is important to grasp that most of the apparent computation is merely implied by virtue of the labels that each frame carries. Examples such as 'Transfer-Inside' and 'Avoid-Detection' from Lange & Dyer's model, or Shastri & Ajjanagadde's *can-sell* predicate give the impression that sophisticated concepts are being dealt with, but this is not the case. All that the network deals with are con-

nected frames and predetermined propagation of bindings. The real intelligence is manifest in the design of the frames and their connectivity; that intelligence did not appear from within. (It is interesting to note that similar criticisms of work in AI was made by Drew McDermott over twenty years ago (McDermott, 1976)).

Furthermore, such networks, while handling Fodor & Pylyshyn's constraint of systematicity, seem to violate all of the others. Such networks do not display productivity since there is no combinatorial syntax. The limits of what can be represented are proscribed by the designer at the network's inception. The constraint which Fodor and Pylyshyn called *coherence of inference* asks that we ignore non-computational labels in the analysis of a system's computational capability. Doing so in this case robs the network of much of its perceived intelligence since the frame labels seem to carry most of the meaning of network output.

Next, we consider the cost of implementing such networks. In all of the network examples under scrutiny, each frame instance exists as a physical (or at least parallel) entity in the network. Starting from activity representing the input phrases or assertions, activity spreads throughout the network, activating frames as required by the building chain of inferences. Such a scheme is highly wasteful for a number of reasons.

First, for a given problem, it is likely that only a small fraction of the frames will be active. Those that never become active serve no purpose and their potential computational resources are wasted. Second, even for frames which are activated, they can remain in that activated state for an indefinite period of time while the spread of network activity switches on other frames. Until the activity 'wave' (for want of a better word) has reached the ends of the network almost all of the frames, active or not, have performed all of the computation that they need to. They then serve merely as memory to maintain the activity wave progressing ahead. It is an assumption (but a reasonable one) that the cost of implementing the computational elements of the frame is greater than the cost of the memory required to hold the results of that computation. Thus the fact that the computational resource is forced to act merely as a memory is a waste of resources.

A potentially more efficient approach could treat all frames as 'generic frame resources'. Selected from a pool of available (blank) frames when needed, a frame could be reused once it had computed its output. Only the output values need be conserved. Of course, one of the appealing elements of the original frame approach is that all of the frames act in parallel, scanning the bindings that are being transmitted, and activate themselves when they have found a fit for most of their roles. To mimic this, the system would need to allocate generic frames in advance of need by anticipating which frames might become active. Once activated, the frame could be returned to the pool leaving only its output as a permanent trace.

In a similar (but distinct) vein, it would be useful if we could benefit from the existence of frames which had similar 'meaning' (semantic content?) by overlapping in some way the physical or knowledge resources used. This does not seem to be possible with such an architecture.

It may be that the designers of these networks did not consider implementation details such as efficient use of resources as a key constraint. But from the point of view of this work, driven as it is by the requirement to be realisable and efficient, it must be a key constraint.

The next issue with marker passing architectures is the robustness of the knowledge encoding itself. Generalisation is implemented only at the level of confidence in the activation of a given frame. It is not clear how the network would react to variants of known inputs. In the Lange & Dyer example, what if it were not the police who were coming but the FBI? Do we need to duplicate every frame which makes particular reference to the police, but with the name FBI instead? What if it were the Police who were coming, but the rock band rather than the law enforcement agents? Such an architecture lacks the ability to respond flexibly to variants of known inputs, a consequence not only of the localist nature of the encoding but also of the fact that concepts which should be implemented as variables find themselves hard coded into the concept name attached by label to the frames.

Next, we consider learning. Since the frames are static entities whose meanings are defined not only by the links that they make with other frames but also by the labels that they carry, it is not at all clear how such a network could learn in an autonomous fashion. Certainly, it cannot inspect its own frame labels and there appears to be no gradual method to modify links (other than to change the strength of connection of an existing link between two frames, which offers only a limited degree of freedom).

Stepping back and looking at the philosophy behind these networks as compared to that which is being promoting in this thesis, it is easy to see why these marker passing networks do not fit with the requirements that have already been laid out. Such networks embody rules and perform logical rule execution. In each case the architecture is utterly distinct from the data it produces and for the target applications this is perhaps acceptable.

But we are looking for something more here. What we want is a unification between the output of the network and its form. We want the system to manipulate not just the result of applying the rules, but the rules themselves. We want the output of the network to be capable of representing the intention to modify its own fabric in a certain way and then be able to do so. Furthermore, by these acts of self-modification we expect the system to improve itself, increase its performance and make itself more able to achieve its goals. All of these elements are missing from marker passing networks. By not taking the first step of actually representing the entities they purport to be processing in a manner which influences the outcome of each computational step, it is hard to see how these networks can ever achieve such objectives.

4.4 Other Important Work

This section includes a number of important contributions to the study of reasoning systems that do not fit into the earlier categories. They are grouped here for convenience.

4.4.1 Austin's Associative Memory for Reasoning (AURA)

This work is an extension of Smolensky's tensor product scheme and illustrates its application to practical systems. Austin combines Smolensky's tensors with his own binary correlation associative memories (Austin & Filer, 1995; Austin, 1997) developed for a related (but distinct) purpose. The goal of the work is to investigate the properties and limits of binary memories as a framework for symbolic reasoning. Rules of the form:

if (A=4) AND (B=TRUE) then (C = FALSE)

are encoded, where A is an integer variable and B and C are binary variables.

The structure of the network is built around two associative memories in series. The first maps the conditions (the left-hand side of the rule) to a unique vector called the separator. The second maps this separator to vectors representing the output actions of the rule. Note that ORed terms in the left-hand side are treated as separate rules since each term can generate the output actions on its own. However, terms from any given rule always map to the same separator vector as the first step.

The tokens themselves are encoded as N-from-M vectors, in which exactly N bits are '1' and the rest are '0'. This gives all vectors the same length and makes the task of identifying signal in noise much easier. Variable names have their own unique vector encoding, as do constants like TRUE, FALSE and the integers.

Bindings are implemented using the tensor product scheme. Since the elements of each input vector are either '1' or '0', the same is true of the resulting elements of the tensor.

$$A = [1\ 1\ 0\ 0\ 0] \quad B = [1\ 0\ 0\ 0\ 1] \quad 4 = [0\ 0\ 1\ 1\ 0]$$

$$\text{e.g. } A : 4 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} [0\ 0\ 1\ 1\ 0] = \begin{bmatrix} 0\ 0\ 1\ 1\ 0 \\ 0\ 0\ 1\ 1\ 0 \\ 0\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0\ 0 \end{bmatrix}$$

Fig. 4-9 Example vectors and tensor for AURA, from Austin (1995)

The figure above shows how a binding of the value 4 to the variable A can be made. The next step is to combine terms which are ANDed together. Austin proposes two schemes, one by using n tensors in parallel to represent the n terms in the condition, while the other superposes (by logically ORing terms) the n input tensors into a single tensor.

In both schemes, the number of '1's that feed into a single neuron in the middle layer is approximately equal to nN^2 . Using the same principles as the Willshaw memory (Willshaw *et al.*, 1969) this known signal strength makes it straightforward to threshold the neurons and obtain the correct separator pattern for a perfectly matching input pattern.

The system also permits partial matching using only a subset of the input terms and, as Austin points out, this capability of parallel partial match is a powerful feature of the architecture.

One drawback with using fewer than n terms to address the n -arity network is that the signal strength is less than that set during learning. To cure this Austin proposes n parallel input networks, one for each arity of input. When applying a query of an arity A, the controller would use the arity A network and apply a threshold equal to AN^2 .

Whether this is actually necessary depends on how the network should treat partial matches. If, given two rules:

A:TRUE AND B:TRUE \rightarrow S₁ and

A:TRUE \rightarrow S₂

followed by the unary query:

A:TRUE \rightarrow ?

we need to interpret what this query expects for other terms. If the query is interpreted in the purely logical sense then we read it as '*A must be true and there must be no other terms*' then Austin's approach of having multiple networks is perhaps justified. The problem arises because the AND mechanism in the term being represented is not reflected in the mechanism of retrieval of the memory: the results that would be obtained for each part of the term alone are superposed without interference.

However, if the query is interpreted as '*A must be true but other parts of the term (if any) can have any value*' then a single n -arity network can handle this. By replacing any missing terms in the input stream by a fixed potential of N^2 that is fed to each neuron in the separator layer, an unknown but perfectly matching variable-plus-binding is simulated. For a partial match which does not want to make commitments for unspecified parts of the input this is probably a better approach. Such a mechanism could be useful in the control system of an intelligent system.

Overall, Austin has applied a novel combination of the tensor product scheme and binary associative memories in a practical way. The fact that multiple memory traces can be overlapped in the same physical network and that a parallel search can be executed is a step forward compared to the Plate encoding scheme discussed earlier. This property seems to be overlooked by most researchers, which is unfortunate. It would appear that, for its target application space, the AURA architecture is a potentially powerful approach.

When considered from the point of view of work in this thesis, there is some similarity. First, the use of N-from-M coding as a controllable medium of representation. This coding scheme is the backbone of AURA just as it is central to the work in this thesis. Another common thread is that memories are stored in parallel across the same set of nodes. This similarity is not surprising given that both architectures have a common base in the Willshaw memory.

There are two notable problems, however. The first, as noted by Austin, is how the network handles partial match searches. Providing different networks for different arity searches may be appropriate for small-to-medium sized networks but for very large networks this seems impractical and inelegant.

At a more exacting level, we could consider the serial nature of the rule execution process. Even though an individual search in the memory is effectively a parallel operation, the architecture restricts the network to perform one such search at a time. While this is also true of every other architecture to be considered in this chapter, it may also prove to be a common shortcoming in the long term. (The outline of an alternative is sketched in chapter eleven (see section 11.4, page 319)).

The other restriction inherent in AURA as presented today is the need to select near orthogonal separator vectors, one for each rule. Once again, such an approach may be practical for the target application. For the work developed in this thesis, however, careful selection of vectors is a luxury that cannot be afforded. Instead, it will be up to the network to dynamically create new encodings as appropriate.

Having said that, the principle of separator vectors does exist in this thesis, in the fabric of the symbol encoding itself. As will be presented in chapter five, a central issue in the development of adaptive symbol systems must be the encoding of symbols and their evolution. The symbol must act as a target for some mappings and as the source for others. In this respect it serves the same purpose as Austin's separator vectors. The difference is that while Austin circumvents the encoding problem by computing them off-line, the network to be proposed must rely on internal algorithms to assign and modify symbols encodings dynamically in response to changing mappings.

4.4.2 Sun's Robust Reasoning

Building on his earlier work in reasoning using variable binding, Sun considered the issues involved in both common sense reasoning and in causal reasoning. The latter is discussed in the next section. Here, we consider his proposal on robust reasoning (Sun, 1994), which illustrates the possible synergies that could be obtained from mixing rule-based and connectionist architectures.

The goal of this research was model the kinds of reasoning that humans make everyday, relying heavily on assumptions and measures of similarity with known entities to fill in any gaps in knowledge. He distinguishes between two types of mechanism for associating a property with an object: deductive inference and similarity. The mechanism for deductive inference is well known and comes from symbolic AI. He uses the notation:

$$(A \rightarrow B)$$

to indicate the inference strength between A and B, meaning the conviction that B is true given that A is true.

The similarity mechanism comes from connectionism and is based on a similarity metric, in this case chosen to be the dot product between two vectors, representing the distributed representations of the objects involved. He uses the notation:

$$(A \sim B)$$

to indicate the strength of similarity, meaning the conviction that B is true given that it has a certain similarity to A which is known to be true.

Using these two mechanisms in combination, Sun goes on to provide a mechanism for a wide range of reasoning situations. Essentially, he creates a chain of inferences and estimates of similarity and merely multiplies the resulting confidence values together to obtain the overall confidence of the chain of reasoning. The fact that the mechanism can address so many reasoning cases is, in itself, impressive. For example, knowing that A is true, that A is similar to B (i.e. $A \sim B$), and that B implies C (i.e. $B \rightarrow C$), allows the confidence that C is true to be estimated:

$$C = A * (A \sim B) * (B \rightarrow C)$$

where '*' denotes ordinary multiplication. He implemented the network using a variant of the variable binding CONSYDER network developed earlier.

The formalisation covers the two elements that we associate with common sense reasoning, and thus provides a plausible explanation of this kind of problem solving. Does it have any drawbacks? There are a number of problems with this formalism, not in terms of the approach but merely with the simplicity of the operations executed.

The discussion on semantic networks made it clear that inheritance can be a complex issue, and yet very simple metrics based on vector similarity are used here. If an operation as simple as a dot product were to be sufficient, the onus is placed on the encoding of the features of the vector to correctly model the relative strengths of similarity to facilitate the correct decision in any case. (Although this would strictly be true regardless of the similarity measure).

A similar problem can be identified for this network as for the earlier marker passing architectures: the use of resources is inefficient and inference is serial. Even if the network can make inferences in parallel, there is no mechanism to act upon them in parallel. While the nature of the architecture does not preclude such an extension, it is not clear how such a mechanism would be integrated into the main data flow.

4.4.3 Sun's Causal Reasoning Architecture

In a section heavy with work by Sun, we briefly mention his investigations into causal reasoning. This work is based on earlier work by Shoham on what is essentially a modal logic (Shoham, 1990). He uses what he calls Fuzzy Evidential Logic (FEL) to model beliefs in a given fact, and defines a formalisation to write rules in term of *necessary* or *possible* conditions.

The result are fairly sound and show that it is possible to write self-consistent sets of facts in the form of Horn clauses. The two main objections to this approach are firstly the restrictiveness of the syntax (which is tuned to modelling cause and effect within fixed boundaries of time) and secondly the disjoint between the knowledge base and the results. This latter point is the same objection that was raised for the marker passing networks: that the framework for representing the output of the network (in the earlier case the activation value of certain frames relative to others while in this case the activity of neurons representing the degree of belief in a given proposition) is not the same as the framework for representing the knowledge itself (in the earlier case this was the definition of the frames and their interconnectivity while here it is the explicit listing of the rules).

The missing element is the closing of the loop that unifies the networks output with its structure, permitting it not only to evolve, but to be self-examining. This idea is synonymous with the refutation of the homonculus as the entity that, by examining the systems output, gives it meaning. Searching for this unification is a central theme of the last chapter of this thesis.

4.4.4 Miikkulainen's DISCERN Architecture

One model which tackles the problem of variable binding in a different way to that used in marker passing networks is the DISCERN script reading architecture of Miikkulainen (Miikkulainen, 1993; Miikkulainen 1994). DISCERN is an implementation of a script architecture, accepting natural language sentences and identifying which of its pre-stored scenarios most closely matches the data. Finally, the model is capable of answering questions by filling in the missing parts of queries from the details of the best match, again using natural language.

The network itself is highly modular, but is based around a hierarchy of self-organising feature maps. One set of maps is used to compress the incoming stream of natural language to a vector, while another encodes the knowledge base of the scripts themselves. These latter maps are arranged like a pyramid, with the smallest map at the top selecting between one of a small number of story scenarios (restaurant script, shop script, etc.) based on the input vector (the compressed story). Lower level maps select between different ways in which each scenario might unfold as a story until at the lowest level individual features encoded in the maps represent bindings for the objects and the roles they fill. Thus variable binding is achieved in a localised way: the firing of a single unit indicates a particular binding of variable and value.

The detail of how the network functions is both interesting and impressive, but is less relevant to this discussion. Variable binding in DISCERN succeeds because the number of variables and possible bindings for each variable is highly restricted. Overall, the feature map approach to variable binding is heavy on resources. Such a system would not scale well.

An approach to reasoning systems that does not follow the established rule-based representation of knowledge is that of case-based reasoning. Little work has been done to date to establish the feasibility of connectionist implementations of this paradigm. A notable exception to this is the work of Malek & Amy(1997) who propose a hybrid connectionist-symbolic architecture. The neural component performs the case match portion of the algorithm.

4.4.5 Aleksander's Artificial Consciousness

For the sake of completeness, a few points will be made on the Magnus project of Aleksander, which aims to provide insight into the nature of human consciousness through the development of what they describe as artificially conscious machine (Aleksander *et al.*, 1995; Aleksander, 1996). Such a treatment is cursory in nature and does insufficient justice to the scope of the project itself.

The Magnus project is concerned with the development of a neuro-symbolic architecture, which is currently trained by simulation in a virtual environment but is expected, in time, to be extended to training in the real-world. The core of the architecture is a state-machine made up of several interacting parts such as sensory networks, output effector networks and the inner state machine that acts as the integrator and central decision-maker.

Symbols (or icons as they are referred to here) are states of this inner state machine and are therefore distributed over many neurons (perhaps tens or even hundreds of thousands of neurons are envisaged for a full system). Complex strings of symbols can be put together by association (Aleksander's example being the concatenation of three symbols for "dog", "bites" and "man" to form the compound icon "dog bites man". Thus, Magnus exhibits productivity and the potential for systematicity, as one would require of such a system. This work clearly builds on his earlier work, such as his analysis of the Fodor and Pylyshyn/Smolensky debate in which he comes down firmly as both a physicalist (believing that the

mind can be reduced to and explained in terms of a mechanism) and a connectionist (believing that a neural network is the most practical way of achieving that goal (Aleksander & Morton, 1993).

By forcing Magnus to build up its knowledge base through direct experience, Aleksander seeks to avoid the common criticism of reasoning systems that they can never be aware of their actions and cannot be said to understand what the symbols they use actually mean because such symbols are not grounded in reality (Searle, 1992). Here, the icons are built up by direct observation of reality (or virtual reality in the present model) and associations are made by concurrence in time, or the connectedness of events (such as “dog bites man”) in a manner which Aleksander believes parallels human development. He cites many references from psychology to justify this viewpoint.

As well as tackling the issue of consciousness, Magnus is also intended to address other, related issues traditionally held to be within the realm of philosophy alone: the possibility of free-will, awareness of self and mechanisms required for emotion. All of these are interpreted in terms of state machines, the manner in which transitions are made and the learning processes they employ.

Offering judgement on many of the philosophical elements of Aleksander’s work is beyond the scope of this thesis. From the purely mechanistic point of view of the design of a reasoning architecture it is too early to tell if the project will succeed. Magnus has many of the elements that are probably required for success: it learns from experience and builds up complex symbolic structure by association.

It is similar in philosophy with the work in this thesis in that it is based on the postulate that the meaning of the symbol is specified purely in terms of the causal effect that it has on the systems state. This idea will be central to the architecture development of chapter five.

One criticism of the work is that some key problems have not been addressed at this time. Efficiency of memory storage is one such area, as is the interpretation and re-interpretation of facts in a more statistical manner to try to extract more complex views of the same observations. In both of these areas, how the data is represented and managed is critical to success. It is possible that Magnus will be able to learn to make associations but will be incapable of putting together the ‘whole picture’ from a set of apparently unrelated facts. One can conceive of a state machine to perform the former task, without it being capable of performing the latter. Particular attention is needed to achieve both and it is not apparent that this line of research has been pursued by the Magnus team.

Chapter eleven enters into much more detail on what, it is argued here, are the main issues that remain in the design of intelligent systems.

4.5 Summary of Reasoning Systems and Discussion

The section has reviewed a number of the main contributions to the understanding of reasoning systems. The review considered production systems, semantic networks and marker-passing architectures as well as a few contributions that fell outside those main categories.

Production systems were seen as a general framework for rules-based reasoning. The drawback of such systems are three-fold. First, the parallel approach to checking which rules are ready to fire masks the serial nature of the rule-firing itself. This bottleneck marks production systems as computationally too weak for our needs. Second, the forward chaining rules-based approach itself is too constraining. There are many types of reasoning that we would like to do (such as reasoning from effects back to causes) which cannot be easily accommodated in such a framework. Finally, their crude handling of system resources with all-or-nothing rules firing (many of which might be irrelevant to further processing) is perhaps a weak foundation for an architecture that must be scaled up to thousands or millions of rules for the types of systems we wish to design here.

The Soar architecture offers a more generalised framework since it allows other algorithms to enter into processing as appropriate. The main problem with Soar is that it is so general that it does not address any of the real issues that must be faced to design a general-purpose intelligent system. Once again, efficiency is not considered, nor is the way in which the system might build up its array of algorithms.

Semantic networks seem to stand apart from other reasoning architectures, perhaps because they have not yet been well integrated into architecture which perform chains of reasoning such as we find in a production system. The principle of semantic networks, that properties that have not been explicitly assigned to an object can be inherited from a parent class, is a sound one and bodes well for efficient storage and implementation. As was noted, however, there are still several drawbacks. First, a general purpose inheritance mechanism does not seem to exist yet. Those that do exist are complex and unreliable. The serial nature of the computation required for each inheritance operation is a major issue. What is needed is a mechanism that is not only reliable and covers all cases but works in parallel for many objects simultaneously and uses minimal resources for each. Such a solution (if it exists at all) would need to be woven into the encoding of the objects (i.e. the symbols) and thus places restrictions on the relationship between the encodings for the parent symbol and those of its children. Chapters five and eleven discuss this problem further.

At the present time, the marker passing architecture seems to be regarded as the most promising approach to the design of reasoning systems. It has the appeal of handling variable binding and hence, of exhibiting the systematicity and productivity required. However, this review has shown several reasons to doubt the efficacy of such systems when addressing the real issues that must be faced.

The efficiency of use of resources is far from optimal in such systems, and the outputs themselves are not in the same representational space as the knowledge base that generated them, making it difficult for the output of the network to affect its structure (vital during learning). Similar to the criticism of production systems, marker passing architectures handle forward chaining of rules only. They cannot search backwards, starting from results and identifying root causes. Overall, such architectures are probably not a fruitful line of future research.

Other approaches have contributions to make to the work in this thesis (and beyond). Austin's use of associative memory as a means of parallel rules search is one important example. Aleksander's Magnus architecture, building up complex representations of the world as combinations of existing objects is another.

Looking back on the various models of reasoning systems that have been described in this chapter, it is interesting to question the goals and assumptions of the workers involved. Each architecture modelled aspects of intelligent reasoning, and was, to a lesser or greater degree, successful in achieving the goals that they had set for themselves. But this begs the question of what they are ultimately trying to accomplish. We suppose that each group of authors was interested in making a contribution to the development of a general purpose intelligent system. To make such a contribution tractable, they selected a sub-set of the capabilities that one would expect from such a system and modelled those.

But an implicit assumption is that by addressing a subset of the overall problem one might produce a solution that is a subset of that final whole. This assumption is perhaps not justified. History has shown that time spent optimising a given approach can lead to a dead-end. Work on search methods is, arguably, one such example. The search approach was the backbone of AI research in the sixties and seventies but has been replaced by more analogical methods such as case-based reasoning on the grounds that search is simply not practical for real-world applications.

In the opinion of this author, so much of what has been reported on existing reasoning architectures falls into the same category. While the feats of deductive inference that they demonstrate seem to put us well on the way to realising the machine intelligence that we seek, in fact they may be merely setting aside for later the main issues and problems that will ultimately necessitate a total redesign of everything they have put in place. That having relentlessly march to the top of the hill, they will find that the mountain that they wanted to climb is really in the next valley.

The alternative is to stop for a moment and consider what we actually want from our intelligent system and then define a representation scheme that actually addresses these needs. Though the initial results may seem slower in coming than those that we are getting from the current lines of research, they might at least have lasting value. That is the approach that we wish to take here.

4.6 Conclusions

This chapter has provided a deeper review of the main contributions to our understanding of neural networks as reasoning systems. This completes the detailed literature review.

In some instances this review has been highly critical of existing work. It is hoped that the reader will appreciate that such criticism comes from viewing each piece of work in the context of the goals established in this thesis. Such goals do not necessarily coincide with those of the original author and so it should not be surprising that many proposed architectures were found wanting.

The overall conclusions were that current neural network approaches to reasoning systems were little more than implementations of existing symbolic architectures. A radical departure in thinking is necessary to make any real progress to achieving a truly neuro-symbolic system.

Key elements that are rarely mentioned in the development of most existing architectures include the efficient use of resources and the ability of the system to represent the intention to modifying its own structure and then to do so. These missing properties will be investigated in much more depth in chapter eleven.

The next chapter begins the work proper. It presents the majority of the knowledge architecture development, which in itself is the high-level driving force for the neural network development of chapters six and beyond.

Architecture Design: Data Representation & Manipulation

5.0 Introduction

This chapter will begin the presentation of the network that was developed to meet the goals laid out in the first chapter, a development which will consist of two contrasting lines of enquiry:

The first is to devise an architecture that will provide the functionality necessary to achieve the goals. In this case, the architecture must describe the conceptual form and structure of the data to be manipulated and the operations that can be performed upon it. As a baseline the system is expected to implement some form of rules-based processing, but not necessarily first-order predicate logic.

The second line of enquiry is to develop and characterise the neural network building blocks that will be put together to implement the architecture. It is important to know how they function, to understand the limits beyond which they fail and to be able to predict and characterise their behaviour when a failure occurs.

This chapter presents the majority of the development of the architecture (the rest being presented in chapter eleven). The role of this development is twofold. First it is to devise a possible solution which fulfils the goals laid out in the first chapter. The second role is to create a list of requirements and constraints that the implementation must meet in order to fully realise that architecture.

This chapter will be concerned with both the architecture and the thought processes that led to its development. The chapters which follow will then address the implementation, consisting of the development and characterisation of the neural building blocks. Finally, chapter eleven will propose solutions to address issues not resolved in this chapter. It is located at the end for two reasons. First because it draws on the results of the neural network development and second, because it is chronologically the newest part of the work to date.

5.1 Basic System Architecture

The figure below shows a high-level view of the system architecture, made up of input and output processors, long and short term memory and a control structure. The system acts in some kind of environment which is a source of stimulus and, presumably, is affected by actions performed by it.

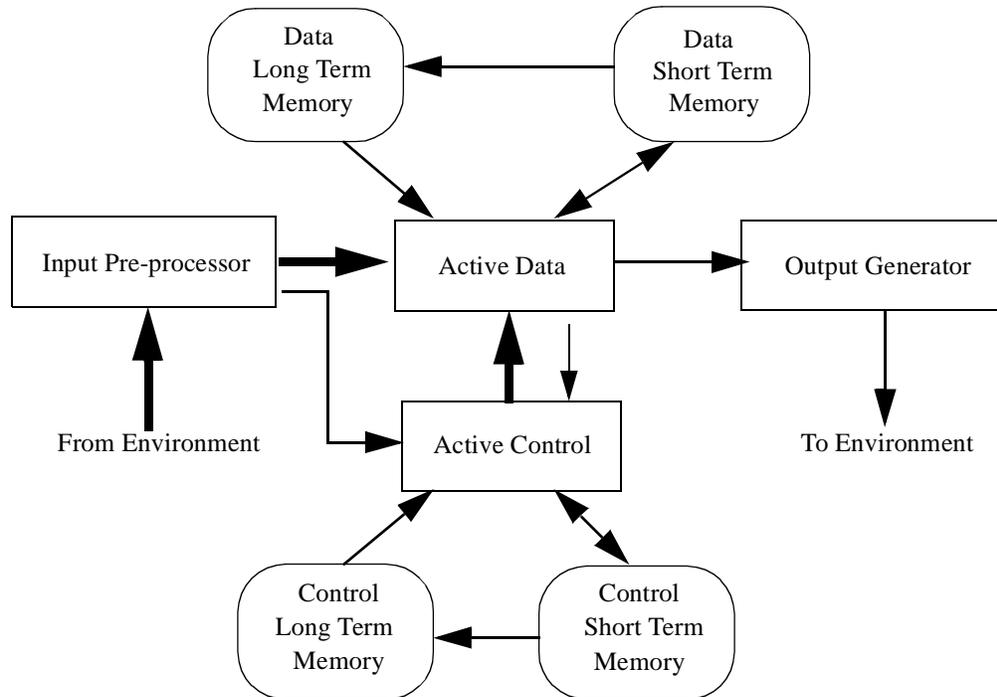


Fig. 5-0 High-level System Architecture

The core of the system is conceptually divided into control and data sections; each section itself consists of three parts: a long and a short term memory as well as an 'active site' where the two sections interact in the course of data processing. Note that information from both the long and the short term memories can directly affect the course of computation in both the control and data sections. By way of contrast there is no reverse traffic directly to long term memory; input to long term storage must pass first through short term memory, in line with the assumption stated earlier that information which is scheduled for permanent storage is a subset of that which has already been held transiently.

The heavy arrows in the figure indicate that the active data site is strongly influenced by the twin streams of incoming data from the pre-processor and control information coming from the active control area. It is assumed that the control area itself receives information both from the data area (in the form of results from each step of computation) and, potentially, some input directly from the incoming data stream in addition to its own memory stores.

Note how this architecture differs from those detailed in chapter four of the review. Here, implementation details such as the distinction between long term and

short-term memory are fundamental whereas for many neural network-based reasoning systems this distinction is either not made or consists in defining short-term memory as the current activity (with bindings) or its frames/schemas, etc. The design of this architecture is motivated by the desire to understand how knowledge can be stored in such a way as to wield it optimally: making it available when needed and preventing it from interfering with processing (and consuming precious system resources) when it is not. The division of memories is the first step towards achieving this objective.

5.2 Knowledge Bases

As previously mentioned, the memory of the system is conceptually divided into two parts; a short term working memory used as a temporary storage ground for ongoing computation, and a long term memory which contains a more permanent record of the information that the system has about its environment.

The format of the long term memory will be that of a knowledge database or tree, storing items and the relationships between them. Such a tree is shown in the figure below. It is seen to be a semantic net although the distinction between object and property has been removed, since it was never clear from the discussion of semantic nets why such a distinction existed in the first place.

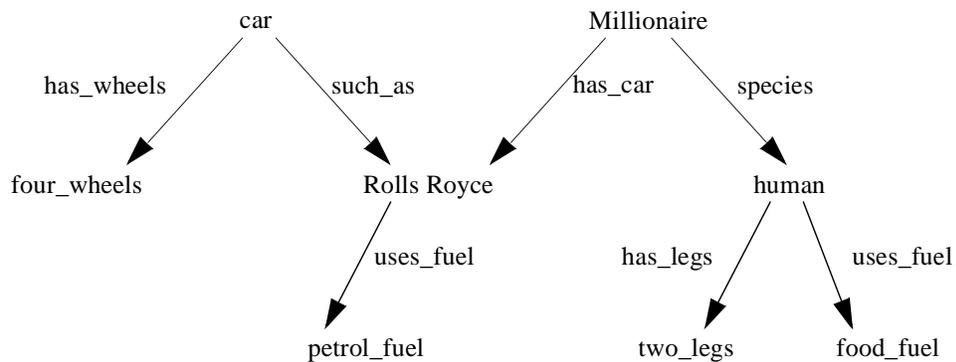


Fig. 5-1 A section of a relational database.

A knowledge tree, such as that shown in the figure above, allows objects and relations to be defined in a manner that is easy for humans to understand. Each 'node' consists of one symbol which represents an object, while the symbol attached to each transition is the control or probe that transforms an input symbol into an output symbol. For example, we could extract the following relation from the graph:

human $\xrightarrow{\text{has_legs}}$ *two_legs*

to indicate that given an arbitrary human being, we expect that the number of legs possessed by that person is two. While the format is in keeping with normal predicate calculus, one point of note is the explicit use of *two_legs* as the result rather than the generic *two*, each symbol embodying its meaning rather than depending entirely on context. This point will be developed in section 5.4.6, page 126, which discusses some of the requirements for a symbol.

One other important point about the tree structure needs to be made, concerning the issue of inheritance as typified by semantic networks. We see from the figure above that the tree defines relations between objects without reference to the generality of that relation. Notice that the statement

$$\text{Rolls_Royce} \xrightarrow{\text{uses_fuel}} \text{petrol_fuel}$$

was made even though the more general relation

$$\text{car} \xrightarrow{\text{uses_fuel}} \text{petrol_fuel}$$

might have been applicable. The problems of property inheritance remains one of central importance in AI. Even in this simple example the generalisation that was performed is not straightforward.

5.3 Data and Control

The compartmentalisation of control and data is a concept fundamental to system design. Even though it is perhaps difficult to imagine that the structure of the human brain can be divided along similar lines, there is extensive evidence from neuropsychology that certain areas of brain (such as the frontal lobes) are specialised in the temporal aspects of behaviour (control processes?) while others such as parietal and temporal lobes are more tuned to spatial tasks and the integration of percepts and relations, respectively (data processes?) (see Kolb & Whishaw, 1996 for references).

Thus to make such a basic distinction in the system architecture is probably not to become completely alienated from the brain metaphor.

Data and control in this architecture exist as separate networks, but with a high degree of coupling between them. In the simplest mode of operation, the flow of control might flip-flop from one network to the other and back again.

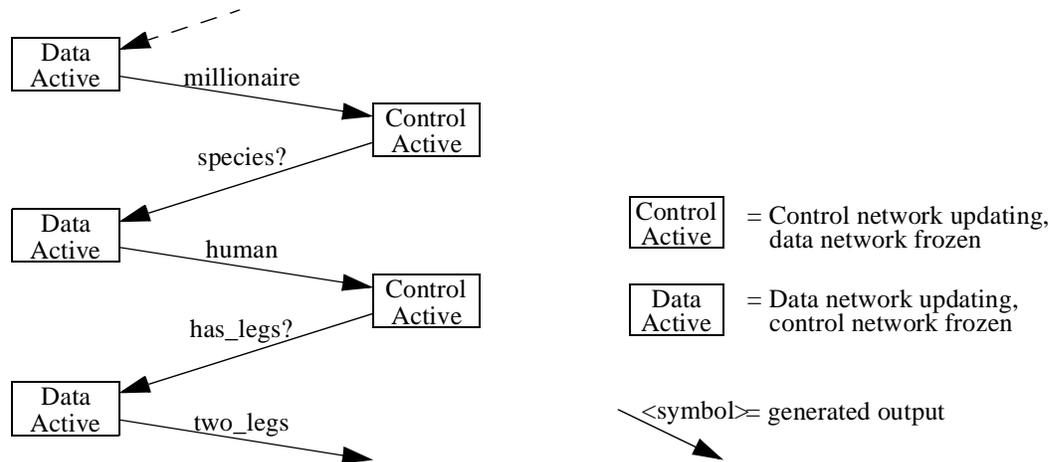


Fig. 5-2 Flow of activity from control to data networks during processing.

Thus, while the data network is active (which means that its state is changing), the output of the control network should not be undergoing a transition. It is tacitly assumed that, in this mode, the output of a network in transition is not meaningful to any other network. But such a mode is perhaps an overconstraint. Though we have no proof today, it seems unlikely that the brain works on lines which restrict one part of the brain to remain in one state while other parts undergo state transitions.

One might define conceptually more complex modes of operation in which both control and data are permitted to be in a continual state of change while still maintaining a meaningful dialogue. However, this simplification makes the network analysis more straightforward.

The almost symmetrical nature of the passage from control to data and back again emphasises the fact that we are able to treat control and data networks as the same, to first order. The output of one is the input to the other. It would have been possible to redraw the knowledge tree given at the beginning of this chapter, but viewed from the perspective of the control network. In that case the transition labels of the old tree become the nodes in the new one and vice versa. The advantage this gives is to permit a single network unit to act as both control and data structure. However, this blurs the distinction between data retrieval and computation. Both are reduced to two basic operations: pattern retrieval and the association of previously unassociated patterns. This important idea is explained more fully throughout this chapter.

Treating control and data networks as the same is important for this architecture for two reasons. First, because it renders implementation simple. We have only a single module type to analyse to establish its functionality and properties. Second because we adopt the philosophy that the symbolic control structures need to have the same properties as their data counterparts: to be compositional, systematic and productive, to allow generalisation and inheritance, etc.

5.4 The Symbolic Principle

This section will discuss some of the necessary properties that an entity must possess to act as a symbol. The argument will begin by highlighting general properties which are common to any symbol system, its symbols and the processes which manipulate them. Such properties are well known from traditional AI and this exposition will draw on much of the discussion from the literature review in chapters two and four. Additionally, the intent is to try to develop and justify extra qualities which will be necessary for a symbol system in a neural network context and which are either less explicit in the literature or do not exist at all.

5.4.1 What a Symbol Represents

To begin, here are some assumptions about what a symbol is and what functions it performs. A symbol, in the context of AI, is a representation of an object, idea or process. One symbol may embody other symbols and the relations between them to an arbitrary degree of complexity (figure 3-3 shows one example, drawn from linguistics). Each symbol within such an expression may itself represent another expression, and so on to an arbitrary depth of embedding.

When an expression containing symbols is manipulated by a set of rules (which are usually defined such that the truth value of the expression is not altered by the manipulation) the full expression represented by the symbol does not need to be made explicit.

This has two principal advantages. First, the irrelevant detail hidden behind the symbol does not interfere with the manipulation undergone by the symbol itself. Implicit in this statement is the assumption that the symbol is in possession of relevant properties, derived from the hidden expression, which inform the process performing the manipulation of what it needs to know to do so correctly. Second, it permits a system of finite resources to handle expressions of arbitrary length and complexity by permitting them to be broken down into a hierarchy of smaller pieces which could potentially be handled serially.

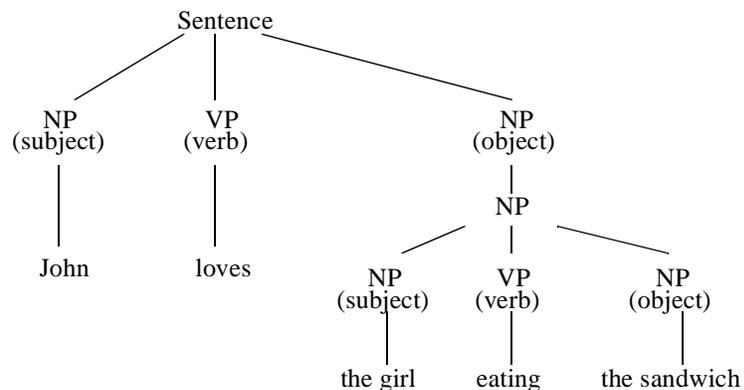


Fig. 5-3 Hierarchical decomposition of a symbol “Sentence”

5.4.2 Within-level and Between-level Relations

In a symbol system there is a *within-level* relationship between a symbol and the expression that it represents. That is to say the elements of the expression are themselves symbols which are potentially as rich and complex as the single symbol which represents them. By way of contrast, the relationship between a symbol encoded as a distributed set of microfeatures and an individual microfeature (which is a recurring theme in the neural network implementation of symbol systems) is a *between-level* relation, that is to say the microfeatures are not themselves symbols and do not possess complex structure. This is an often stressed point in the attacks made by practitioners of AI on the distributed representation models (again, Fodor & Pylyshyn, 1988, is the classic example of such an attack).

5.4.3 Symbols as Property Carriers and as Pointers

As a consequence of the properties so far defined, a symbol on its own must contain sufficient information to locate the expression for its meaning (in the terminology of the C++ language, the symbol would be acting as a *pointer* (Stroustrup, 1991)). The act of retrieving this information is referred to as a *distal access* in recognition of the fact that the needed data may be located far from the point where it is used (Newell, 1990).

In addition, the symbol must also be a conduit of the properties needed by any process which will manipulate it. How this is handled is system-dependent. The symbol itself could embody this data explicitly: part of the intrinsic form of the symbol could then carry the needed data.

Alternatively, the symbol could act as a pointer to the information just as it does for its own structural decomposition. In a well designed system we would expect this usage information to be easier to access (measured as a function of the resources and time required to obtain it) than the decomposition data. This follows since usage information is, by definition, more likely to be needed during manipulation of expressions containing the symbol than the decomposition of the symbol itself.

5.4.4 Accessing Properties: Two Approaches

Given a particular object, it is easy to think of several different processes we might want to carry out for which different properties of the object are relevant. For example, given *is_car(x)* we could conceive of two processes each of which is concerned with different aspects of *x* as a car. One process, *buy_petrol[]* might be concerned with the amount of petrol in any vehicle about which the system is concerned. For each object in the system, it needs to check if the amount of petrol it has is a relevant concept. The *uses_petrol()* property of *x* would be helpful here. Another process, *plan_route[]* might be more interested in the ability of *x* to transport someone from A to B in a certain time. The *uses_petrol()* property is less useful in this context.

How should this be handled? There is a dichotomy which exists in a lot of AI on this point. For some applications such as a production system, extra properties are made explicit:

```
is_car(x) => fast_transport(x)
```

so that knowledge that x is a car causes other properties to be made explicit as tokens in working memory. The process *plan_route*[] might then only be concerned with *fast_transport(x)* and not with *is_car(x)*.

The alternative method used in AI is to attach the properties to the symbols concerned, creating object-property pairs which are themselves symbols. In the linguistic example shown earlier, the symbol “John” was the subject of the sentence. The symbol for the subject could be manipulated without knowledge that it is John which is the bound value in this expression, but other types of process may require knowledge of the particular subject. The hybrid symbol <john-as-subject> is neither the symbol for John nor the symbol for subject. It is another symbol entirely, which carries properties allowing processes which act upon it to do so without recourse to other sources of property information about ‘John’s or about ‘subject’s.

Of course, the symbol <john-as-subject> can be decomposed into symbols for John and the generic <subject> symbol, but the goal sought by creating such a hybrid symbol is to reduce the frequency with which the decomposed symbols must be used directly.

Are the two approaches doing anything different or are they merely two different schemes which produce the same functionality but were developed in different subfields of AI? In terms of implementation, they are clearly *not* the same. The former commits resources to making properties of a symbol into explicit tokens in working memory, through some function akin to a look-up.

The predicates for the properties of an object have an existence that depends upon, but is separate from, that of their parent symbol. As a consequence, when the parent symbol disappears from working memory (i.e. is no longer valid) there must be a normalisation phase which removes the dependent property symbols, a task which consumes system resources. This is a common problem in non-monotonic logic in which predicates which are established as true by a certain set of data, are invalidated by later information.

The hybrid system does not suffer as much from the problem of truth maintenance. Necessary properties are encoded in the symbol itself, or at least are accessible directly from the symbol encoding, obviating the need for the look-up and extra step of making a property explicit as a separate relationship before being able to use it. This latter approach has the potential to be faster since the bottleneck of rendering explicit each property predicate and adding it to working memory is removed. When a symbol is invalidated and removed there are no other tokens in the form of explicit properties which need to be cleaned up afterwards. However, it commits the symbol encoding to make those properties available without the intervention of an explicit tokenisation step: thus when the creation of the symbol

occurs, its encoding must facilitate access to its properties. It can either instantiate them explicitly or act as a pointer to them, but in neither case does the property itself become a separate token in the working memory.

5.4.5 Complexity of Symbol Encoding

A further property of symbols is that the complexity of a symbol (in terms of the quantity of information needed to specify it uniquely) is not necessarily related to the size or complexity of the data hierarchy it represents. Thus one symbol of a certain 'size' might represent only a single other symbol or, say, a data tree of twenty levels with a branching factor of one hundred.

In the context of implementing a symbol system in a neural network, each symbol would be represented by a pattern of activity over a fixed number of neurons. In the discussion of Smolensky's tensor product encoding scheme, it was noted that handling variable length expressions is a task not well matched with the neural network approach. Thus, one extra constraint which it seems reasonable to impose is that all neurally-implemented symbols have the same 'size' which is to say that the number of neurons needed to represent any one symbol is independent of the symbol itself as well as independent of its contents.

This point reinforces one made earlier, that the symbol acts as a pointer to its contents rather than containing them explicitly. The pointer approach permits symbols to be of fixed size while representing an expression of potentially unbounded complexity and depth.

Also, note that this constraint was also a motivation for most of the reported work on symbol encoding presented in chapter three, including that by Hinton, Chalmers and Chrisman. In all three cases, the aim of the research was to investigate the effects of compressing knowledge trees of variable complexity and depth down into a single vector of fixed length. The vector acts much as a symbol: being a token that can be used in other expressions to represent the original symbol tree.

5.4.6 Noise Tolerance in Symbol Encoding and Processing

In a system which is subject to noise (either of input data or due to processing error) the physical encoding of symbols should contain sufficient redundancy to avoid the erroneous classification of one symbol as another. Since the neural medium is an inherently noisy environment to work in (not only is there noise due to the probabilistic nature of the firing of each neuron, but in many neural architectures the cross-talk between stored memories contributes to the error in every operation) then the processes themselves are subject to error. One further property of symbols and one that is a consequence of this uncertainty is that of *context independence*. In the simple knowledge tree example given above, one relation used as an example was:

human $\xrightarrow{\text{has_legs}}$ *two_legs*

In a generic symbol system (implemented in software, for example) the result of the transition could equally well have been just the symbol *two* and the meaning of two legs would have been clear.

$$\text{human} \xrightarrow{\text{has_legs}} \text{two}$$

But this clarity is derived from the context of the given transition *has_legs*. This is sufficient when no errors of translation occur. But in a neural network, despite design effort to minimise the chance of error, the resulting symbol after a transition is not guaranteed to correspond to the symbol learnt. For example:

$$\text{human} \xrightarrow{\text{has_legs}} \text{paris_city} \text{ (???)}$$

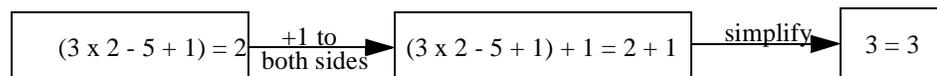
Extra context built into each symbol allows feedback to the control process to double check that the result is one of a set of possible correct results. In this respect, the control process has an ‘expectation’ of the form of the result and can take appropriate action if the result violates this expectation.

In receiving the result *paris_city*, the control process, which expected something with a context of *_legs*, knows that the retrieval process did not execute correctly. How it would handle such events is clearly implementation dependent, but at the least the control process has the option of going back to the previous state and trying again, perhaps allowing added noise to bring about an alternative choice.

5.4.7 Expression Length Limited by Finite Resources

Continuing the discussion of finite symbol size, the symbol approach is illustrated in figure 5-4 below, with a simple example from mathematics. In (a) no hierarchy is used whereas in (b) the bracketed portion is represented by the symbol *x*. The first process applied to both sides is independent of the value of *x* and so can be performed on the abstracted form in (b) yielding a result which is simpler to represent than that in (a). The second process, simplification requires the contents of the symbol *x* to be completed. The result obtained in (b) is thus only partial.

(a) No symbol hierarchy



(b) One level of symbol hierarchy

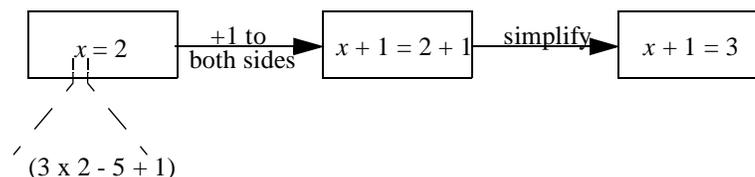


Fig. 5-4 Simple example of the use of symbols in mathematics

In a theoretical sense the length of an expression can be arbitrarily long. However, in the context of this architectural development there will be a finite quantity of resources available to represent any given expression in parallel. Thus, the abstraction provided in (b) will be needed to ensure that no expression requires more resources than are available. Conversely, an expression which uses less than the total available resources would be leaving some resources idle which is inefficient.

5.4.8 Structural Sensitivity by Processes

Turning to the processes manipulating expressions made up of symbols, one requirement demanded by the symbolic approach is that any transformation must be sensitive to the structure of the expression being transformed. Thus the format of the input data must be able to influence the way in which the algorithm treats it. The lack of such sensitivity was one criticism made of existing neural network candidates for a cognitive architecture (see chapters two and four, or Fodor & Pylyshyn, 1988).

To facilitate structural sensitivity, the symbol encoding, either directly or by way of its properties, must provide feedback to the control process to allow the actions performed upon it to be tailored to its structure.

5.4.9 Symbol Encoding Causally Related to Content

In the pure world of mathematical logic (extended to the realm of programming languages with some qualifications), the symbols that can be allocated to control and data tokens are arbitrary. Thus the symbol *two_legs* can refer to a creature with two legs, but it could equally well refer to the expression $y=2x + 7$ or to the concept of communism (Marx & Engels, 1848).

In the implementation of this symbol system, such arbitrary association of name and meaning is a luxury which cannot be afforded; instead we demand that *the encoding of a symbol be causally related to its designated syntactic behaviour*. Why is this necessary? It stems from the physical implementation of the symbol system in which the processes which conceptually transform the data symbols are realised as physical processes which receive only a subset of the whole symbol being processed.

For example, imagine an adder to be constructed as a simple 8-bit ripple adder in CMOS logic, which is intended to perform the following hexadecimal computation: $7 + 6 = D$.

The symbols have their usual, hexadecimal meaning. The adder requires binary coded information, so the symbols '7' and '6' must be thus coded. Each of the eight bits of the ripple adder is a 1-bit full adder and, thus, has three input bits (*a_in*, *b_in* and *carry_in*) and two output bits (*sum* and *carry_out*).

By applying the binary patterns for '6' and '7' to the adder, we observe that the output binary pattern (from the sum bits) corresponds to the hexadecimal number 'D' thereby exhibiting the required behaviour at the symbol level. But we know that even though conceptually the ripple adder summed two 8-bit numbers,

the underlying physical process was implemented as the causal action of single bits in the medium of transistors. No one transistor was affected by all of the bits of any operand and so there was never a place we could point to and say ‘the symbol transformation process was executed there’.

Returning to the neural network implementation of symbols, the argument proceeds along similar lines. The physical processes of the network (the neurons) receive a subset of the pattern for each symbol, and must compute the output of each operation based only on the information available locally, not on any global knowledge of what any symbol is in its entirety. Thus it would seem that the behaviour of a symbol in an expression or relation must be directly related to its structure.

Despite the preceding argument, it is possible to work with arbitrarily encoded symbols but the effect is somewhat illusory. The idea is to hide the causally-encoded symbols behind a mapping to and from non-causal (arbitrarily encoded) symbols. In this way we could construct a system which takes arbitrarily encoding symbols as input, processes them and produces arbitrarily encoded symbols as output. Only the central process would be constrained by the causality property; the translation from arbitrary encoding to causal encoding (and back again) would be done as a pre- (and post-) processing step.

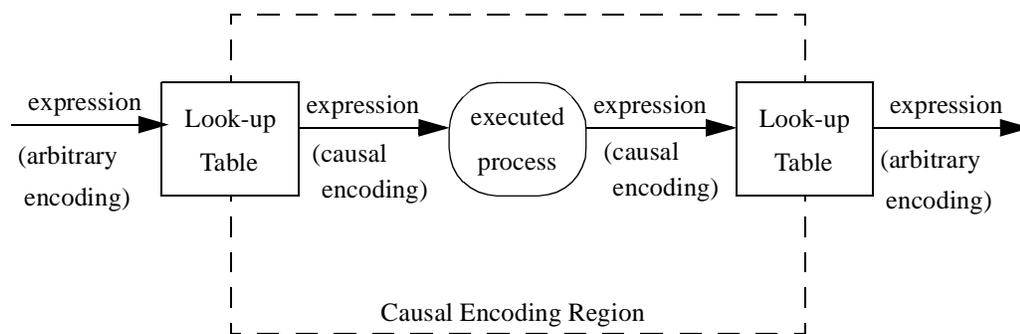


Fig. 5-5 Pre- and post-processing steps would permit arbitrary encoding

While being possible in principle, in practise the addition of a look-up operation both before and after execution of a process will, at best, slow down processing. How would the mapping from one set of encoding to the other be maintained? In a neural implementation, how would the tables of mappings be constructed and accessed?

There are no apparent gains in adding a level of arbitrary encoding, but the need to allow mappings between arbitrary symbols does manifest itself transiently during learning. Consider a symbol, S, whose encoding has been optimised so that it will operate correctly in whatever relationships have been specified for it. While learning a total new relationship for S there must be changes in the network structure. These changes must occur either in the mapping from a statically encoded symbol S to the target symbol, or in the encoding of S itself, or both.

If all changes occur outside of *S* then the network is essentially making mappings between arbitrarily coded symbols. This is a situation which we would like to avoid because, as just described, the arbitrarily encoded symbol *S* is a source of inefficiency to the system as the number of mappings it makes increases. A more efficient system would also modify *S* to facilitate the mapping. This scheme will be adopted in the work to be presented.

However, it is assumed that making changes to *S* to permit the new mapping without disrupting existing mappings will take time and so a transient condition will exist where the new relationship is facilitated by an arbitrary mapping, which over time will be replaced (made redundant) by subtle changes to *S*. This is expanded upon in the next sub-section.

As was noted in the review of reasoning systems in chapter four, the view that symbol encoding should be causally related to its meaning is not new. The work of Aleksander, for example, is also founded on the view of a neural reasoning system as a state machine. States of the network correspond to symbols (or *icons* in his nomenclature) and the transition from one symbol to another is a function of the bit pattern used to encode it. (Reviewed in section 4.4.5, page 113).

If there is one area in Aleksander's work on Magnus that appears to be missing (but is a central theme here) it is that there seems to be no *structure* in the state encodings of Magnus. By structure, we mean that the encoding used for any given symbol, although causally related to the effect that the symbol will have on future states of the system, could have been arbitrarily chosen since it appears to bear no relationship to the encoding chosen for any other symbol. It is the structure of symbol encodings that we argue will make inheritance and generalisation practical in large neuro-symbolic networks.

This statement concerning Magnus' internal structure may prove to be incorrect since details of its internal structure are not well publicised to date. It is hoped that the Magnus team have addressed this crucial issue themselves. If so it would be interesting to see how they have approached the problem.

5.4.10 Symbol as Both Source and Target of a Mapping

One property of symbols is implicit in all of the previous discussion: in the course of any process (which we take to be some form of mapping), a symbol is both a source and a target. As a target symbol, its encoding must be such that the process maps input symbols to that particular output symbol. As an input symbol, its encoding must be such the process will causally produce the correct output symbol.

In the previous section, the commitment was made to force a symbol's encoding to be causally related to its syntactic role in any expression. But consider a system in which new knowledge is added continually during normal operation. New relationships for symbol *X* arrive intermittently, as illustrated overleaf. First, we learn that symbol *X*, when operated on by process *C* must produce symbol *P*. Later, we learn that when operated on by process *D* it must produce symbol *Q*.

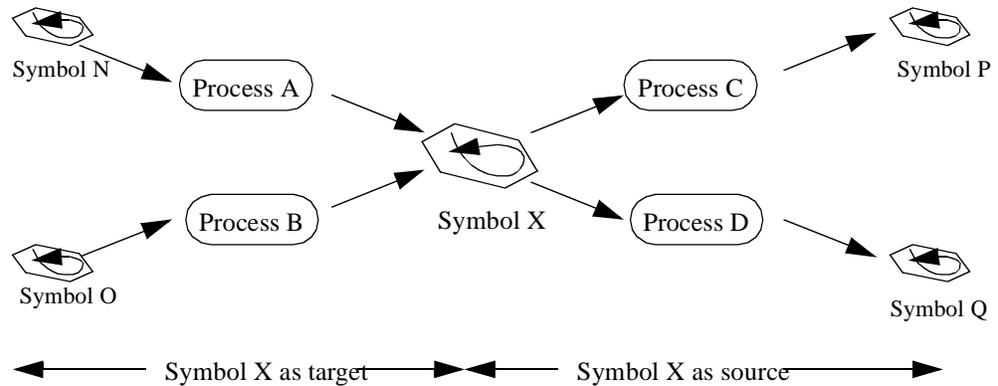


Fig. 5-6 A symbol, X, acts as both a source and a target in many processes

As each new relation cannot be anticipated, the symbol encoding assigned initially for symbol X may become increasingly sub-optimal for facilitating the appropriate transformations. Later in this chapter, the properties of neural networks will be used to try to alleviate this problem by permitting the symbol encoding itself to change over time to try to track the changing web of relationships in which it is involved.

5.4.11 Summary

This section has presented and discussed many of the properties that a symbol is expected to possess to function correctly in a Classic AI environment. While most properties are globally applicable to any symbol system there are a few, such as demanding that all symbols are represented using the same number of bits, which are more relevant to a neural network implementation.

The next section briefly considers some candidate symbol encoding scheme which were rejected. Following that, the focus fully onto issues of symbol encoding in a neural network which led to the final encoding scheme.

5.5 Rejected Candidate Symbol Encodings

At this point, a number of candidate encoding schemes for a symbol will be presented and then rejected. The reasons for each rejection will be discussed with a view to clarifying later architectural choices.

Note that in all of these candidate encoding schemes, we are trying to represent two different types of information. First, the contents of the symbol. These contents are the structure (made up of other symbols standing in certain relations to each other) that this symbol stands for. The other type of information being represented is the usage information of the symbol itself: those features of the symbol structure that are most often needed when using the symbol itself in other symbolic structures and computations.

The various encoding schemes in this section represent several ways to handle both these types of information. Two fundamental conduits for the data are to encode it explicitly in the symbol itself or to encode a pointer to the information in the symbol. To allow a symbol to be encoded with a fixed number of bits we cannot encode the contents of the symbol within the symbol itself. That option goes against the most basic philosophy of symbols in any case. Therefore at the very least a pointer must be used to access the contents of the symbol. Here we are more concerned with the options for the usage information.

5.5.1 Single Pointer, Encoded-Usage Scheme

In this first scheme, a symbol is made up of a fixed number of bits, N . The content of the symbol (i.e. its constituent structure) is accessed by an address which occupies a fixed portion of the symbol encoding. However, the usage information, those properties of the symbol which must be known by any process which will operate on it, are encoded explicitly in the symbol itself.

It is assumed that the address is encoded redundantly so that a predetermined number of bit errors can be corrected for. Thus, the probability of data corruption during processing (due to errors in the address) can be reduced by higher levels of redundancy, as required. The cost is extra bits in the address field.

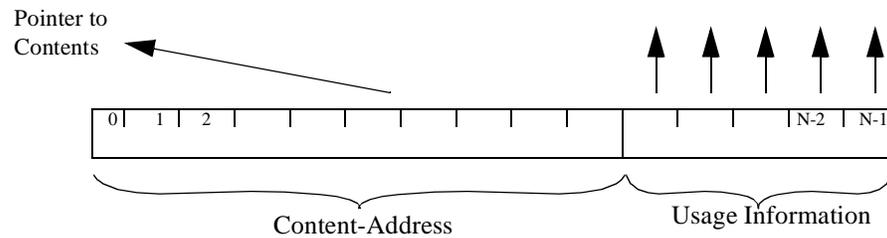


Fig. 5-7 The single pointer, encoded-usage scheme of symbol encoding.

The advantages of this system of encoding are that the contents (which are of indeterminate size and complexity) are only accessed when needed. This approach is in line with the basic philosophy of symbol systems. The usage information, however, is directly accessible by the process without a memory access. This permits the process to be accelerated in systems where memory access is a bottleneck to performance.

The disadvantage of this scheme is that it places artificial limits on the quantity of usage information that any symbol can display and use. When this limit has been reached for a particular symbol, how does the system cope with a new process which requires extra usage information not already coded there? If it abandons some of the existing data then other processes will no longer work. Thus, the use of direct coding in the symbol itself is rejected as a methodology for this architecture.

5.5.2 Double Pointer Scheme

In this scheme, a symbol is again given a fixed number of bits, N . These are divided into two sections: the longer first section is an address or pointer to the contents of the symbol. The shorter, second, section is an address to the usage information of the symbol.

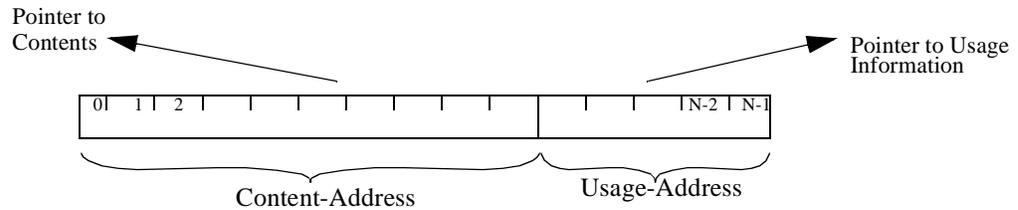


Fig. 5-8 Double pointer scheme allocates one pointer to each type of information

It is assumed that the quantity of content information is larger than that of usage information. Thus the address range necessary to indicate the location of the contents data will require more bits to specify.

As before, the indirection in the contents address allows structures of unlimited size and complexity to be represented by a single symbol. The usage information is similarly accessed, avoiding the problems, just described, of explicitly encoding such information.

There are a number of disadvantages still present in this scheme. The first is one of efficiency. There is a constant need to make memory accesses to retrieve usage information, which is potentially a bottleneck to performance.

Secondly, in the event that the location in memory of either the contents or usage information is changed (such as during memory management operations), then every instance of every symbol must be located and changed. This problem could be avoided by using a hash-table encoding for each symbol so that the two pointers are located in a table and accessed by its entry number. While pointer updates would need to be performed in only one place, every access to the symbol would require a look-up to locate its pointers followed by accesses to memory via the pointers themselves. It is assumed that this scheme is too slow for a very large network. In any case, it represents a serial bottleneck during computation. Though we might alleviate this problem by providing multiple copies of the pointer translation tables, such an approach would be counter to the philosophy of parallel, distributed processing.

Finally, such encodings do not seem to facilitate the inheritance of properties by virtue of membership of a category (generalisation), except by explicit modification of the usage information on each symbol. If, for example, we have a symbol which represents a member of a particular class, how do we facilitate the transfer of properties (as appropriate) from the parent class to the child symbol? We could, at the inception of each symbol copy parent usage information into its own usage data structure, but this is a costly operation and requires continual maintenance as parent classes are themselves updated or refined.

What the scheme lacks is a way of referencing the parent class in the symbol's usage information, so that the processes which act upon it know to access the parent category for usage information that must be inherited. Another way to view this is that inherited properties should be retrieved dynamically when needed rather than held statically for each child.

5.5.3 Treble Pointer with Inheritance

To overcome the third and final problem of the double pointer scheme, an obvious extension to consider is to allow a second usage pointer which allows access to parent (or class) usage information. Thus, a symbol A which is part of a class denoted by symbol B could inherit properties from it using a second field. Any updates to the usage information of the parent usage information would automatically affect all symbols which belong to that class since they are accessing their parent's usage information in addition to their own.

Processes that use symbol A would need to make two accesses: the first to the parent's usage data and the second to usage data specific to symbol A itself. As is normal in inheritance schemes, a conflict between the two types of information is resolved by accepting the symbol specific data in preference to the category data (Russell & Norvig, 1995).

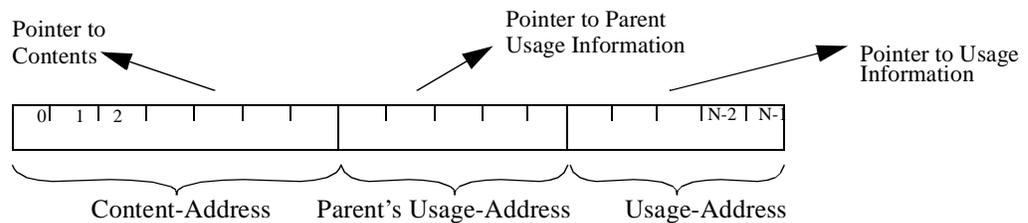


Fig. 5-9 The triple pointer scheme with inheritance.

Clearly two accesses would lead to a slower system than one requiring a single access per symbol. More serious is that such a scheme takes a very coarse view of inheritance. One might envision that an object is a member of many different classes to varying degrees and should be able to inherit properties from any and all of these classes.

To extend the triple pointer scheme to four or even more pointers is clearly possible, but does not follow one of the stated goals: the system should become more efficient as it is scaled up. Allowing increasing numbers of classes will slow down the system in proportion to the number of classes and consume resources in every symbol. Also, requiring a fixed number of classes per symbol is clearly an artificial limit. Some entities may belong to many classes, while others may belong to few. These would lead to an inefficient use of resources.

One of the major objectives in the neural network encoding of the symbol is to try to avoid these problems; to try to produce a subtle and flexible encoding which gives the system fine grain access to inherited properties without the cumbersome requirements of accessing exactly P pointers for P classes.

5.6 Development of a Robust and Flexible Symbol Encoding

The next few sections will outline the thought processes which led to the form of the neural network-based symbol encoding for the architecture. The discussion begins with the distinction between a mapping based on a function and one derived from a look-up table. This leads into a few points on generalisation.

The next sections address two issues of hierarchy in the system. First there is hierarchical knowledge structure which allows symbols to inherit properties from other symbols. Second, there is hierarchical learning which describes how the resources used to store symbols and relationships can be shared using higher order correlations between symbol encodings. This leads into the concept of different persistencies of memory traces.

Following the discussion on hierarchy is a section on learning issues, in which the evolution of symbol encoding to facilitate access and to optimise resource usage will be considered. Next, the operations which can be performed by the network will be described, followed by areas for future architectural exploration.

The last section of the development will reconsider the goals outlined in chapter one, in the light of the defined architecture. The stage will then be set for the network development which begins in chapter six.

5.7 Look-up versus Computation

It is appropriate to distinguish the two ends of a continuum which describes the manner in which data is retrieved by the system from its memory. These two extremes will be referred to as 'look-up' and 'computation'.

First consider the look-up principle. Given an input and a relation, one could construct a simple table which provides the output. Processing would consist of scanning the table to locate the entry corresponding to the two inputs and returning the associated output. A new entry can be quickly added to the end of the table without disturbing the other entries.

The advantage of this approach is that data items are not erased by subsequent additions to the table. As noted earlier in this chapter, this approach is also amenable to symbols which have arbitrary encoding.

However, there are several disadvantages to the simple look-up table. First, the larger the table, the slower the look-up process will become (assuming a fixed finite resource is available to perform the look-up). Second, a system must exist to maintain the table, to detect and correct inconsistencies between entries. Third, generalisation is difficult to implement.

Input Symbol	Relation	Output Symbol
car	has_wheels	four_wheels
car	uses_fuel	petrol_fuel
human	uses_fuel	food_fuel
millionaire	species	human

Fig. 5-10 A purely look-up table approach to processing.

The process of generalisation depends on the extraction of commonalities between inputs that allow the system to generate outputs for previously unseen cases by extrapolating from previous seen cases. In its purest form, a system based on look-up is incapable of interpolation and is, therefore, not applicable in situations in which generalisation is required. (Any form of interpolation is defined as computation in this context. For pure look-up, there is no systematic relationship either between a given pair of symbols or between any two lines in the table).

At the other extreme of information retrieval is the purely computational. Here, there is no storage of individual cases. Instead, the output of the retrieval process is calculated using some function.

$$Symbol_{out} = f(Symbol_{in}, Relation_{in})$$

where $Symbol_{out}$, $Symbol_{in}$ and $Relation_{in}$ are encoded symbols and the function $f()$ is a numeric function which performs the mapping. While in the strict mathematical definition of a function, the possible mappings for $f()$ could include the look-up function as a special case, we assume that $f()$ makes no reference to any previously seen input symbol. Instead, the function is an extraction of the underlying process and is applicable equally to previously seen cases or novel cases.

In the purely computational approach, generalisation is trivial since any symbol can be used as an argument of the function $f()$, producing the appropriate output symbol. But there are fundamental practical problems with this approach. First, the function $f()$ must be identified. Perfect identification of the mapping function would require knowledge of every possible case which, in real systems, is often not practical. Also, the complexity of the function itself would be a strong determiner of the quantity of resources necessary to implement it in a given technology. That complexity depends upon the encoding of the symbols as well as on the actual mapping itself.

Furthermore, the readiness with which generalisation can be achieved also depends upon the symbols encoding. In qualitative terms, a function for which

similar input symbols produce similar output symbols would be smoother than one for which relatively small changes in the input symbols had a large impact on the output. In many practical contexts (digital signal processing (DSP) applications or neural networks, for example) smooth mapping functions are easier to implement than those with many discontinuities (see, for example, Bishop, 1995).

From the definition of the extremes, it seems that both look-up and computation have advantages and disadvantages. In a symbol system is it necessary to choose between them? The answer is no. Indeed one of the advantages of a neural network implementation of a symbol system is that it facilitates the representation of information at either end of the lookup-computation continuum or even at some intermediate point. How this might be achieved is the subject of chapter ten.

5.7.1 The Learning Process as a March down the Continuum

Information arrives at the sensors of the system from the external environment. This information is discrete, encoded as symbols. These symbols represent events in the external world. In this chapter these events have taken the form of simple relations as in a simple database but in principle they could be more complex, representing the state of some world and the results of actions performed by the system in that world. In any case, they are discrete and arrive serially. Initially, the only option the system has is to store them as independent relations. Recall of those relations then acts as a look-up.

But over time, more and more relations appear and must be stored. The goal we seek is that given enough examples of certain relations, the system will learn to extract the regularities from the data, building its own model of the world and be able to generalise in the face of previous unseen data. To do this, it must internalise the examples it has and create a function which will perform the mapping in a range of contexts. The process of doing so transforms the encoding of the stored relations from simple look-up to a full computation in which individual examples no longer figure explicitly. We can think of this internalisation as a march down the continuum from one extreme towards the other.

It is worthy of note that traditionally the AI symbolic approach is best suited to the look-up form of computation and has performed much more poorly at the function-based task of generalisation, whereas the neural network approach makes a form of generalisation easily achievable (based on interpolation between points in the mappings for two cases which flank an unknown one) but is poorer at quickly storing unrelated relationships between the codes for two objects.

However, the current interest in hybrid AI-ANN schemes (as discussed in chapters two and four) is a consequence of the belief that a neural network implementing an appropriately coded symbol system could bridge that gap.

5.8 Hierarchies of Knowledge

It is assumed that the world has structure which is manifest in the appearance of statistical trends in the relationships defined for objects known to the system. These trends permit the formation of categories or classes which bring together objects which share common mappings. Using this technique, storage efficiency can be achieved by defining a mapping once at the class level instead of once for each member. The concept of inheritance is exploited in AI (Russell & Norvig, 1995) as well as in conventional programming languages such as C++ (Stroustrup, 1992).

This section begins by reviewing ideas on classes and inheritance, then proposes a form by which the usage pointer of a symbol could facilitate inheritance by acting as an incremental vector. Finally, there is a qualitative discussion on the manner in which a control process could exploit the usage information.

5.8.1 Classes & Inheritance

Consider one coarse grained example. The system is given mappings for several objects which share a common property. It could use the evidence of similarity between the results of these mappings to group them together, creating a category and allowing the members to inherit the property mapping by virtue of their membership to that category.

In this way, we might produce the following grouping:

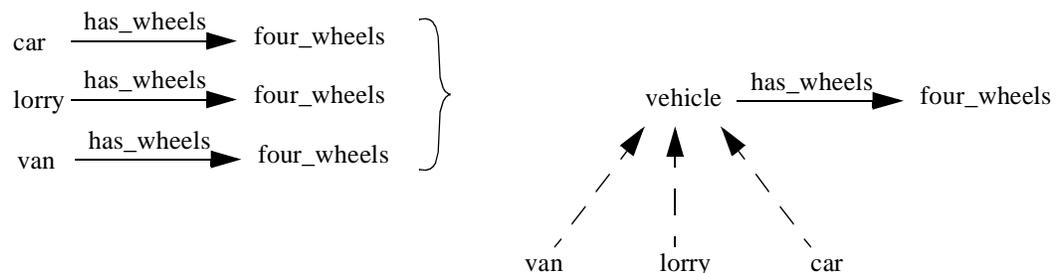


Fig. 5-11 Creation of a category whose members share a common property.

where the dotted arrows denote membership of the category. The new category 'vehicle' inherits the mapping on behalf of its three members. Queries concerning the number of wheels of a car would cause a reference to the parent category, accessing the mapping there.

As discussed in chapter four in the discussion of semantic networks, inheritance permits a form of generalisation, since an object may join a category by virtue of one set of properties, but may access another property by inheritance which had never been specified for that object directly. In this way, the object inherited the value of the mapping by virtue of its similarity to other objects. (See section 4.2, page 92).

Note that it is usually the case that individual members may override the default value supplied by the parent by keeping a local value for any given mapping. Thus we could define a new member, 'Robin Reliant' of the category vehicle which would inherit all properties of vehicle but may override some:

Robin Reliant $\xrightarrow{\text{has_wheels}}$ three_wheels

Thus, a member is not constrained to accept all of the properties of a category with their values unchanged. In addition, a member could belong to multiple classes, reflecting different aspects which might be of interest. Inheritance can thus be made through multiple channels.

Categories may also belong to other categories, creating a network of inheritance. How inheritance is handled is implementation dependent. The fact that it permits inductive inference complicates the situation since the system must be able to deal with cases when mappings it makes through inheritance are incorrect. There may be conflicts due to different mappings arriving from two or more branches of the inheritance hierarchy. Once again, the system must deal with this.

To implement inheritance in a system where the symbol encoding must be causally linked with its syntactic role, there must be a level of representation beneath that of whole symbols in which the membership of various categories is indicated. In earlier sections, this was referred to as usage information and, for reasons that were discussed at the time, the symbol encoding was described as acting as a pointer to that usage information (see section 5.4.3, page 124 and section 5.5, page 131).

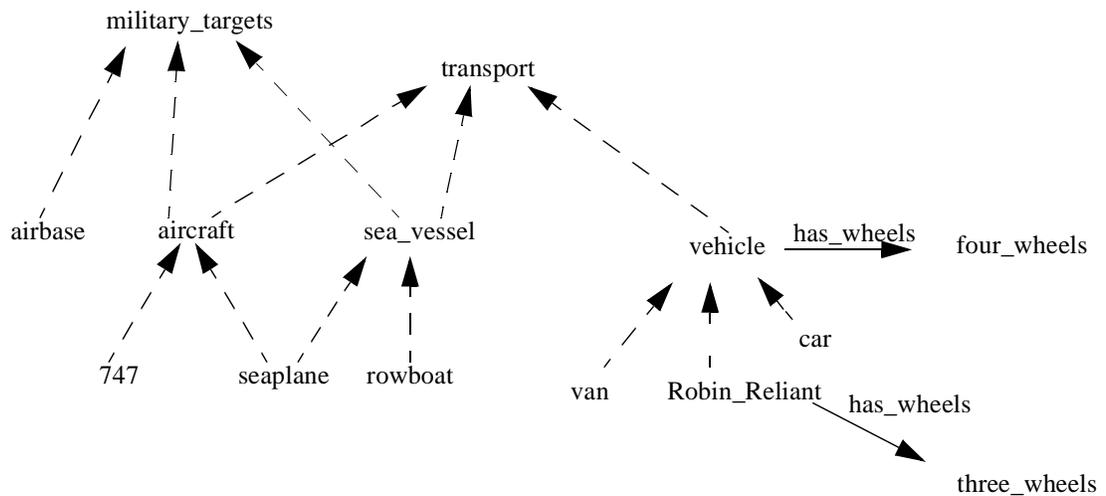


Fig. 5-12 Data structure with Inheritance.

5.8.2 Usage Information Encoded as Incremental Vectors

Individual symbols could have their own usage information, but it would be more efficient if objects shared as much data as possible. Consider a system with N levels of categorical hierarchy, with level N-1 as the highest level (in which we might define the broadest categories such as ‘animal’, ‘vegetable’ or ‘mineral’) and level 0 as the lowest, fine grained level at which each object instance resides.

The category at level 1 is still very fine grained and essentially an abstraction of a number of objects at level 0. The properties of a category at level 1 are in some way an average of properties of its members, while the members are permitted individual variations around that average.

Ascending the hierarchy, it is reasonable to assume that each level is more abstract than the level below. Thus the very highest levels constitute the central tendencies of very large and broad categories, with each level below them providing finer grained information as a delta on the properties inherited from above.

The further down the hierarchy one progresses, the more varied will be the objects and categories that one encounters, a result of many incremental refinements. How does this allow usage information to be encoded?

We expect the rough value of the pointer (its direction in vector arithmetic or its address in a computing model) to be specified by the highest level of the hierarchy. Each level below it contributes a delta, so that the pointer reached at level 0 is unique for each object. For a system in which all knowledge has been well assimilated, the adjustments for each level of the hierarchy will be smaller and smaller as we descend.

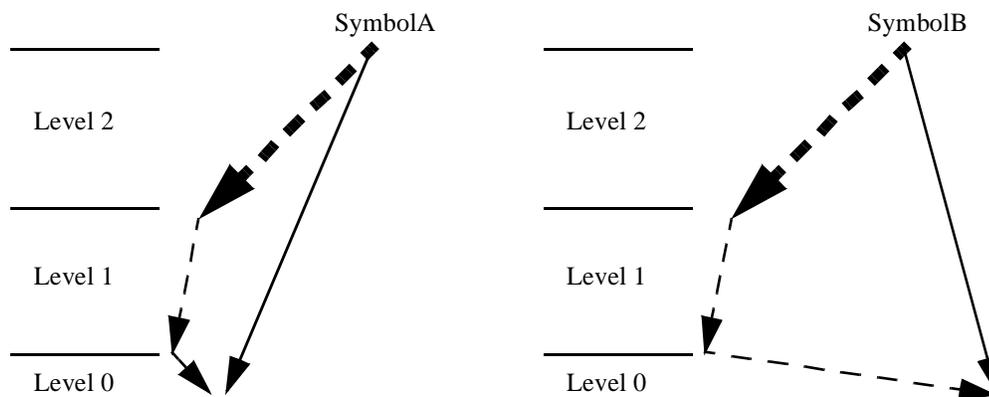


Fig. 5-13 Usage pointer for two symbols.

In the figure above, two symbols are shown. For SymbolA, the level 2 pointer (shared by all members of the highest category) dominates the overall usage pointer (the solid vector). Each level contributes to the building of the pointer, but in a well balanced system the contributions of each lower level is less than the one before.

Now consider SymbolB, which breaks the rule by having a large correction factor at the lowest level, level 0. Here the symbol has recently been involved in new associations. Properties which had been assumed for SymbolB by virtue of category membership, were shown by example to be incorrect and so the system records this by making a modification to the usage pointer which is specific to the symbol itself (i.e. at level 0).

It is possible that the newly learned relationships are truly specific only to SymbolB, in which case they should not propagate to the categories in which SymbolB is a member. However, the new data might be more generally applicable than SymbolB alone.

How and if the new information is propagated up the hierarchy is not a straightforward question to answer. Even if we assume that any particular relationship which is supplied to the system is true, we are making an inductive leap to promote it as a property of a higher category. A complex system may be able to use deductive inference based on other information.

Alternatively, it could use a trial an error approach, based on making the assumption that the particular situation either is (or is not) generalisable to the level above and then using test cases to assess the validity of the assumption.

Whatever the induction method used, the system may be forced to reorganise, adjusting the categories to which SymbolB belongs and adjusting the average value of the pointers in the levels above to capture the central tendencies of the new class membership. This begins by adjustments in the membership of level 1 and so on up the tree. The higher levels must be much slower to change since they represent the central tendency of a large number of objects. At the lowest levels the categories contain only a few highly specific instances and can change much more rapidly without globally affecting the database.

To facilitate this, we must ensure that two criteria are met: first, that the degree with which the connections between symbols respond to new learning events must decrease as one ascends the hierarchy. Thus the lowest levels change most rapidly and the highest levels change most slowly.

Second, that localised changes in a given level must be capable of overriding (masking) the effects of the levels above. Thus, when a new relationship for SymbolB arrives, the local changes to the mappings for SymbolB can immediately alter the usage pointer for SymbolB (overriding the central tendencies of the levels above) but does not affect other symbols which have the same category membership as SymbolB. This property should exist for all levels, so a change to a category at level K should initially affect only its members until such time as its effects can be generalised to level K+1. How this concept of hierarchy is realised is implementation dependent, but clearly the symbol encoding must incorporate it at a fundamental level.

5.8.3 How The Usage Pointer Would Affect Computation

The purpose of the usage pointer encoded in the symbol is to supply sufficient information to the executing process to ensure that its contents are manipulated without necessitating a direct access. Thus, in the implementation we would imagine that the vector representing the usage information must interact directly with the vector or vectors of the control processes to affect the transformations carried out.

The complete usage pointer is the symbol itself and, in the perfect case, would be unique to that symbol (i.e. no two symbols are identical). By saying that the executing process is sensitive to the constituent structure of the symbol and that the usage pointer represents that structure in all relevant ways, we are committing the executing process to 'decide' how it will treat the symbol (and hence its constituents) based solely on information available in the usage pointer: the symbol encoding itself.

Since it would be impractical to define a specific way of treating each and every symbol in the context of each and every process, then there must *necessarily* be structure in the processes and the usage pointers. Thus, the usage pointer of each symbol has a structure which allows common features in usage pointer encoding to trigger common responses from the executing process. The hierarchical nature of the usage pointer, as described in this section, is one means of achieving this goal.

Similarly, the executing process would interact with the symbol by way of a hierarchically structured 'execution pointer'. The structure of this pointer would be developed in such a way as to produce the required transformations of the input symbols. Symbol and process would develop together during the maturation of the data organisation. New symbols would *necessarily* need to conform to the developing principles of organisation in order to be understood. How they might do that is discussed in a later section.

5.9 Hierarchies of Learning

The review of issues from neuroscience described how the human memory system appears to have many sub-systems, each dedicated to retention of a different type of information. Also, the complexity of the biochemistry underlying each synapse has led researchers to believe that the simple division of memory as either long or short term may be too rough a categorisation and that more complex mechanisms of learning may be involved which permit many degrees of persistence.

In this section, some of the consequences of possessing a richer arsenal of synaptic behaviour will be considered. It begins by considering the benefits of more complex association between symbols and continues by considering what properties must be possessed by the underlying units of representation (in this case artificial neurons) to manifest the desired benefits.

For a memory system, the coarsest distinction that can be made is between long and short term memory. Permanent relationships between symbols make up long-term storage (LTM) in this system. Similarly, short term memory (STM) would include variable bindings and relationships which were recently learned. The latter use of STM would theoretically permit the system to learn a relationship and, upon discovering that it was incorrect, to remove it again without damage to LTM. However, this is not the way that STM is traditionally used. In most models (e.g. Carpenter & Grossberg, 1994) STM refers only to a reverberation in the activity of the network and does not result in synaptic modification.

The issues highlighted in this section are closely linked with what has been referred to as the *stability-plasticity dilemma* in neural network theory (Grossberg, 1988). At its core, the dilemma is the need to trade-off, on the one hand, the flexibility of a system when faced with a changing environment and, on the other, the persistence of pre-established feature detectors or relations whose presence is significant when viewed over a longer time scale than the short-term changes. In a system of limited resources there is always a trade-off to be made and this section proposes a novel approach to the problem using memory persistencies.

Consider the figure overleaf, illustrating two binary vectors. The vectors represent symbols that are to be associated. Whenever SymbolA appears, it should trigger the recall of SymbolB in a separate network.

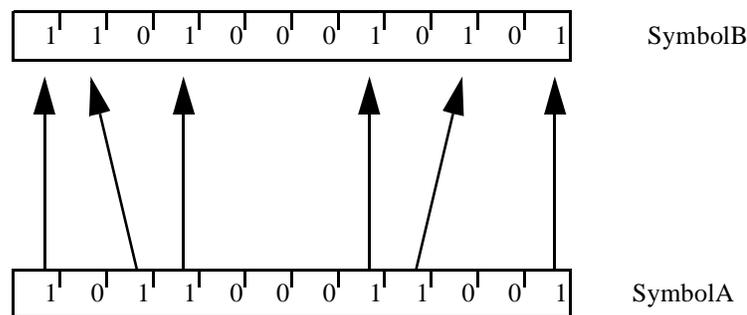


Fig. 5-14 Direct association of two symbols.

Learning this association could be achieved by direct connections between the significant features of SymbolA and those of SymbolB. As more such associations are made, it may be possible to combine the activity by extracting features from the mapping and creating one or more hidden layers of detectors to perform this function; this is done in many neural network architectures today (the MLP of Rumelhart, Hinton & Williams, 1986b, will be used as the example here). Each level of feature detectors is capable of extracting a higher level of association

between the bits of the symbol vectors. Such a structure is illustrated in figure 5-15, below.

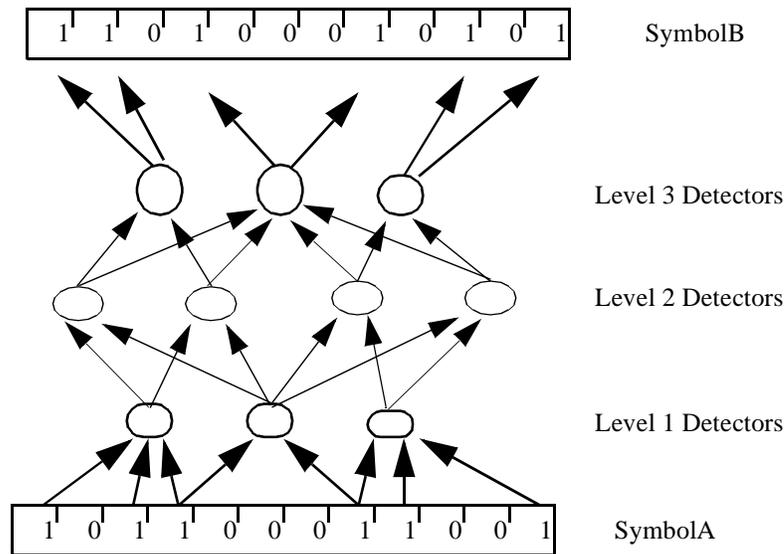


Fig. 5-15 Indirect association of symbols using feature extraction

Traditionally, however, there have been a number of problems with the use of intervening levels of feature detectors. Firstly, it usually takes many repetitions of the full set of patterns to learn the mapping. This is due to the necessarily slow evolution of the weight vectors of the feature detectors themselves.

Second, most known learning processes (such as error backpropagation as used in most MLP applications) are not guaranteed to converge on a set of feature detectors which will optimally represent the statistics of the data. Finally, it is difficult to differentiate between associations which are temporary and those which are intended to be permanent.

What behaviour would we like to achieve? First, that temporary symbols would be quick to create. Next, that temporary associations would be quick to make. Third, that permification (or consolidation) of symbols or associations would be possible, either on demand based on external cues or due to a high frequency of usage. Finally we would like one particular learning event to have minimal impact upon existing memories (particularly memories which are semantically unrelated).

While the presence of high-level feature detectors allows important statistical relationships to be extracted, it is also the source of slow learning in associations networks (the MLP being a good example). To try to achieve the best of both, the model to be adopted is one which permits features extracted at any level to be used at any higher level. Such a structure is shown in figure 5-16.

In general, the units which form the feature detectors may be the same units that represent the symbols themselves or may be separate. In either case, new associations can be made easily by permitting direct, but temporary, connections between the active units of the input symbol and those of the output symbol.

While the connections remain in place, the symbol association exists. If unused then within a short time they will decay and vanish, breaking the association. (Clearly, the definition of 'short' would depend on the learning environment and system needs).

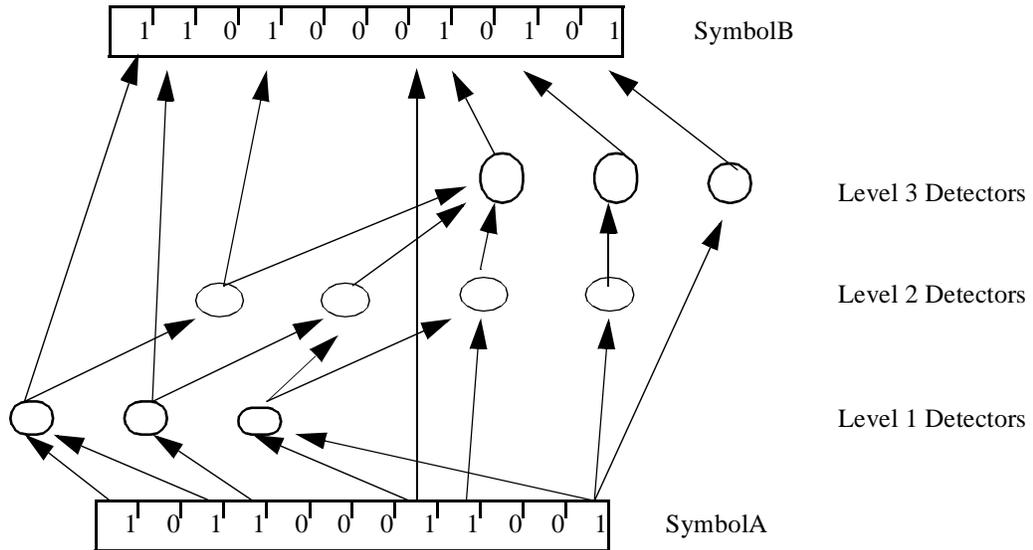


Fig. 5-16 Learning hierarchies allow mixtures of features of different levels

If the association is to be made permanent (due to frequency of usage or an external cue) the level 1 feature detectors must be modified based on statistical correlations in the input bits. The choice of which detectors to modify is based on which require the minimum change in their weight vector to represent the new correlation.

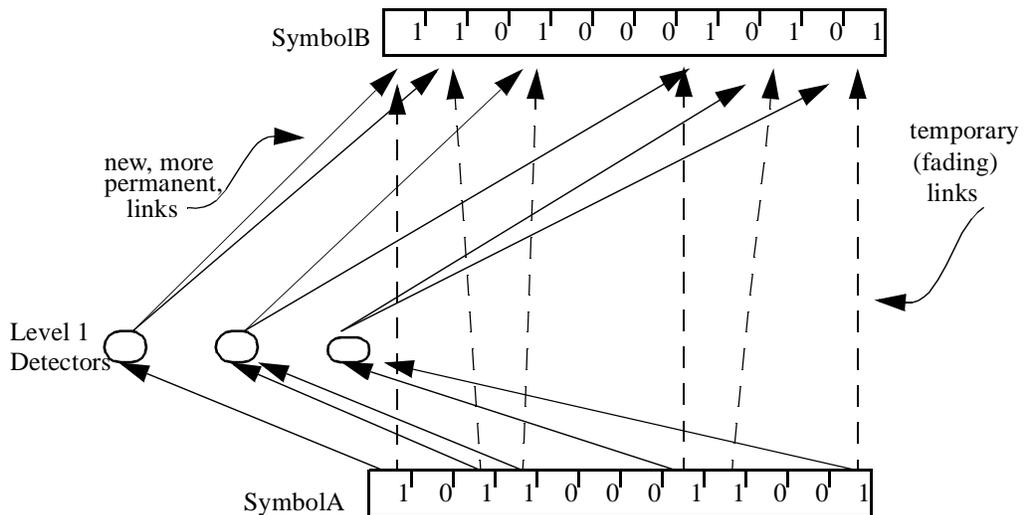


Fig. 5-17 Adjustment of feature detectors that supersede direct (temporary) links

Thus, as illustrated above, the level 1 detectors make slight modifications in order to represent symbol A. The direct links (dotted) fade quickly but their effect is picked up by more permanent links from the level 1 detectors.

This is repeated at each level of detectors. Each time, links from one level to the output symbol are replaced by more permanent links from the level above. This is accompanied by a slight change in the feature detectors at the higher level to make them sensitive to the pattern at the lower level.

The underlying assumption is that the temporary links have great impact but only for a short time and in response to specific stimuli. In contrast, the more permanent links are made by increasingly subtler changes to the synapses, but which affect the response to a wider range of input stimuli.

The goals in this approach are twofold. First, to accommodate both fast learning and optimised storage, two apparently opposing requirements. Second, to allow the significance of a feature to be reflected in its permanence in the network.

While the ideas presented in this section are similar in approach to the description of classes and inheritance in the last section, to first order they concern two different concepts. It is interesting to consider how the similarities might be combined, but the details are implementation dependent and will be left until later in the network development.

5.9.1 Different Durations of Activation

An extension to the learning hierarchy principle allows the duration of activation of a connection to depend on its position in the hierarchy. Thus, a direct connection from unit A to unit B would be active as soon as unit A is active and would cease as soon as unit A is deactivated. For higher level correlations, however, when unit A is deactivated, the connection continues to provide activity to unit B for a certain duration which depends upon the correlation level of the connection.

The idea behind such a scheme is that higher-order features, once detected, should be able to have an effect on the updating of the network over a longer period than a low-order feature or direct connection.

5.10 Symbol As Representative for a Constituent Structure

As discussed earlier in the basic properties of any symbol, we know that each symbol is a point of access to its constituent structure. That structure is made up of symbols which stand in certain relations to each other, and each symbol is potentially as complex as its parent in terms of its own constituent structure.

In this implementation, we place the additional constraint that every symbol must be encoded using the same number of representational units so that a network

of finite resources can be used handle any symbol without regard to the complexity of its constituent structure.

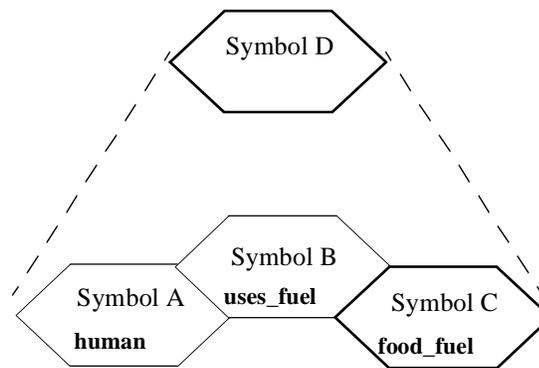


Fig. 5-18 Every symbol can embody multiple symbols of equal complexity.

In the architecture, this means that a set of symbol in some relation, each with their own encoding, must be somehow compressed down to a single symbol with its own encoding. This new encoding must include not only a pointer to allow a process to access the original structure but also usage information which is relevant to the process concerned.

Clearly, there must be a loss of information between that represented in the whole expression at one level and the symbol use to represent it at the level above. Thus, the process, P, which creates the parent symbol (D in the figure) must be able to select elements of the usage information of the represented expression which a relevant to the next stage of processing. Thus, the abstraction process is not executed in isolation but is guided by the control network to fulfil context specific requirements.

5.11 Unsupervised Development of Symbol Encoding

In a simple network such as an MLP, the patterns which must be associated are supplied by some external agent. In this system however, the network must provide its own encoding for individual symbols for a number of reasons which have been argued throughout this chapter. Principally, the fact that the effect of a symbol must be *causally* related to its encoding means that the encoding cannot be arbitrary.

However, if some degree of redundancy is used in defining and interpreting the encoding of a symbol then there are multiple codes which refer to the same symbol and which would be interchangeable in the course of normal execution without changing the result of pre-defined computation (i.e. operations for which the symbolic results of executing a process on a symbol have been specified by an external source). This level of redundancy permits the network to render a symbol unique by randomly moving a small fraction of the bits each time a new relationship involving it is learned. If the degree of disruption is below the intrinsic level

of redundancy, all such modified symbols will exhibit normal behaviour in existing relationships. This is illustrated in figure 5-19, below.

Furthermore, each unique symbol may form relationships of its own. The shifted bits are indistinguishable from the original bits in these new learning events. Furthermore, if context information is coded into a parallel network and associated with the new mapping, then each learning event causes the symbols (with their modified, unique bit patterns) to be associated with context information which is unique to that event.

One way of looking at the process of rendering a symbol unique is that the original symbol essentially form a category for its children. Alternatively, one could view the original symbol as an overgeneralisation of the real relationships involved, which is divided into sub-categories by learning new examples in differing contexts which may produce different results.

In either case, the result is that symbols which form many relationships will exist in many slightly different forms. If a parallel mechanism were implemented (at a sub-symbolic level) to cause two frequently used symbols with similar encodings to become dissimilar over time, then the encodings of the originally similar instantiations of the symbol would diverge. To balance this, there would need to be a convergence mechanism to allow irrelevant divergence to be avoided, perhaps based on the infrequency of use of one of the encodings.

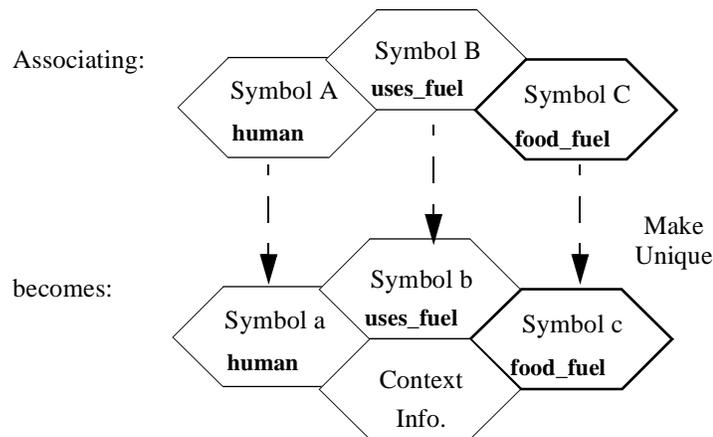


Fig. 5-19 Each new association links modified versions of core symbols

The overall behaviour that is sought in this approach is that the encoding for a symbol is not imposed from the outside of the system but is instead driven by the interdependence of the mappings from one symbol to another. The use of noise in each new mapping allows the network to diversify its symbol encoding, to react to new relationships among existing symbols and to pave the way for inductive inference as the next stage of architectural development.

5.11.1 Relation to Ho-Kashyap Encoding

In the review of Casasent & Telfer's work (section 3.1.2, page 48) the comparison of several possible learning algorithms for associative memories showed that a modified version of the Ho-Kashyap learning procedure provided the best storage capacity among those that were considered. While $2N$ patterns was stated as the capacity of a network with N neurons, the demands of the algorithm were to modify the output patterns themselves during an iterative learning procedure.

For this network, the architecture design demands that the resulting symbols be modified to facilitate their function. It is interesting to compare the requirements and results of the Ho-Kashyap algorithm with the procedures we must define for the symbol encoding.

First, we note that for Ho-Kashyap, it is necessary to compute the pseudo inverse of the input pattern matrix as the first step of the algorithm. This is, itself, a compute intensive and non-local computation, which should discourage its use in a system design that aims for ease of realisation through localised computation.

Next, we note that Ho-Kashyap requires us to pre-compute all of the output patterns before the network is in use. There is no iterative procedure that can tweak the encoding 'on-the-fly' when the system is in use. This means that we must know all of the patterns that must be stored in advance, something which is not possible in a dynamic, learning environment.

In terms of philosophy, we note that Ho-Kashyap seeks only to make the forward mapping from each of a set of input patterns to its corresponding output pattern. But here the encoding, even after modification, must maintain all of its existing relationships both as input and output (i.e. as both a source and a target symbol, as described in section 5.4.10, page 130). Therefore, we need an algorithm that is both simpler and more local than Ho-Kashyap, one that can act in an adaptive fashion in a learning environment and one which gives us control over the degradation in the performance of existing relationships.

Fortunately, we do not necessarily need an algorithm that makes optimal changes to a symbol in a single pass. It may be acceptable to make the changes over multiple cycles or multiple re-uses of the symbol provided that the symbol is not re-used too frequently before there has been time to modify it appropriately. This approach takes us more towards a more standard, iterative, learning procedure such as back-propagation.

5.12 Choice of Symbol Encoding

From the critique of symbolic computation presented in chapters two and four together with the development of ideas presented in this chapter, it was decided that the encoding of a symbol should be made up of a single pointer rather than an allocation of bit fields to multiple pointers. This one pointer must be capable of addressing the constituent structure of the symbol and to provide access to

the usage information required by processes executed on it. In addition, the pointer must be self-sustaining since the value of a pointer (i.e. the encoded vector) must be the target of mappings which lead to its symbol from others.

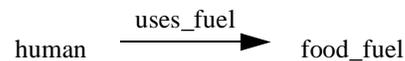
[Note that if we consider the N-bit encoded symbol as a vector in N-dimensional space, the use of fields to separate multiple pieces of data is equivalent to treating the N-space as a set of non-overlapping sub-spaces. The symbol vector is then seen as having a component in each sub-space. Operations which act on individual fields of the encoding are analogous to vector operations that act within the separate sub-spaces. With the single pointer approach, we treat the whole vector as existing in a single space. The benefit of this is that there are no artificial limits on the boundaries between sub-spaces and the representation can develop such that each “field” consumes as many “resources” (in this case available dimensions in N-space) as are needed. Since the resources are limited, the set of fields compete for them as the encoding develops but note that this is a dynamic process which continues for the lifetime of the symbol.]

Returning to the choice of symbol encoding, we note that because the system will be noisy, coupled with the requirement that each symbol encoding must be able to causally affect computation, we must accept two new constraints. First that symbols with similar mappings should have similar encoding and second, that the encoding should have structure of its own to permit generalisation by the executing processes.

The implementation phase must quantify these concepts in the context of the neural network medium and the symbol level operations the network is required to execute.

5.13 Detailed Architecture Structure

This section describes in more detail the structure of the architecture and the main operations which are available to it. The core of the architecture is the ability to store and access relationships of the form:



which will be abbreviated to:



In all cases the symbols themselves are coded using the same quantity of system resources so that the same units which represent symbolA are equally capable of representing symbolC, and vice versa.

The architecture, illustrated in the figure overleaf, is built around a single large memory store which hold all of the data for the symbols and their relation-

ships. The output of this memory can sustain the pattern for only a single symbol. In addition, there is a small working memory which is used only as a short term storage of symbol encodings. The output of the working memory is also capable of representing a single symbol. Symbols stored in the small memory are used only to reactivate the main memory. None of the pointer or relationship information is stored there.

The rationale behind this scheme is that there is only one place in the symbol representation hardware where the data structures of symbols and their constituents is located. The working memory serves only as location for the second symbol in a pair which will operate on the symbol which is active in the main memory. (An alternative scheme in which separate and complete networks exist for each of the three symbols in a relationship makes it difficult to manage the linking of a symbol in any of the three roles with its other links. Thus it was avoided).

Symbols which are stored temporarily in working memory are tagged by the control process so that it can access them by tag rather than by content. This permits the control process to place symbols there and reactivate them using the tag at appropriate points in the algorithm.

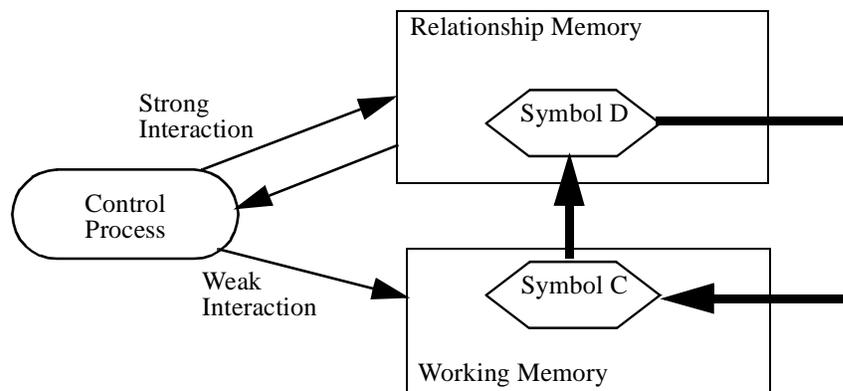
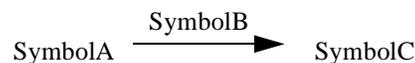


Fig. 5-20 A more detailed schematic for the symbol processing architecture.

5.13.1 Learn New Relation

As a fundamental operation, the system must be capable of learning a new relationship between existing symbols. Associations are of the form:



and are executed in two passes. In the first, SymbolA is set up in the main memory and SymbolB in the working memory. An operation is performed to combine the two into a single symbol, still in the main memory, which embodies the concept <SymbolA-SymbolB>.

During the second pass SymbolC, which is in the working memory, operates on the hybrid symbol from the first pass, creating a single SymbolD which embodies the entire relationship. As was described in the last section, at each step a number of the bits in each symbol are corrupted by noise to ensure uniqueness in the relationship. Also, external context information may form a part of the learned mapping, but such a scheme will not be considered further in this development.

5.13.2 Encapsulation of an Expression as a New Symbol

The creation of hierarchy, a fundamental requirement for the symbolic architecture, is a consequence of the mechanism described above for the learning of a new relationship. Any number of symbols can be amalgamated in this way, although only two at a time. Thus all tree structures would have a branching factor of two.

As is shown in figure 5-21, three symbols, A, B and C which form a relationship can be merged to form a single symbol, D which can represent them at the next level of the hierarchy.

Symbol D must possess certain properties which have been enumerated throughout this chapter. The most important are that (1) it must act as a pointer to usage information which is appropriate to the set of symbols that it represents; (2) it must be capable of being accessed to retrieve the original set of symbols and (3) it must be representable using the same quantity of resources needed to represent *one* of its constituent symbols. This third requirement permits symbol D to take part in further computation using the same set of units that were used to represent its constituent structure in an earlier timestep.

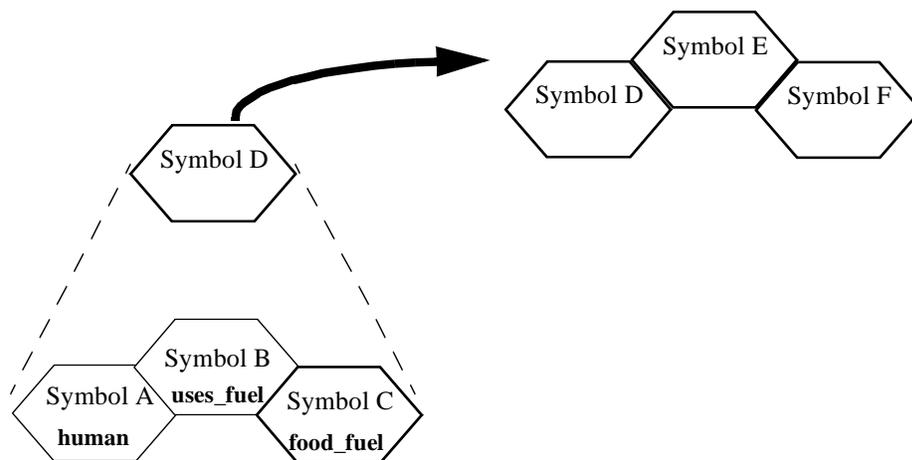


Fig. 5-21 Symbol D created to represent relationship between A, B and C

The creation of symbol D is an action instigated by an external cue, such as a decision by the control network. From arguments presented earlier in this chapter, the encoding for D cannot be arbitrary; instead it must causally represent the relationship from which it was created.

Once it has been created in main memory, one of two things can happen to Symbol D. Either it can be operated on immediately, perhaps using symbols in working memory as other operands of the executing process. Alternatively, it could move to working memory itself and become an operand for a future operation which acts on another symbol. The control process (another network) makes this choice, which may be sensitive to the encoding of Symbol D itself.

5.13.3 Access Contents of a Symbol

As a reverse operation to the creation of a single symbol from a set of symbols in a relation, it must be possible to ‘unpack’ the symbol, accessing its contents and retrieving any of the constituent parts.

Using the same example, Symbol D would be active in the main memory and the intention is to retrieve one of the three symbols which make up its constituent structure. To be able to do this, the control process must have knowledge of the basic roles to which the three symbols are bound. The usage information in the encoding of symbol D must provide this information, guiding the control process in its choice of the stimulus it must present to the main memory to cause Symbol D to transition to the required constituent symbol, A, B or C.

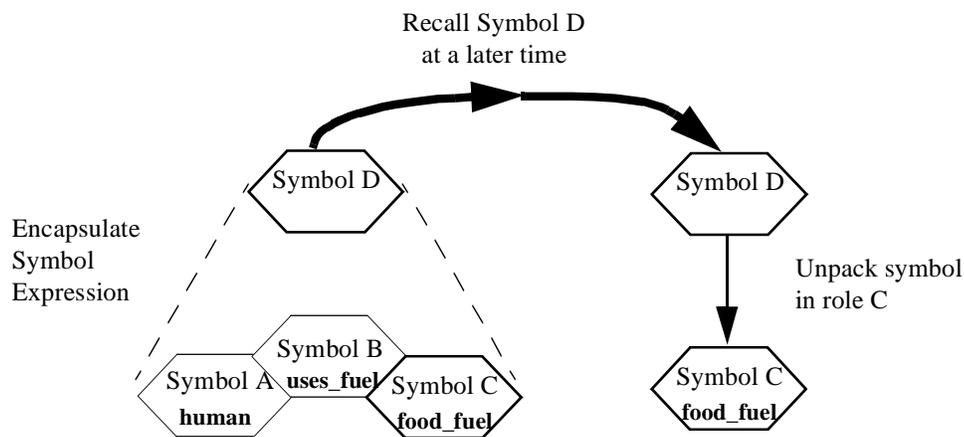


Fig. 5-22 Unpacking a symbol to access content information

5.13.4 Sub-symbolic Operations

A number of operations are assumed by the architecture which act at a sub-symbolic level. They all concern the re-organisation of symbol encoding, for various reasons. First, the optimization of encoding to improve storage capacity. Second, the permification of data or relationships which were originally stored only temporarily. Thirdly, for inductive learning, the promotion of properties to apply to their parent categories requires changes in the encoding of the categories themselves.

Based on arguments of practicality derived from critical examination of neural and classical AI architectures as presented in this thesis, it is asserted that the optimisation of the symbol encoding *should not be handled at the symbolic level*. Furthermore, it is asserted that the clarification of the interface between these two

levels lies at the heart of the problem of creating practical intelligent systems. The key questions which remain to be resolved concern the exact nature of the symbol optimisation and the consequences that changing symbol encoding will have on the performance of the system itself.

5.13.5 Summary

This section has described the data processing parts of the architecture at the symbol level. The operations that it must execute were described, but the mechanisms by which each operation would be achieved are not readily explainable at the symbol level. The implementation must fill in the missing details, providing a mechanistic account of data processing at the sub-symbolic level with is consistent with the operations defined at the symbolic level.

5.14 Conclusions

This chapter has presented the development of the knowledge architecture. There were two major topics of discussion. First, the properties of a symbol system were described, both in a general sense and in the context of a neural network implementation. The encoding scheme of the symbols themselves was described for a learning system with facility for hierarchical knowledge and inheritance as well as a variety of memory persistencies.

Second, the scope of the knowledge base to be implemented and the operations which could be performed upon it were outlined. Specifically, the network will permit the association of objects through a mapping which is defined by a third symbol. Recall of either operand and/or the operator is facilitated through a simple look-up. Symbol structure is implemented by transforming the ensemble of symbols into a single symbol which can be represented using the set of units as one of its constituents. This parent symbol carries usage information for its constituents and can itself be accessed to regain the internal structure, as required for any system based on symbolic computation.

The architectural level will return to the focus of attention in the penultimate chapter as an act of unification: looking ahead at the hurdles that remain to consolidate the symbol and neural levels and realise true neuro-symbolic systems. This activity is largely one of analysing remaining symbol level issues. It brings into the discussion elements that emerged in the review of reasoning systems presented in chapter four. In this chapter these elements have not been the focus of attention, as more fundamental issues of symbol encoding have dominated. This makes sense since the next chapter takes the issues of symbol encoding down to the neural level.

In the next chapter then, we turn to the neural aspects. The chapter is largely concerned with the encoding of patterns. Subsequent chapters will begin analysing the simplest network based upon the chosen encoding and then add extra features with the goal of providing a plausible building block for the architecture described here.

Foundations for the Neural Building Block

6.0 Laying the Foundations

The previous chapter discussed the high level architecture of the network. By analysing the necessary properties that the system would need to possess and the functions it would need to perform, the outline of a candidate architecture was constructed. Little attention was paid to the actual properties of the neural substrate on which the machine will be implemented, but along with the architectural definition came a set of constraints on the encoding of individual symbols within the network which must be taken into consideration during the implementation phase.

This chapter starts at the other end of the design spectrum from the knowledge level. Here attention is focused on the neural building block which will form the repeatable unit of the implementation level. A cursory inspection of the number of possible neural network candidates showed that the range is considerable. The surveys in chapters two and three represent only a subset of the possible network architectures. It was not assumed at the outset that the “best” neural architecture with respect to the stated goals would be a single existing architecture such as an MLP. Instead, the design process was permitted to allow hybrid schemes or even fundamentally new constructs, where appropriate.

This chapter begins with some general choices for the network substrate. The bulk of the discussion thereafter will consider the coding scheme that was used to represent all data items during processing. Next, a first attempt to define the structure of neural building block will be described. It will be left until the next chapter to define the basic learning procedure and to analyse the properties of this simple network. Further chapters will then present embellishments to the basic structure, to produce a network which can implement the full architecture.

One final note is required to put this chapter in context with the rest of the work. Chronologically speaking, what is presented in this chapter is work which occurred before the architectural development. It is placed with the other chapters on network development mainly for the sake of presentation. Thus, some of the discussion re-examines assumptions that were made during the architectural development, such as the distributed code.

6.1 Recurrent Network

At the architectural level it is clear that the fundamental network functionality is the transformation and re-transformation of data patterns. A simple feed-forward MLP type network is not sufficient: some form of recurrent network is needed. In principle, an MLP could be used if coupled to a non-neural mechanism for recirculating the output back to the input. (Indeed, in software terms there is no difference between this scheme and a fully recurrent network). However, it would be desirable to use a single homogeneous substrate for the network, using as little ‘non-neural circuitry’ as possible to implement the system. The rationale was that a homogeneous design should render the system easier to realise, keeping in mind the goal of being practically implementable.

In fact, the choice of a recurrent network does not eliminate much of the field. Most neural networks are either inherently recurrent or can be made so by using some of their outputs as extra inputs. Thus, while discounting purely feedforward networks, the rest of the field remains intact.

6.2 Replicable Neural “Unit”

Since the original conception of the network saw of the network was that it would consist of a large number of neurons, it seemed a reasonable simplification to demand that the network be made up of a large number of identical ‘units’ each containing many neurons, as is the case in the human cortex; Practically speaking, adopting a homogeneous network structure like this has many advantages. The principle benefits are reviewed below.

First, we can study the properties of a single unit (in a simplified environment representing neighbouring units) and then extrapolate this to the whole network. This would be easier to do than to consider the whole network simultaneously. Secondly, from the point of view of credit assignment during learning, each unit can be considered independently and share the task equally.

Third, in the implementation of any architecture (for example, an integrated circuit) the existence of a replicable unit greatly reduces design time and construction costs. Finally, a replicable unit makes it easier to build fault tolerance into the network.

But what form should this unit cell take? To preserve generality as much as possible, only two constraints were placed on the neural unit. First, that it must consist of a block of N neurons that have high intraconnectivity (i.e. with their peers in the same block). Second, that the connectivity outside of the block is limited to a certain maximum radius, r . How the neurons are actually connected and what equations govern their operation was left unspecified at that time. The intention of these two constraints was to simplify the analysis by modularising the whole system as well as to render it more practical to realise by keeping connectivity local.

As a side note at this point, one major objective that was envisaged was that the regions are all identical in more than just their basic structure. It was demanded that all regions should always *aim to* represent the same quantity of information as their fellow units. Clearly under transient conditions of learning the actual amount of information held and transmitted by any one unit might vary from unit to unit and this quantity might change over time.

Defining a property such as the equality of data storage was intended to allow each neural unit to make adjustments to its data storage based on locally available information, but from which would emerge a global consistency.

Note that this approach rules out networks such as ART and its variants, in which extra neurons are recruited to represent new patterns that are too different from template patterns stored by existing neurons (Grossberg, 1976a). The reason for eliminating such networks is that in an implementation of such a network the final population of neurons would need to exist from the system's inception and be activated as needed. Not only does this imply a waste of available resources when the network learns its first few patterns but it also imposes a hard upper limit on the number of separate classes that can be represented. Both of these restrictions violate key constraints of this work.

6.3 The Data Coding Scheme

The next choice to be made was the coding scheme for the data vectors themselves. In the architectural development one of the themes which kept recurring was the critical role played by the choice of data representation. This choice could not be arbitrary if the goal was a system which was both robust and efficient. This section looks at candidate schemes, beginning with a look at the characteristics of a single neuron.

If each neuron, N_i , has potential U_i and a threshold value, T_i , then its output is given by:

$$\begin{aligned} O_i &= 1 \quad \text{when } U_i \geq T_i \\ &= 0 \quad \text{otherwise.} \end{aligned}$$

The choice of binary neurons does limit the expressive power of the network, compared to a network of neurons with graded or sigmoidal response. However, it will be shown (later in this chapter and in a later chapter on learning) that this step makes the analysis much easier than one with a more complex output function.

The next step was to decide how the data will be represented on these binary neurons. The detailed review of chapter three revealed that the literature contains a wealth of existing work to draw on in this area. Overall, three different coding styles tended to recur in many of the associative memories that were covered.

The simplest was the 1-from-N coding, leading to the grandmother cell approach to feature representation. This is often referred to as 'one-hot' encoding in

integrated circuit design and is frequently used to encode control signals (e.g. Weste & Eshraghian, 1992).

The second was the so-called *standard coding* in which a set of N neurons can represent any of the 2^N binary patterns available to it. The third scheme is the K-from-N code in which a fixed number, K, of neurons fire (output = '1') and the rest are silent (output = '0').

Here are some examples of these three codes for a 4-bit vector:

0001	1100	0000
0010	0011	0100
0100	1001	1010
1000	0110	0111
	1010	0001
	0101	1111
1-from-N Vectors	K-from-N Vectors (K=2)	Unconstrained Binary Vectors

The literature survey discussed the use of the three types of coding in an associative memory from the point of view of achievable storage capacity, general properties of the codes were often overlooked. For the sake of completeness and to aid further discussion, several of the key properties of each coding scheme will next be examined and compared.

6.3.1 Range of Expression

Range of expression, $V(N)$, refers to the number of legal vectors that can be represented with an N-bit vector using each coding scheme. This is an issue, since almost all schemes impose limits on what constitutes a legal vector and we wish to avoid selecting a scheme which permits only a small number of legal vectors. Clearly, the unconstrained binary vector has the best range of expression since it places no limits on each binary digit. For the 1-from-N code, only a single '1' appears in each vector. Thus, only N different legal codes can be generated, making the 1-from-N code a local representation scheme.

Finally, the K-from-N code. The number of legal vectors is now a function both of the length of the vector, N, and the number of allowed '1's, K. Calculating the number of legal K-from-N vectors, $V(K,N)$, consists of finding the number of ways of picking K identical objects from a set of N. This is given by the standard formula from combinatorics. The number of legal vectors for the three schemes are thus:

$$\begin{aligned}
 V(N) &= 2^N \quad \text{for the unconstrained binary vector} \\
 &= N \quad \text{for the 1-from-N code} \\
 V(K, N) &= {}_N C_K = \frac{N!}{K!(N-K)!} \quad \text{for the K-from-N code}
 \end{aligned}$$

Clearly, the range of expression for a 1-from-N code is narrow compared to the unconstrained binary code. While 1-from-N encoding is useful for simplifying the control logic of integrated circuits, its expressive power all but rules it out for large neural network design. However, its worth pointing out that it is often used for networks which use data clustering algorithms and competitive learning techniques. (See section 2.3.2, “Unsupervised Learning”, page 30). It has the added advantage of being easily interpretable by an external observer.

The table below compares the three schemes for various values of N. For the K-from-N scheme, the number of ‘1’s in the vector was taken to be 20% of the total number of bits.

Table 6-0 Comparison of three data coding schemes

No. of bits, N	No. of ‘1’s (K-from-N only)	No. of Legal Vectors in 1-from-N Scheme	No. of Legal Vectors in K-from-N Scheme	No. of Legal Vectors in Unconstrained Binary Scheme
10	2	10	45	1024
20	4	20	4845	1.05×10^6
50	10	50	1.03×10^{10}	1.13×10^{15}
100	20	100	5.36×10^{20}	1.27×10^{30}
200	40	200	2.05×10^{42}	1.61×10^{60}
500	100	500	2.04×10^{107}	3.27×10^{150}

Even for small values of N, the limits of the 1-from-N code are clearly visible. For the K-from-N code the numbers are below those for the unconstrained case, but grow exponentially with N.

It should be noted that the number given here are limits on the number of different legal vectors which can be generated in N bits. It says nothing about the memory capacity of the network itself.

6.3.2 Information Content

Here, we consider two quantities related to the quantity of information that can be stored in an N-bit vector using each of the candidate schemes: the maximum amount of information we can store in N bits and the efficiency of the encoding, as the information stored per bit, given by:

$$I(N) = - \sum_{x \in \chi} p(x) \cdot \log p(x)$$

$$E_{max}(N) = \frac{I_{max}(N)}{N} \text{ bits/bit}$$

where x is summed over the whole set of legal vectors, χ . In all of the discussions to follow, the base of the logarithm will be two and hence the unit of information will be the bit. Regardless of the coding scheme used, maximum information content is achieved when the probabilities for each legal vector are equal (Cover & Thomas, 1991).

It is straightforward to calculate these quantities for the three coding schemes. First, for the unconstrained binary vector, the maximum entropy occurs when the probability that any bit will be '1' is 0.5. In this case, the entropy and efficiency are given by:

$$I_{max-unconstrained}(N) = N \text{ bits}$$

$$E_{max-unconstrained}(N) = 1 \text{ bits/bit}$$

For the 1-from-N vector encoding, we find that there is always one bit which is '1' while all others are '0'. Therefore, there can be only N legal vectors, each of which will appear with probability 1/N:

$$\begin{aligned} I_{max-1fromN}(N) &= -N \cdot \frac{1}{N} \log \frac{1}{N} \\ &= \log N \text{ bits} \end{aligned}$$

$$E_{max-1fromN}(N) = \frac{\log N}{N} \text{ bits/bit}$$

Finally, in the K-from-N scheme, the number of legal vectors is equal to the number of ways of selecting K items from N. Thus:

$$\begin{aligned} I_{max-KfromN}(K, N) &= -\sum p(x) \cdot \log p(x) \\ &= -\log \frac{(N-K)! K!}{N!} \text{ bits} \\ E_{max-KfromN}(K, N) &= -\frac{1}{N} \cdot \log \frac{(N-K)! K!}{N!} \text{ bits/bit} \end{aligned}$$

The table below compares the three schemes in terms of the efficiency of their maximum information content for different vector length, N. For the K-from-N scheme a default value is used for K, equal to 20% of the value of N.

Table 6-1 Efficiency of information storage for different values of N

		Efficiency in Information Storage		
No. of bits, N	No. of '1's, K (K-from-N code only)	1-from-N Vector, (bits/bit)	K-from-N Vector, (bits/bit)	Unconstrained Binary Vector, (bits/bit)
10	2	0.332	0.549	1.000
20	4	0.216	0.612	1.000
50	10	0.113	0.665	1.000
100	20	0.066	0.689	1.000
200	40	0.038	0.703	1.000
500	100	0.018	0.711	1.000

From the table, we can see that the 1-from-N code is a poor means of storing information, particularly for large N. The K-from-N code, however, becomes more efficient with increasing N for a given K/N ratio.

In the literature review, many of the authors ignored this aspect of the information storage capacity of their networks, concentrating instead on the total number of patterns that could be stored. Nadal & Toulouse did consider information content, also taking into account the reduction in total information per pattern as a result of errors during recall (see section 3.1.2, page 48). This aspect will be considered in more detail later in this chapter.

6.3.3 Robustness

In the discussion of the Kohonen feature map (section 2.3.3, page 31), it was pointed out that one source of problems during the learning phase was the physical network topology which could conflict with the ‘representational’ topology as the network developed led to ‘twisting’ of the resulting net and hence poor representation of the underlying data.

The reason why this inherent topology was present was to allow the network to produce reasonable results in the face of unfamiliar or noisy stimulus: if the stimulus did not activate the correct node, at least it should activate one of the nodes physically adjacent to the correct one. Physical adjacency was used to identify a ‘near-miss’. Without this constraint, the 1-from-N code employed in the network would otherwise produce either a correct answer or a totally incorrect answer (with no hope of even detecting the error, let alone correcting it). In this case, a single bit error reduces the signal to noise ratio to zero.

In a K-from-N scheme, the presence of an error in one of its K ‘1’s modifies, but does not eradicate, the result. Consider the 5-from-10 code in the figure below. A network error resulting in a single bit error produces a vector which has a Hamming distance of two with respect to the correct vector. The signal to noise ratio for a single bit error is $(K-1)/1$. Considering implementations of the K-from-N code with different values for K, we see that as K approaches $0.5N$ the relative effect of one erroneous tends to a minimum value.

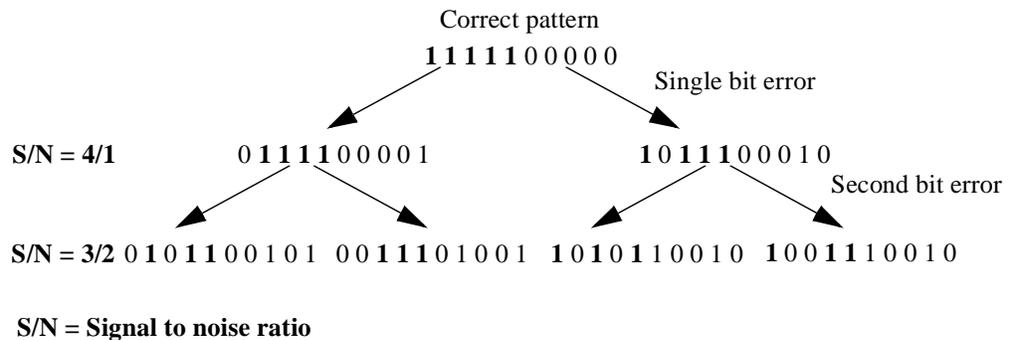


Fig. 6-0 Example of the K-from-N code illustrating its robustness

Notice that optimising the value of K for robustness is usually at odds with optimising it for maximum storage capacity. As discussed in the review of Nadal & Toulouse’s analysis of associative memories, the Willshaw memory has been shown to have optimal storage efficiency when the number of ‘1’s in the pattern $K = \ln N$, a figure that is usually far less than the $K = 0.5N$ value that leads to maximum robustness. Thus there is a trade-off to be made between storage efficiency

and robustness and the choice made for K clearly depends on the particular storage and noise requirements of the environment. If the noise tolerance and the target storage level are over-constrained for a particular value of N the only recourse would be to increase the number of neurons.

6.3.4 Easier to Control Interactions Between Regions

Each region may have to interact with many others. In such situations there will be ample opportunity for positive feedback which is the hallmark of instability. Notice that the length of each vector is given by \sqrt{n} , where n is the number of '1's. For both the 1-from- N scheme and the K -from- N scheme this vector length is always fixed (with values 1 and K , respectively). Changes in each vector affect only its direction in space.

For the unconstrained binary vector, however, the vector length can vary with changes in the bits in the vector, since the number of '1's can vary between 0 and N . In a system with positive feedback, this added degree of freedom would make systems based on an unconstrained binary representation scheme harder to control both dynamically and during learning. These ideas will be pursued in several later chapters.

We note that in the literature survey of chapter three there is a dearth of work into the use of K -from- N coding in recurrent networks such as the Hopfield or bidirectional associative memory (section 3.2, page 60). This is a puzzling omission given the above discussion on the extra degree of control that could be exercised over a fully connected network using such an approach. For purely feedforward, associative memories, such as those considered by Casasent & Telfer, and Nadal & Toulouse, the use of K -from- N coding has proved to be a powerful strategy and it seems interesting to consider what benefits such a strategy could bring to the auto-associative memories for which the storage capacity seems to be comparatively low.

6.3.5 Equality of Information Contribution

Looking at each region as the transmitter of information and those regions to which it projects as receivers, we can consider the K -from- N code in a channel (in the information-theoretic sense). In these terms, the channel in each direction is one of constant power, \sqrt{K} , as given by the constant length of the data vector.

There are several possible sources of channel noise. Principally: (1) For channels between system inputs and a neural region there is noise on the input signals, due to sampling as well as fundamental noise in the stimulus being sampled. (2) Interference from other stored memories. Since the value of this contribution does not change on the time scale of single updates of the network, this interference is often referred to as 'slow noise' (Amit, 1989).

(3) Between regions, there will be noise in the data vectors due to the non-zero probability of misclassification. (4) For both input and inter-region channels, there may be a noise component added deliberately as a part of the network's functionality.

6.3.6 Choice of Coding Scheme: K-from-N

After carefully weighing up the properties of the candidate schemes, the choice of coding scheme within a neural region will be the K-from-N code. It seems to share the properties of good information content and range of expression with the unconstrained binary code as well as the predictability and controllability properties of the 1-from-N code. As such, it represents a good engineering compromise. Further analysis of the properties of the K-from-N code will go hand-in-hand with the rest of the network development.

6.3.7 How To Decide the Value of K?

By selecting the K-from-N coding scheme for the neural unit, a new parameter has entered into the problem: K, the number of neurons out of the N which must fire at any time. How does one decide this value?

As was mentioned earlier, setting $K=0.5N$ allows the maximum number of different patterns to be represented. Is that the best value to choose? Not necessarily: there are more criteria to consider than the information content of individual vectors. The absolute number of stored vectors is one such consideration, as is the noise tolerance during recall and subsequent use of the output vector. We might also consider the requirements of storing complex data structures, something which may also impose restrictions on the vector encoding. The value of K will play a role in quantifying all of these concepts. After analysis of the memory capacity of the network in the next chapter, these ideas will be pursued in more detail.

6.4 Properties of K-from-N Codes

This section presents further details on the properties of the selected encoding scheme: K-from-N codes. We have already established in the last section that the number of N-bit vectors made up of K '1's and N-K '0's, $V(K, N)$ is given by:

$$V(K, N) = {}^N C_K = \frac{N!}{K!(N-K)!}$$

Two further aspects are investigated here. First, the variation of the information stored with K and with N. Second, the effect of building redundancy into the K-from-N encoding.

6.4.1 Information Content

During the process that led to the selection of K-from-N as the choice for coding, the information content of vectors using the K-from-N scheme was stated as:

$$I_{max} = -\log \frac{(N-K)! K!}{N!} \text{ bits}$$

It is interesting to see how the density of information representation (i.e. the maximum quantity of information that can be represented per bit of the vector) I_{\max}/N , changes as a function of both N and K . The graph below illustrates this, by plotting this information density against the proportion of firing neurons, $p = K/N$, for a number of different vector dimensionalities, N .

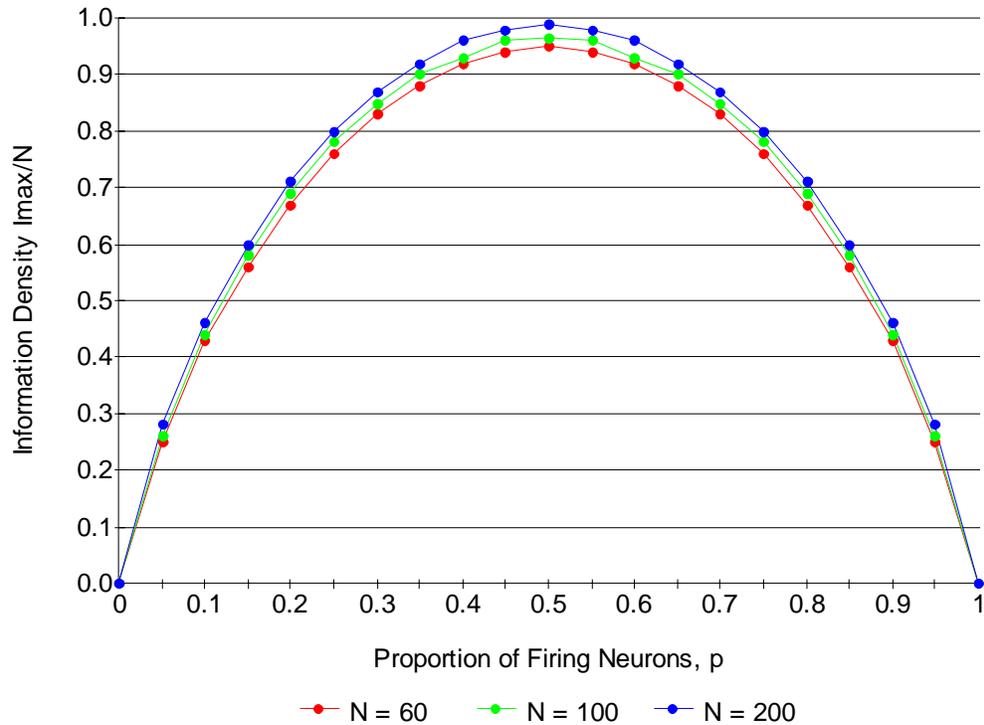


Fig. 6-1 Graphs of Information Density vs. Fraction of Firing Neurons

Note that as was mentioned in the review of the literature on associative memories, the density of information stored in an N -bit vector using K -from- N coding is a strong function of K . This is clear from the graph. The result that the maximum information content is achieved using $K=0.5N$ is well known, as is the fact that the total information content tends to zero as K approaches either zero or N .

What is perhaps less obvious, but clear from the plot is that there is also a dependency of the information density on N , so that for a given value of p the information density I_{\max}/N tends to its maximum value only as N tends to infinity.

6.4.2 Building Redundancy into K -from- N Code

From the results presented so far, it would seem that for even medium vector lengths, N , (in the tens or hundred of bits) the number of legal vectors expressible with a K -from- N code is extremely large: growing exponentially with N . However, as mentioned earlier, for a robust system there must be allowance for the fact that errors will creep into the output vector at each step due to noise in the input data, errors in classification, cross-talk between stored memories (so-called slow noise) as well as noise deliberately added as part of the networks normal function.

A sensible approach to such problems is to build redundancy into the code to permit error correction of faulty bits. We select a subset of the legal vectors and call them the ‘golden’ set. Vectors which are similar but not identical to a golden vector (as determined by some metric) are assumed to have errors. These errors can be identified and corrected if they are not too great.

Formally, we define a set of vectors, ϕ , which are a subset of the full range of legal vectors, ν . Around each vector, $\bar{\phi}_i$, we define a neighbourhood of legal vectors, η_i , none of which belongs to ϕ . For a particular $\bar{\phi}_i$, the neighbourhood is defined as the set of legal vectors which satisfy the distance metric, d :

$$\eta_i \equiv \{ \bar{x} \in \nu : (d(\bar{x}, \bar{\phi}_i) \leq M) \wedge (x \notin \phi) \}$$

$$\text{where } d(\bar{x}, \bar{\phi}_i) \equiv \frac{1}{2} \sum_{j=0}^{N-1} [\bar{x}^j (1 - \bar{\phi}_i^j) + \bar{\phi}_i^j (1 - \bar{x}^j)]$$

The summation j is over the bits of the vector. Here, \bar{x}^j refers to the j th bit of the vector \bar{x} . The distance metric $d(\bar{x}, \bar{\phi})$ is nothing more than a count of the minimum number of ‘1’s that must be moved to transform one vector into the other.

Note: that the distance d is twice the Hamming distance between any two K -from- N vectors. Also, the metric d has the properties required of any metric; it is always zero or positive, the self-distance $d(\bar{\phi}_i, \bar{\phi}_i)$ is zero, it is transitive (so that $d(\bar{\phi}_i, \bar{\phi}_j) = d(\bar{\phi}_j, \bar{\phi}_i)$ for any $\bar{\phi}_i, \bar{\phi}_j$) and it fulfils the triangular inequality.

Since every legal vector has constant length, \sqrt{K} , we can visualise each golden vector $\bar{\phi}_i$ as a projection to a point on the surface of an N -dimensional hypersphere of radius \sqrt{K} , with its neighbourhood of radius M as an area on the surface on the sphere. This is illustrated in figure 6-2, below.

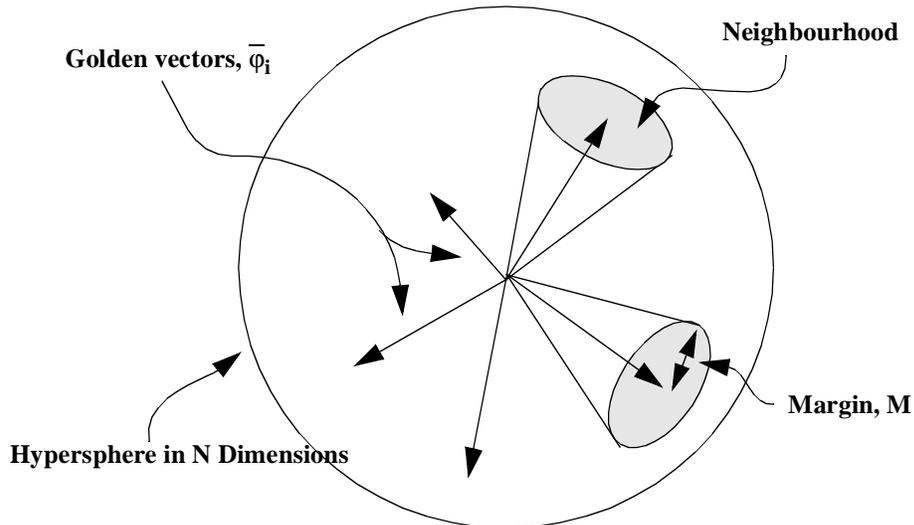


Fig. 6-2 Legal K -from- N vectors with redundancy.

From the definition of the neighbourhood, any vector which is a distance $d \leq M$ from a golden vector will be classified as that vector. The original selection of the golden vectors must take account of the fact that no two golden vectors may be separated by a distance less than $2M+1$ ¹. If this rule is ignored, the neighbourhood regions could overlap and it would no longer be possible to distinguish between some pairs of valid vectors.

To clarify matters, here is an example vector for $\bar{\phi}_i$ for a 4-from-10 code. Below is such a vector and examples of neighbourhood vectors at different distances from it. These few examples are only a small subset of the possible vectors at either distance.

•Distance = 0

$$\bar{\phi}_i = 1111000000$$

•Distance = 1

$$\bar{x}_0 = 0111000001, \quad \bar{x}_1 = 1011001000$$

$$\bar{x}_1 = 1101010000, \quad \bar{x}_2 = 1110000010$$

•Distance = 2

$$\bar{x}_0 = 0101000101, \quad \bar{x}_1 = 0011000011$$

$$\bar{x}_2 = 1001100001, \quad \bar{x}_3 = 1100010010$$

Notice that the vectors at distance = 1 are only a single ‘shift’ from the original vector, $\bar{\phi}_i$. At distance = 2 the vectors are two ‘shifts’ from the original, etc.

The next issue to consider is the size of each neighbourhood. A large neighbourhood keeps the golden vectors well apart and permits a greater number of errors to be corrected than a small neighbourhood. For example, with a margin $M=3$ we permit the vector to have three erroneous ‘1’s and still be able to correctly identify the correct (golden) vector. The drawback of a large neighbourhood, however, is that the number of possible golden vectors is reduced as the neighbourhood size increases.

The next step is to derive a formula which quantifies the size of the neighbourhood and hence to calculate the number of possible golden vectors for a given degree of error correction.

The number of golden vectors, $G(M, K, N)$, is a function of the vector length N , the number of ‘1’s in the vector K , and the number of erroneous ‘1’s that can be corrected M . Specifically:

$$G(M, K, N) = \frac{V(K, N)}{Z(M, K, N)}$$

1. The value $2M$ comes from the widths of two neighbourhoods. The ‘plus 1’ takes account of the golden vectors themselves. For example, for a margin of 3, each golden vector must be at least 7 bit shifts from any other.

where $V(K, N)$ is the number of legal K -from- N vectors, defined earlier, and $Z(M, K, N)$ is the number of such vectors in a neighbourhood of width M . Thus, to calculate G for any given M, K and N we need a formula for Z .

In fact, the derivation of Z is straightforward. It can be shown that the number of legal vectors at a distance, M , from any golden vector can be expressed as:

$$Z(M, p, N) = Z(M-1, p, N) \times \frac{1}{M^2} [N^2 p(1-p) - (M-1)N + (M-1)^2]$$

where $p \equiv \frac{K}{N}$ is the proportion of 1s in the vector, and $Z(0, p, N) \equiv 1$

The derivation of this formula has been included as an appendix to this chapter. A few points should be noted. Firstly, Z is calculated using a recursive formula which sums the number of legal vectors at each distance from zero to M . Secondly, the definition of $Z=1$ for $M=0$. This corresponds to the golden vector itself at the centre of the neighbourhood. There is only one vector at the centre of each neighbourhood ($M=1$) so the value of Z must be one.

The reduction in the number of independent vectors will, of course, impact the information content of the vectors. The representation efficiency measure is similar to that used earlier, but now we use the number of golden vectors rather than the total number of legal vectors in the numerator:

$$E_{max-KfromN}(M, K, N) = -\frac{1}{N} \log\left(\frac{(N-K)!K! \times Z(M, K, N)}{N!}\right)$$

where the factor $Z(M, K, N)$ in the numerator of the log accounts for the reduction in allowable vectors due to redundancy.

Table 6-2 on the following page brings all of these points together. For various bit vector lengths N , allowed '1's K , and error margins M , some of the quantities that we have been discussing are calculated: the basic number of legal vectors using the naive K -from- N formula V , the size of the neighbourhood Z , the number of golden vectors using that neighbourhood and the representation efficiency, E_{max} . Note that values for the margin equal to or greater than half the value of K are not permitted since they force the neighbourhoods of two golden vectors to overlap.

It is clear that even for small values of the margin, M , the size of each neighbourhood (in terms of the number of vectors it encloses) is large. Thus, the number of golden vectors, G , for each case is a small fraction of the total number of legal vector, V . However, despite the reduction in the number of distinguishable vectors, their number is still large even for small N and K .

Finally we see that the representation efficiency, E_{max} , scales roughly linearly with N , but the effects of a relatively large margin drastically reduce it. To

allow both a good level of redundancy (high value of M) and a high efficiency of representation (high value of E_{\max}), the values of N should be as high as possible.

Table 6-2 Measure of neighbourhood size, no. of golden vectors and storage efficiency for several network configurations.

# of Bits, N	Allowed '1's, K	Allowed Margin, M	Number of Legal Vectors with no margin, V	Neighbourhood Size, Z	Number of Golden Vectors, G	Representation efficiency, E_{\max} (bits/bit)
100	5	1	7.5×10^7	4.8×10^2	1.6×10^5	0.173
100	5	2		4.5×10^4	1.7×10^3	0.107
100	10	2	1.7×10^{13}	1.8×10^5	9.56×10^7	0.265
100	10	4		5.5×10^8	3.1×10^4	0.149
200	10	2	2.25×10^{16}	3.1×10^6	5.3×10^{20}	0.344
200	10	4		1.1×10^{10}	2.0×10^6	0.104
200	20	4	1.61×10^{27}	2.1×10^{11}	7.8×10^{15}	0.264
200	20	8		3.0×10^{18}	5.3×10^8	0.145

6.5 Consequences of Commitment to Redundancy

In the forgoing argument, the idea of redundancy was added in a somewhat cavalier fashion. It was stated that any output vector which was not one of the 'golden set' would be assumed to be erroneous and would be mapped to the golden vector into whose neighbourhood it fell. This is a reasonable objective, but how can it be achieved in practice?

The network will be updating itself every time period in accordance with its own internal dynamics. There will be no resource available to compare the output at each time step with every exemplar vector to correct for errors explicitly. The handling of golden and non-golden vectors must be a fundamental property of the network. We can set the objective for error handling in one of two ways.

6.5.1 Errors are compensated for automatically

The natural dynamics of the network must cause any erroneous vector to converge to the nearest golden vector as part of normal operation. In this case, the network must end up in a state corresponding to a golden vector before the controller moves on to the next processing step.

6.5.2 Errors are ignored when the output vector is used

In this scheme, errors in the resulting output vector are not corrected for, but it is assumed that the next stage of processing will be able to ignore them because the vector is sufficiently close to a golden vector to make no difference on the results at the next step. Now the onus is on the subsequent stages of processing to treat this vector as if it were a golden vector, despite the errors.

Realistically, both schemes will probably come into play. The next sub-section presents one possible interpretation of these constraints.

6.5.3 Convergence Constraints as Demands on Encoding

Based on the options considered in sections 6.5.1 and 6.5.2, we postulate one way in which the encoding might be constrained by the dynamics of the system. This proposal takes into account both the fact that noise will distort the interpretation of the incoming pattern and the fact that redundancy in the encoding permits some of the errors to be removed. The next few paragraphs describe the issues in more detail.

Imagine that the state of a region of neurons is represented using a K-from-N code, such as has been described throughout this chapter. The output of the neural region will provide input to other regions on the next time step during normal processing. The list of other regions might well include the source region itself, via feedback. If the neural regions act like the associative memories discussed in the review of chapter three (section 3.1.2, page 48 and section 3.1.3, page 57) then the input pattern will illicit potential due not only to the desired output pattern but also additional noise due to the overlap of other stored memory traces. Initially, this noise may cause neurons in the target regions to fire erroneously.

Part of the task performed by the target regions is to correct as many errors as possible by positive feedback, a process that depends on redundancy in the encoding of the pattern to be recalled. We know from the review of existing recurrent networks that the recall process itself can converge on states that do not exactly correspond to stored memories. We need to accept that regardless of the architecture recall may not be perfect.

Without needing to resort to specific details of the recall process, we know that the mechanism must correct sufficient errors such that the resulting output pattern is sufficiently close to the target pattern. What is sufficiently close? Close enough that, in its own turn, the resulting pattern will bring about correct pattern completion on the *next* cycle. We can define two circles around the target pattern, as shown in the figure below.

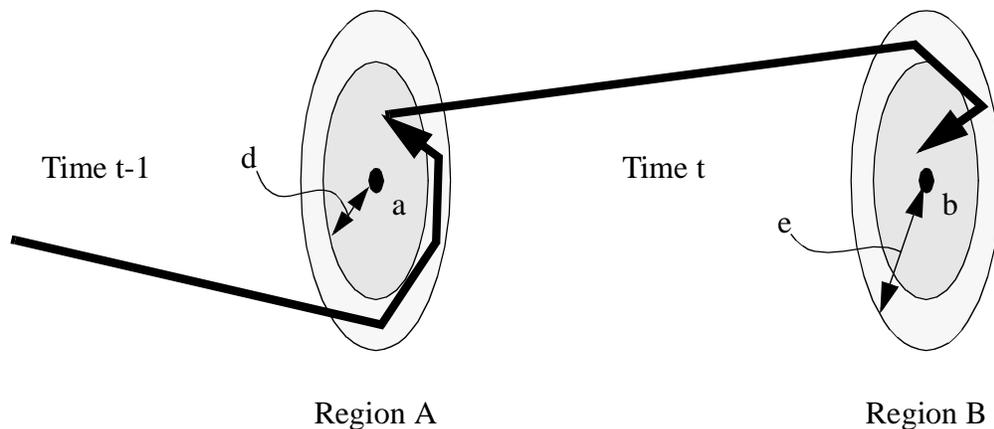


Fig. 6-3 Target circles for convergence: initial and final

The figure shows two regions. The pattern a that is represented by region A lies at the centre of two circles. Due to the imperfections of the recall process the final output of the region A at time $t-1$ may not lie at the exact centre. Instead, we permit it to lie within a distance, d , of the centre, which defines a circle that we refer to as the *final target circle*. This is the inner circle and has the dark shading in the figure. We leave the issue of fixing a value for d until a little later.

[Note that two timescales are implied in this example. The times $t-1$ and t correspond to single convergences of the networks. Within each such tick of the clock there will be many changes in the output state of the networks as they converge to a stable state. This lower level of granularity is not important to the discussion and will be ignored].

At time t , the output of region A is passed on to other regions (B in the figure). The intent is to evoke pattern b in the neurons of region B. But interference from other stored memories in region B causes errors during the first recall made in region B, so that the initial output of the region could lie anywhere inside a wider circle of radius e around the vector b , defined by the *initial target circle*, shown with lighter shading in the figure.

During subsequent cycles of region B the network will update its state vector and ideally it would converge on the target vector b . However, as was the case for region A, we will be content if the final vector lies within a distance d of the target vector.

Overall, the values for distances d and e are related and depend both on the memory loading and, more fundamentally, on the memory structure. This dependence manifests itself in the size of the basin of attraction in the target region and in the ability of the network (or lack thereof) to correct the last few errors in the recalled pattern.

We cannot arbitrarily decide to increase the final radius, d , since this defines the starting point for the projection into the next region and places constraints on the initial target radius, e . The two must be decided together. No formal method will be proposed in this thesis to calculate the values of d and e since such a method would be tied up with the convergence properties of multiply-connected, recurrent networks and is non-trivial. Instead, it is left for future work that would look specifically at the interconnection of regions of neurons.

6.6 Conclusions

In this chapter, the development of the neural building block for the network was described. As part of the development, binary neurons with activation values of '1' or '0' were selected as the basic elements of the network. Next, three coding schemes for the N-bit data vectors were considered: An unconstrained binary code, a 1-from-N encoding and a K-from-N encoding. The K-from-N code was chosen since it combines the best features of both the 1-from-N and the unconstrained codes. It is efficient in its coding of information, while the constant vector length should make it easier to control networks subject to positive feedback. Finally, the notion of adding redundancy to the coding scheme was explored.

The next chapter will investigate the use of the K-from-N code in the simple network. This work will begin with the prescription for storing vectors in the network and then analyse the network dynamics. Finally, the memory capacity of the network will be investigated, both theoretically and by simulation.

$B(m, p, N)$ is the branching factor and is a function of vector length, proportion of '1's and the current distance from the base vector. Thus:

$$Z(m, p, N) = Z(m-1, p, N) \times B(m, p, N) \dots \dots (2)$$

To find $B(m, p, N)$, we consider how many patterns we can generate at distance m . There were pN '1's in the left hand side of the base vector, where p is the proportion of '1's in the N bit vector. But of those pN '1's, $m-1$ have already been moved out. Thus, there are $pN-(m-1) = pN-m+1$ remaining. Each of these can be moved to produce a new vector.

Now consider the '0's into which we could move the selected '1'. There were $(1-p)N$ '0's in the base vector, but $m-1$ have already been filled with '1's. Thus there are $(1-p)N-(m-1)$ remaining to choose from. Any one could be filled to generate a valid next vector. Thus the number of valid vectors which we could generate at this stage is the product of these. i.e. $[pN-m+1] \times [(1-p)N-m+1]$. However, this is not the value of the branching factor, B . There is one further factor to take into consideration: duplication. If we consider an arbitrary vector a distance m from the base vector, we see that there are several vectors at distance $m-1$ which could have given rise to it.

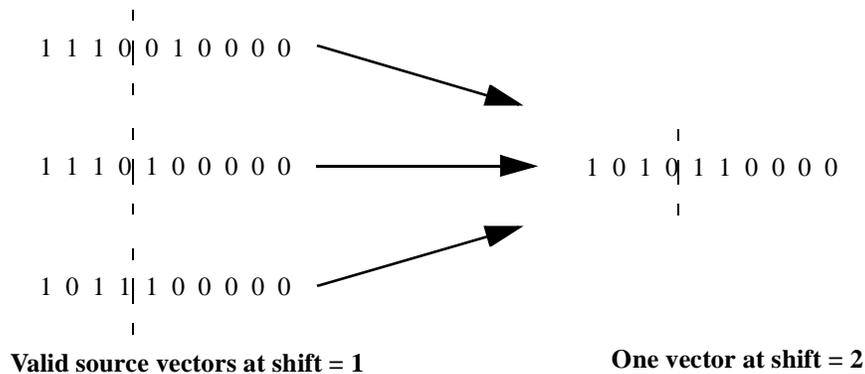


Fig. 6-5 A vector at shift k can have multiple parents at shift $k-1$.

In fact, there are precisely m '1's which could have been moved to reach the vector at shift = m , and precisely m '0's which could have been the one that was just filled. Thus each vector at shift = m had m^2 progenitors at shift = $m-1$. This provides the denominator to the branching factor, which thereby counts each vector only once. Thus we obtain:

$$B(m, p, N) \equiv \frac{1}{m^2} [(pN - m + 1) \times ((1 - p)N - m + 1)]$$

$$= \frac{1}{m^2} [N^2 p(1 - p) - (m-1)N + (m-1)^2] \dots \dots (3)$$

which, when substituted back into equation (2), gives equation (1) which was to be proven.

Analysis of Storage and Retrieval for the Simple Network

7.0 Introduction

In the previous chapter, the development of the basic neural building block was begun. It was decided that the network must be based on some sort of recurrent structure in order to meet the barest requirements of the architecture specification. In addition, the fundamental structure of the block was defined as a group of identical sum-and-threshold units which are highly interconnected. Finally, by considering the properties of various coding schemes for the data vectors themselves, the K-from-N coding scheme was selected on the grounds of its efficiency of information storing and the properties of predictability and controllability.

Having selected the coding scheme, the next logical step was to consider the basic learning procedure of the network. Essentially, the aim was to answer four questions: First, how does the network update itself at each time step? Second, what is the procedure for storing patterns in the network? Third, what happens during the retrieval process? And finally, how many patterns can be stored simultaneously?

These key questions will be considered in order. Answering them will complete the definition of the basic network. Subsequent chapters will add functionality to control the processing performed by the network and to improve its storage capacity using more advanced learning techniques.

7.1 The Patterns to be Stored

As described earlier, the patterns to be presented to the network for memorisation were K-from-N vectors. They were drawn at random from the set of all such legal vectors of length N. Each such vector contains exactly K '1's and N-K '0's. Choosing any two vectors from this set, it is clear that they are not independent: the dot product of two such vectors, V_1 and V_2 is:

$$u = \sum_{j=0}^{N-1} V_1^j \cdot V_2^j$$

Thus, $E(u) = \frac{K^2}{N}$

where $E()$ is the expectation operator. For all useful cases $K > 0$ and $E(u)$ is therefore a positive, non-zero quantity which is a consequence of the non-independence of elements in each vector. A set of golden vectors as defined in the last chapter would consist of a subset of the complete set of legal K -from- N vectors.

7.2 Network Output Updating

The basic network is shown in the figure below. The network consists of N binary-thresholding neurons, each of which receives an afferent connection from every other neuron. This input is modulated by a set of connection strengths, W .

The outputs of all neurons are monitored to ensure that precisely K neurons out of the N (those possessing the most potential above their own threshold, T_i) are firing (i.e. have output '1') at any time. The rest are silent (i.e. have output '0'). The K -from- N rule is enforced by an extra input, V , called the activity regulator.

The V input provides an equal background level of input to every neuron in the region. If more than K neurons fire, this added background level is reduced until only K are firing. Conversely, if fewer than K neurons fire then the background level is increased until exactly K are firing. The time constant for this background level is sufficiently small that the adjustment is effectively instantaneous compared to the normal updating of the neurons.

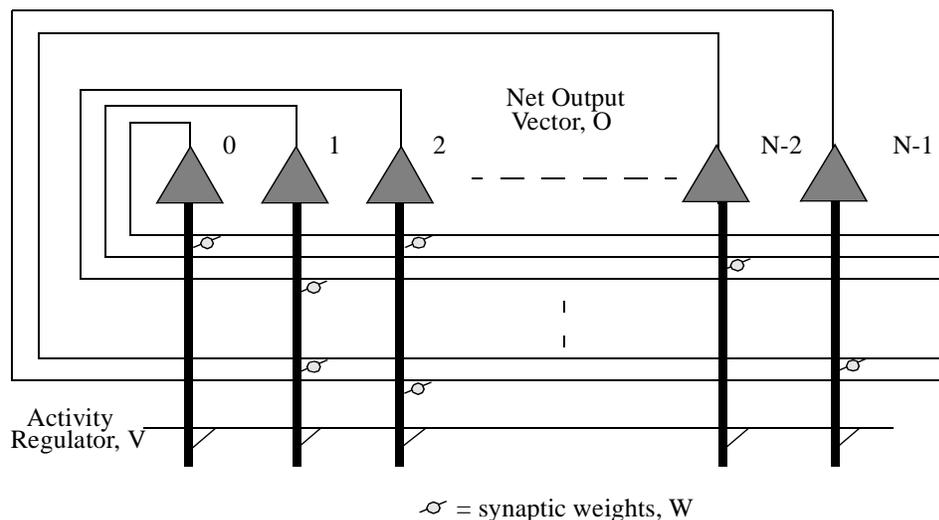


Fig. 7-0 Basic network showing N fully connected elements and external input

The potential, U_i , of a single neuron at time t has three components:

$$U_i(t) = \sum_{j=0}^{N-1} O_j(t-1) \times W_{ij} + X_i(t-1) + V$$

where O_j is the output value of neuron j , W_{ij} is the strength of the connection from neuron j to neuron i . How the values of W_{ij} are assigned will be discussed in the next section. The X_i term is the external input to neuron i coming either from the environment or perhaps another network. Finally, the V term is the activity regulator mentioned earlier.

The output of neuron i , O_i , is calculated using the threshold function:

$$O_i(t) = 1 \text{ when } U_i(t) - T_i \geq 0$$

$$= 0, \text{ otherwise}$$

where T_i is the threshold of neuron i . At each stage of processing, the inputs are applied to the network and new potentials and outputs are calculated. The activity regulator, V , is adjusted to ensure that only K from N neurons have enough potential to fire:

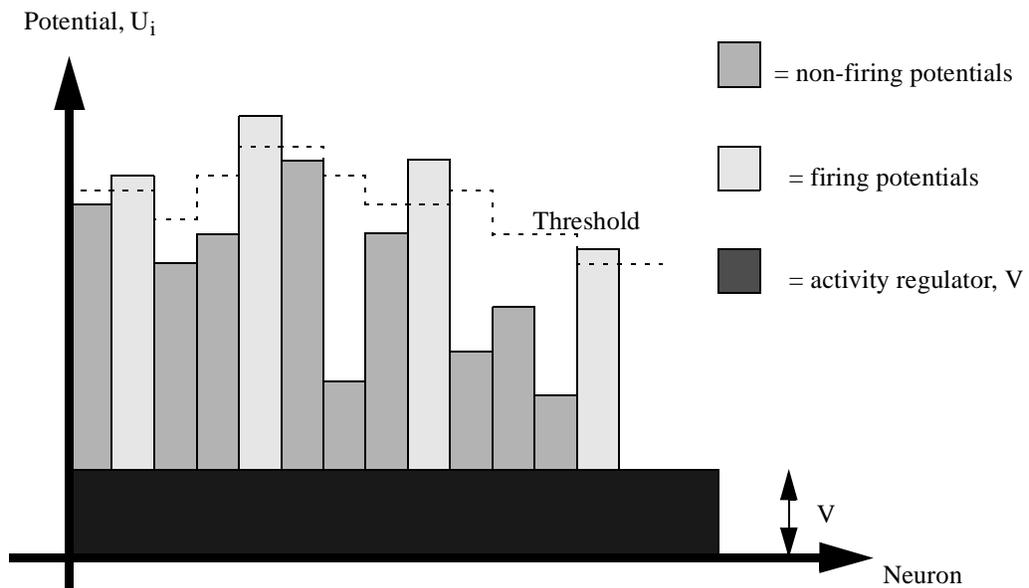


Fig. 7-1 Graph of potentials for the neurons in a single region

The threshold of each neuron is indicated by the dotted line. Note that each neuron has its own threshold, though the long term aim of each neuron is to try to maintain the same value as its peers. This point will be expanded upon later.

A useful way of interpreting the output threshold operation is that the K neurons which are permitted to fire are those whose potentials surpass their threshold by the widest margin.

7.3 Storing Patterns

The arrangement of the network is very similar to that of a standard Hopfield network. However, the storage prescription which was employed was not at all similar. Replacing the additive formula of the Hopfield network was the learning scheme of the non-holographic associative memory (Willshaw, Buneman & Longuet-Higgins, see chapter two).

It should be remembered that this storage prescription was used by Willshaw *et al.* in a solely feedforward, hetero-associative, network to create a matrix of connections from inputs, \mathbf{X} , to outputs, \mathbf{O} . Here, it was applied to a fully connected, auto-associative, network. A recap on the hetero-associative storage prescription for the non-holographic associative memory is in order, before adapting it for the network under study.

Initially, the weight matrix, W , is empty, i.e. all weights are zero. As each pattern is presented to the network the connections, W_{ij} , are made according to:

$$W_{ij} = B, \text{ when } X_j = 1 \text{ and } O_i = 1 \\ = \text{unchanged, otherwise}$$

B is a constant, defined as the basic weight strength. Whenever a connection is made in this simple network, it has strength B . Since ultimately the choice of which neurons will fire is based on their *relative* potential rather than an *absolute* value then B is an arbitrary constant¹.

A small aside is necessary at this point to highlight the difference between the Hopfield prescription and the “absolute” prescription of the non-holographic memory. Notice that in the proposed procedure, no change is made to the network as a result of making a connection if one exists already. This contrasts with the additive Hopfield prescription where repeated storage, even of the same pattern, would continue to change the value of weight W_{ij} . This difference has both positive and negative implications for the simple network being discussed.

The non-additive nature of the algorithm is desirable because a stored pattern is harder to disrupt (a point that will be quantified later on). The undesirable aspect is that the relative frequency of presentation of a particular pattern is no longer discernable from the network. A network in which one pattern was presented 99% of the time would have the same weight vector at the end of training as one in which that same pattern was presented, say, only 10% of the time. For some applications, this is acceptable but for many it is not.

Returning to the discussion of network storage, what does this storage prescription mean in physical terms for the network? It has already been stated that the

1. As the network is developed in later work, extra complexity will be added and it will be necessary to place constraints on the value of B .

difference between the standard non-holographic memory and this network is that the hetero-associative mechanism of the former is replaced by an auto-associative one in the latter. No neurons are designated either as inputs or outputs. Instead all neurons receive an input and produce an output which is seen by all other neurons. To illustrate this, the figure below shows the neurons arranged in a 2-D array, with the K firing neurons shaded.

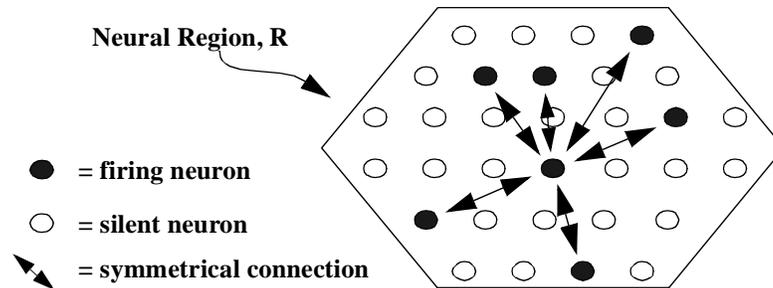


Fig. 7-2 The learning process in the basic network

In this example, the number of neurons, $N = 32$, while the number firing, $K = 7$. The process of learning a new pattern begins by forcing the pattern onto the neurons. (For the basic network this is accomplished using external signals which take precedence over the internal connections of the network; More sophisticated methods will be discussed in later chapters).

The learning itself consists of setting to a constant, non-zero, value any weight which connects two firing neurons. In the figure, the black arrows indicate these new connections for only one of the firing neurons. Every firing neuron makes connections in this way, but for the sake of clarity only those for a single neuron are shown.

We can now state the actual learning rule that was used for the simple network, as a modified form of the procedure used in the Willshaw memory.

As pattern P_i is applied to the network, forcing the outputs of the neurons to conform to that pattern (such that $O_i = P_i$), then connections between neurons are modified such that:

$$W_{ij} = B \text{ when } O_i = 1 \text{ and } O_j = 1 \text{ and } i \neq j$$

$$= \text{unchanged, otherwise}$$

where B is a constant. Note that the condition $i \neq j$ ensures that a neurons does not make a connection with itself.

After learning Z patterns in this way, the weight matrix will be filled with elements each of which has a value of either zero or B . Choosing any two neurons, i and j , at random, the connection from i to j will have the same value as that from j to i since the storage prescription naturally gives rise to a symmetrical matrix. The probability that this connection has been made (i.e. has a value of B) is a useful quantity in the analysis of the memory capacity of the network. Here, it is given

the symbol h . The more patterns that have already been stored in the network, the more likely it is that a connection already exists between any two neurons.

The calculation of h follows the same principles as that described in Willshaw, *et al.*(1969). Assuming that Z patterns have been stored in the network, then the probability that a connection exists between any two neurons is given by:

$$h = 1 - (1 - p^2)^Z$$

where p is defined as the proportion of firing neurons, $\equiv \frac{K}{N}$

The proof: on any presentation, the probability that a particular neuron fires is p . Thus, the probability that two particular neurons fire (and will have a connection made between them) is p^2 . The probability that this connection will *not* be made for a particular pattern is $1 - p^2$. For the connection to remain unmade for all Z patterns this independent event must be repeated Z times. Thus, the probability of no connection over all Z patterns is $(1 - p^2)^Z$. Finally, the probability that no connection was made between these two neurons during any of the trials is one minus this probability. Hence we obtain the definition of h as $h = 1 - (1 - p^2)^Z$.

The probability, h , is zero when $Z=0$, i.e. when no patterns have been stored. As each pattern is learned, this probability increases. Since $0 \leq (1 - p^2) < 1$, then $h \rightarrow 1$ in the limit as $Z \rightarrow \infty$.

7.4 Weight Matrix Saturation

In this section, we explore the consequences of the choice of learning procedure. This discussion will not consider the network updating procedure nor the network dynamics. These will be described in later sections. Instead it will look at some of the metrics which will be useful to analyse the memory capacity of the network and the “resilience” of individual patterns in the face of continual learning.

To begin, consider the case where the network is in a state corresponding to a stored pattern. The output of each neuron, O_i exactly corresponds to the pattern P_i stored earlier. What are the potentials of the firing and the non-firing neurons?

$$U^{firing}_i = \sum_{j=0}^{K-1} (1 \times W_{ij}) = B(K-1)$$

$$E(U^{non-firing}_i) = E \left(\sum_{j=0}^{K-1} (1 \times W_{ij}) \right) = BhK$$

where $h \equiv$ probability of an existing connection

The potential for a firing neuron is equal to $B(K-1)$, the total potential from $K-1$ connections. Once again, the '-1' term takes into account the fact that no neuron makes a connection with itself.

For the non-firing neurons, the potential depends on the statistics of previous learning so consider only the expectation of the potential which is related to the probability of a connection, h . Since there are K firing neurons, which have a probability h of make a connection of strength B with a non-firing neuron then the expected potential is BhK .

Consider the properties we expect for the network whose state corresponds to a stored pattern. What we demand is that the potential of neurons which are to fire in the pattern (the "on" neurons) must be higher than the rest (the "off" neurons). If this condition is met, the pattern will be a stable point in the network dynamics. If the potential of one of the "off" neurons is higher than that of an "on" neuron, then the former will begin firing at the expense of the latter; the pattern is no longer a fixed point and, to all intents and purposes, has been forgotten. Thus we need the following condition to be true:

$$U^{firing} > U^{non-firing}$$

$$B(K-1) > BhK$$

$$K-1 > hK$$

[Note that this is true only in the thermodynamic limit of $N \rightarrow \infty$ and $K \rightarrow \infty$, in which case the actual number of connections between "on" neurons and a particular "off" neuron approaches the expected value, hK . The variability of the number of such connections is considered later on].

For small Z , $h \ll 1$; thus the potential of any non-firing neuron will be low relative to that of a firing neuron. None of the non-firing neurons will receive enough potential to surpass any of the firing neurons so the state will persist indefinitely: each stored pattern is a stable state of the network.

The next step is to consider some of the factors which influence h , the connection probability. From the definition of h , we see that it is a function of p , the fraction of firing neurons in any pattern and Z , the number of stored patterns. A graph of h against Z for various values of p appears on the next page.

Examining the shape of the curves, it is clear that $h = 0$ when $Z = 0$ and $h \rightarrow 1$ when $Z \rightarrow \infty$ regardless of the value of p . However, p does influence the rapidity with which this convergence occurs with respect to Z . For small p , the rate of convergence is lower than for high p .

Next, we note that the probability of a connection increases as the number of stored patterns increases. The rate of change of connection probability with number of stored patterns, dh/dZ , is positive. But it is also clear from the figure that this rate itself (while remaining positive) is much higher for low Z than for high Z . This can be easily shown by differentiating h with respect to Z :

$$\frac{dh}{dZ} = -\ln(1-p^2) \times (1-p^2)^Z > 0, \quad (\text{since } \ln(1-p^2) < 0)$$

and, $\frac{d^2h}{dZ^2} = -[\ln(1-p^2)]^2 \times (1-p^2)^Z < 0$

This is in line with intuition; as more and more patterns are stored, the chances of trying to remake an existing connection will increase. Thus, fewer *new* connections will be necessary with each additional pattern stored.

Finally we consider the relationship between h and network saturation. By definition, that fact that $h = 1$ implies that every possible connection between neurons has been made. The network is incapable of differentiating a known pattern from an unknown one and is therefore saturated. For the network to be useful the value of h should be kept below 1.

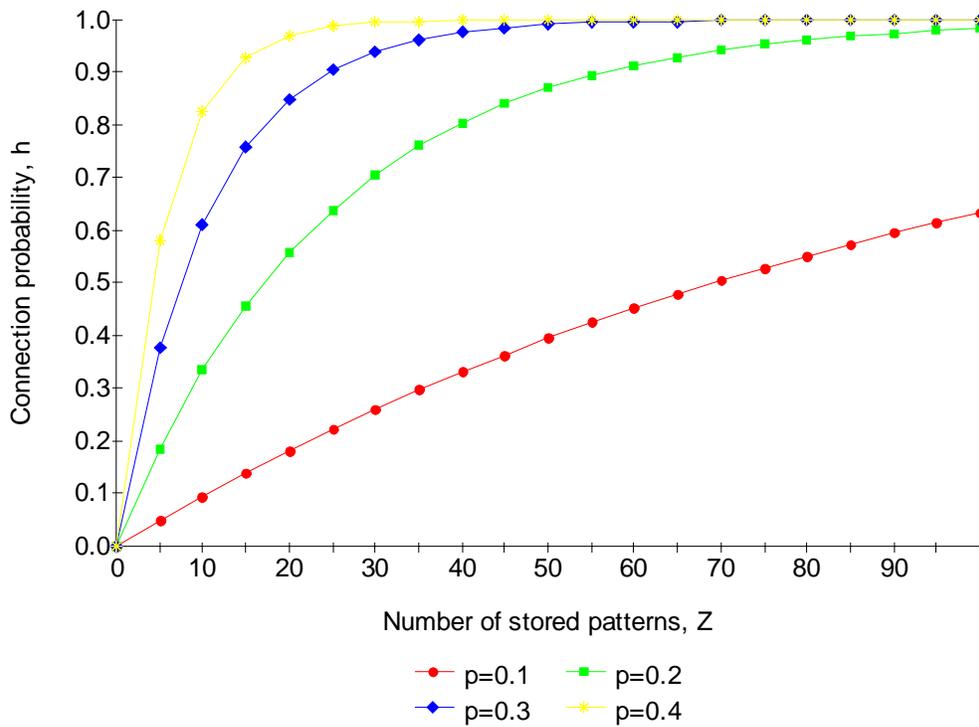


Fig. 7-3 Graph of connection probability vs number of stored patterns

Nadal & Toulouse showed that the maximum storage density in the weight matrix occurs when $h=0.5$. However, the approach taken here does not necessarily seek to achieve the highest storage density. There are other constraints such as the existence of suitably wide basins of attraction around each stored memory that were not considered by Nadal & Toulouse, nor any other author discussed in the review chapters. These considerations prevent us from merely investigating the $h=0.5$ case alone.

For values of p even as high as 0.3, the probability of a connection rises sharply towards 1 with increasing Z . Therefore, to keep h small for large Z , the value of p should be as low as possible. But low p corresponds to low information content per pattern. Thus, there is a trade-off between the maximum number of patterns stored and the information content of each pattern, and this trade-off is controllable through p .

The disadvantage of a value of p near zero is that the quantity of information stored in each vector is also low. The analysis of memory capacity is presented later in this chapter and will quantify this.

7.5 The Process of Retrieval: Network Dynamics

As viewed from the outside, the act of retrieval is essentially a task of searching for the stored memory that “best” matches the given input. The definition of best match is given by the basic difference metric first stated in chapter six.

However, the internal mechanism of the network during retrieval is better described as a minimisation of an energy function in N -dimensional space. For each state of the network, there is a corresponding energy, given by:

$$E = -\frac{1}{2N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} O_i O_j W_{ij} + \frac{1}{2N} \sum_{i=0}^{N-1} T_i O_i$$

where T_i is the threshold of neuron i and the sums over i and j range over the complete set of neurons. This energy is a measure of the relative “goodness of fit” between the current network state and the stored memories. It has two components; the first is due to the inter-neuron connectivity. The second component, of opposite sign to the first, is due to the internal threshold of each firing neuron. It has the effect of penalising the firing of high threshold neurons with respect to their lower threshold peers.

External inputs, X , are not considered at this time; they are tacitly present in the setting of the initial state of the network, but are subsequently removed and do not affect the dynamics of the simulation.

When the system is in a state other than that of a previously stored memory, updates will generally cause changes in the pattern of firing neurons. The next step in the analysis is to show that all such changes reduce the value of the energy function, E . This analysis is similar (but not identical) to that used for the Hopfield network (Hopfield, 1982). Recall from chapter two that the state vector in a Hopfield network consists of ‘+1’ and ‘-1’ values whereas here the values are ‘+1’ and ‘0’. Also the state vector in the Hopfield network can contain +1 and -1 values in any proportion; this network requires exactly K -from- N of the bits to be +1 and the rest to be 0. These differences alone justify a reverification of the dynamic properties of the network to be sure that properties of the Hopfield network can be reliably extrapolated to this network. It turns out that they cannot.

The task of the analysis is to show, given the update rule for the neuron outputs, that the postulated energy function does indeed characterise the system and hence, that it can be used to make further predictions about network performance.

We proceed by considering the change in system energy, E , between two time points t and $t+1$. The energy of the system at these two times is given by:

$$E(t) = -\frac{1}{2N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} O_i(t) O_j(t) W_{ij} + \frac{1}{2N} \sum_{i=0}^{N-1} T_i O_i(t)$$

$$E(t+1) = -\frac{1}{2N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} O_i(t+1) O_j(t+1) W_{ij} + \frac{1}{2N} \sum_{i=0}^{N-1} T_i O_i(t+1)$$

The change in energy for the whole system is then:

$$E_{diff} \equiv E(t+1) - E(t)$$

$$= -\frac{1}{2N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} [O_i(t+1) O_j(t+1) - O_i(t) O_j(t)] W_{ij} + \frac{1}{2N} \sum_{i=0}^{N-1} [O_i(t+1) - O_i(t)] T_i$$

If no neurons have changed state between time t and time $t+1$, then $O_i(t+1) = O_i(t)$; thus, the total change in energy is zero, as expected. If the state has changed, then the K-from-N rule implies that for every neuron which has begun to fire, another has ceased. This is the first major divergence from the Hopfield model.

Let A denote the neuron which was firing at time t but ceases to fire at time $t+1$. Let B denote the neuron which begins to fire only at time $t+1$, replacing A in the active set of neurons. Since we update the neurons asynchronously, the only changes which can contribute are the change in energy in a single time step are due to neurons A and B. Therefore, we can simplify the equation for E_{diff} by including only terms in which they appear:

$$E_{diff} = -\frac{1}{N} \sum_{i=0}^{N-1} O_i(t+1) W_{Bi} + \frac{1}{N} \sum_{i=0}^{N-1} O_i(t) W_{Ai} + \frac{1}{N} (T_B - T_A)$$

There are now only three terms, each relatively simple. The third term contains threshold information. Only the thresholds of the firing neurons are visible in the energy difference, so this term represents a change from a contribution by the threshold of neuron A to that of neuron B. The first and second terms show the change in energy resulting from every neuron's connections to and from neurons B and A respectively. The factor of $1/2$ in the original equation has gone since we have summed two terms for each neuron pair: one *from* neuron A and one *to* neuron A (*idem.* for neuron B).

From the above equation, we have an expression for the energy change due to a non-firing neuron supplanting another in the active set. But at this point there is no indication of which neuron is supplanting which other. We need to introduce the dynamic update rule to answer that question. This rule dictates that the K neurons that fire are those which have the greatest surplus potential over and above their own threshold. Since B has supplanted A in the active firing list during the update from time t to time t+1, we know that

$$U_A(t+1) - T_A < U_B(t+1) - T_B$$

where $U_A(t+1)$ and $U_B(t+1)$ are the potentials at time t+1 of neurons A and B respectively, and T_A and T_B are their respective thresholds. These potentials can be rewritten in terms of the outputs and weight matrix and then simplified to give a simple expression for dynamic update rule. Thus:

$$\begin{aligned} \sum_{j=0}^{N-1} O_j(t) W_{Aj} + V - T_A &< \sum_{j=0}^{N-1} O_j(t) W_{Bj} + V - T_B \\ \sum_{j=0}^{N-1} O_j(t) W_{Aj} - T_A &< \sum_{j=0}^{N-1} O_j(t) W_{Bj} - T_B \\ (T_B - T_A) &< \sum_{j=0}^{N-1} O_j(t) (W_{Bj} - W_{Aj}) \end{aligned}$$

The result seems intuitively correct. What it is saying is that for neuron B to overcome neuron A and take its place in the active firing list then not only must B overcome A in potential (the right hand side) but it must do so by a margin equal to the difference of the two thresholds (the left hand side). This captures the idea that the update rule only permits a neuron to start firing if its *excess* of potential over and above its threshold is greater than that of an already firing neuron.

The final step in the analysis is to combine this new result with that for the energy difference, E_{diff} , given earlier. The easiest way to do this is to note that both expressions contain a term $(T_B - T_A)$. Substitution then gives:

$$\begin{aligned} E_{diff} &= -\frac{1}{N} \sum_{i=0}^{N-1} O_i(t+1) W_{Bi} + \frac{1}{N} \sum_{i=0}^{N-1} O_i(t) W_{Ai} + \frac{1}{N} (T_B - T_A) \\ E_{diff} &< -\frac{1}{N} \sum_{i=0}^{N-1} O_i(t+1) W_{Bi} + \frac{1}{N} \sum_{i=0}^{N-1} O_i(t) W_{Ai} + \frac{1}{N} \sum_{i=0}^{N-1} O_i(t) (W_{Bi} - W_{Ai}) \\ E_{diff} &< \frac{1}{N} \sum_{i=0}^{N-1} [O_i(t) - O_i(t+1)] W_{Bi} \end{aligned}$$

Thus the change in energy can be calculated purely in terms of weights to and from neuron B. The sum over the change in output of all neurons, i, reduces to two terms, one for A (since $O_A(t) = 1$ and $O_B(t+1) = 0$) and one for B (since $O_B(t) = 0$ and $O_B(t+1) = 1$).

For all other neurons, the change in output is zero so their contribution to the change in energy is also zero. Applying these simplification yields:

$$\begin{aligned} E_{diff} &< \frac{1}{N}(W_{BA} - W_{BB}) \\ &< \frac{W_{BA}}{N} \end{aligned}$$

In the final line we make use of the fact that no neuron is permitted to make a connection to itself, so that $W_{BB} = 0$.

The final result of the analysis shows that with the proposed update scheme the energy, although bounded from above, is still permitted to increase at each time step, by an amount proportional to the magnitude of the weight between the two neurons whose states are changing. This is a major divergence from the Hopfield result and is caused by a hidden asymmetry between the potentials before and after the neurons are updated.

It should be noted that the update decision is based purely on information available at time t . At that time, neuron B has acquired more surplus potential than neuron A and thus is permitted to replace it in the active set. Upon arrival at time $t+1$, the energy contributions from every neuron except A and B have indeed been reduced, as predicted.

The problem arises from the fact that at time t one contribution of potential which helped neuron B to overcome A came from A itself. At time $t+1$ this contribution is now gone and has not been replaced by an auto-associative term since no neuron is permitted to make a connection to itself. Thus the actual potential of B at time $t+1$ will be slightly reduced relative to its potential at time t , contributing a positive term to the system energy. Similar arguments show that neuron A, while defeated at time t , may see its potential increase at time $t+1$ due to a contribution from the now dominant and firing neuron B. Each of these effects contributes $\frac{1}{2N}W_{AB}$ to the energy change.

So does this imply that such a system will gain energy and thus will not converge? That depends on the relative contributions of the energy dissipation by the other firing neurons and the energy generation due to A and B (fixed at W_{AB}). Clearly, in the limit of a very large number of neurons, N , the relative contribution of a single weight will be swamped. Even so, the possibility of gaining energy at any given step is a source of concern. At best, a positive contribution to the energy will slow the rate of convergence. At worst it could disrupt convergence altogether. One task of future development will be to try to eliminate this positive term.

Simulation results of the network, illustrating the issues of convergence, will follow the section on the analysis of memory capacity.

7.6 Analysis of Memory Capacity

The focus of attention now shifts to another important issue in network design, that of memory capacity. From the discussion in chapter six on the amount of information stored in a single vector using K-from-N coding, it was shown that the maximum information that can be represented is only a fraction of a bit per bit of the vector. This is due both to the basic coding scheme itself (which imposes a statistical dependence on the value of each bit given the others) and to the redundancy which is added to cope with errors in the vector during processing. The definition of the memory capacity, $C(K, N, M)$, of the network is:

$$C(K, N, M) \equiv Z_{\max}(K, N, M) \times I_{\max}(K, N, M)$$

where Z_{\max} is the maximum number of legal vectors that can be stored and reliably recalled in the network and I_{\max} is the maximum number of bits of information that can be stored in each vector. Note that all the quantities here are functions of K, N and M.

The definition is simplistic in that it treats the network as a device whose sole purpose is to store and retrieve unrelated vectors. A more sophisticated analysis of the network (left for subsequent chapters) will consider more complex powers of storage and recall based on structure within the patterns themselves and relations between patterns. Such structured storage cannot increase the fundamental capacity of the network in terms of the number of bits it can store. Data structure storage is usually based on correlations between patterns which hold less information than their uncorrelated peers. Thus the measure stated above, although crude, does place a true upper limit on absolute storage capacity and thereby retains some merit.

Since the value of I_{\max} was considered in the previous chapter for various values of K, N and M, it is the derivation of a formula for Z_{\max} which is needed. The first step in evaluating the maximum number of vectors that can be stored in a network is to decide what actually constitutes successful storage. A number of metrics have been proposed and the choice of metric depends on the application.

One possible measure, often applied to MLPs, is to insist upon, say, 99% recovery of stored bits over the whole pattern set (Rumelhart, et al. 1986). Here, individual patterns may contain multiple errors; only the ensemble average counts.

A more strict measure places bounds on the number of errors permitted within each and every pattern (Amit, 1990). This latter measure is more appropriate in a system where the output of one processing step feeds directly into the next. An ensemble average of error is irrelevant if a single pattern is permitted to contain sufficient errors to lose the sense of a result and thus ruin an ongoing calculation. Thus the stricter measure will be applied in this analysis.

Thus, for this basic network, a pattern vector, \bar{x} , can be said to have been successfully stored if it fulfils two requirements. Firstly, the vector must be a stable state of the network dynamics. Secondly, when presented with a version of \bar{x} in which some bits have been corrupted (up to a known limit, L), the network must be capable (in a finite number of updates) of correcting enough errors to bring the vector to within a specified distance, M , of the target vector, as defined by a distance metric, d . The metric d , as specified in chapter six, defines the difference between two K -from- N vectors as the minimum number of '1's that must be moved to transform one vector into the other.

Two scenarios must be considered. The general case allows a certain number of bits to be corrupted. To say that the memory is retrievable is then to ask that the network is capable of correcting enough of the errors to qualify it as an example of the stored memory. The more fundamental case is that in which the network begins already in a state corresponding to a previously stored memory. If this memory is correctly stored, this state should be a fixed point, i.e. subsequent updates of the network should not change its state. We consider this special case first, then follow with the general case.

7.6.1 Network State Corresponds to Stored Pattern

For a network which lies in a state corresponding to a stored memory, the graph below illustrates the potentials generated in both firing and non-firing neurons.

Each firing neuron makes a connection of strength B with all other firing neurons. Hence, its potential is $B(K-1)$ with probability one. In the ideal case, the potentials of non-firing neurons would all be zero. In practice each non-firing neuron has a non-zero potential due to connections made to currently firing neurons. These connections were made while learning patterns other than the current one and are, therefore, a source of noise in the current context.

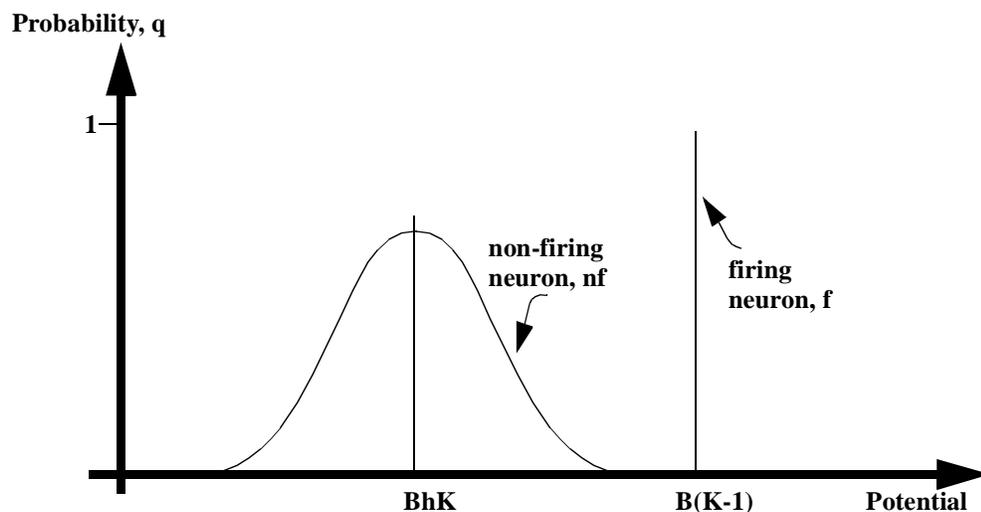


Fig. 7-4 Potential distributions for neurons in known pattern state.

Since the probability that a non-firing neuron makes a connection with a firing neuron is h (as defined earlier) then the expected value of the potential that each non-firing neuron receives from the active neurons is based on K random events, each of probability h . The binomial distribution gives the mean potential of each non-firing neuron, U_{nf} , and its variance, σ_{nf}^2 , as:

$$U_{nf} = BhK$$

$$\sigma_{nf}^2 = B^2h(1-h)K$$

For large K the distribution approaches that of a Gaussian. Provided the potential of every non-firing neuron remains below that of every firing neuron, the stored memory patterns will be stable points in the network dynamics. However, if a non-firing neuron manages to attain a potential greater than that of a firing neuron it will displace it from the pattern, firing in its place and producing an incorrect output state. The probability that a particular non-firing neuron could reach the firing potential is simply the area under the gaussian curve in which the potential is greater than $B(K-1)$. This is shown in the next figure.

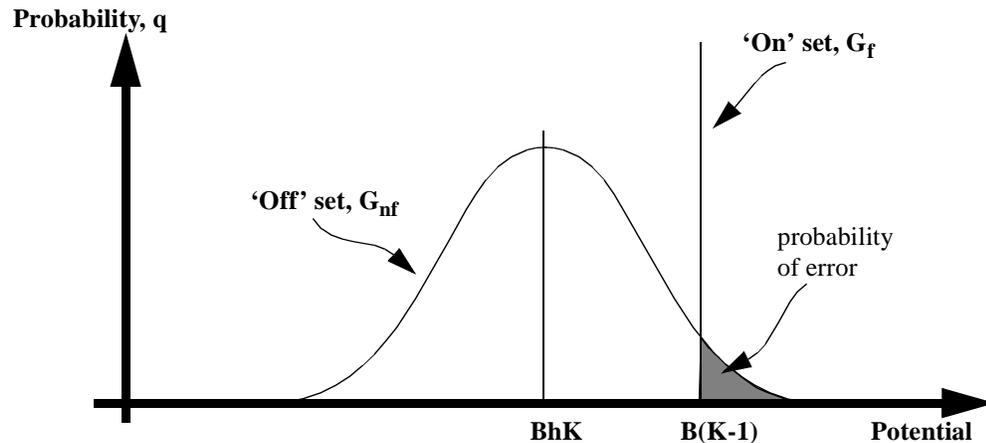


Fig. 7-5 Probability that neuron that should be silent might fire

For the case in which the network state corresponds to a stored pattern, we know that the only way a non-firing neuron can surpass the potential of a firing neuron is if it makes a connection with *every* firing neuron. Its potential will then be BK , more than the $B(K-1)$ of any currently firing neuron. Since the probability of making any given connection is h , the probability of making K such connections is h^K . However, any of the $(1-p)N$ non-firing neurons could achieve this potential, so the expected number of 'off' set neurons doing so for pattern y is:

$$N_{err}(y) = (1-p)N \times h^K$$

$$= (1-p)N[1 - (1-p^2)^Z]^{pN}$$

substituting for K and h . For the whole pattern set consisting of Z patterns, the expected number of errors is therefore:

$$N_{err}(Z) = N_{err}(y) \times Z$$

$$= Z((1-p)N[1 - (1-p^2)^Z]^{pN})$$

The graph below shows the expected error for a single pattern in a network of 200 neurons as a function of the number of stored patterns, Z . Each graph corresponds to a different value of p .

The rate of divergence can be clearly seen to increase for increasing p . Note that the expected number of errors even for $Z=N=200$ is still less than one per pattern for $p=0.07$. This is a statistical effect in the limit of an infinite network, however. In practise the network will become saturated as Z approaches N and it will become increasingly difficult to differentiate between patterns since the potential of every neuron will be near maximum and hence nearly identical.

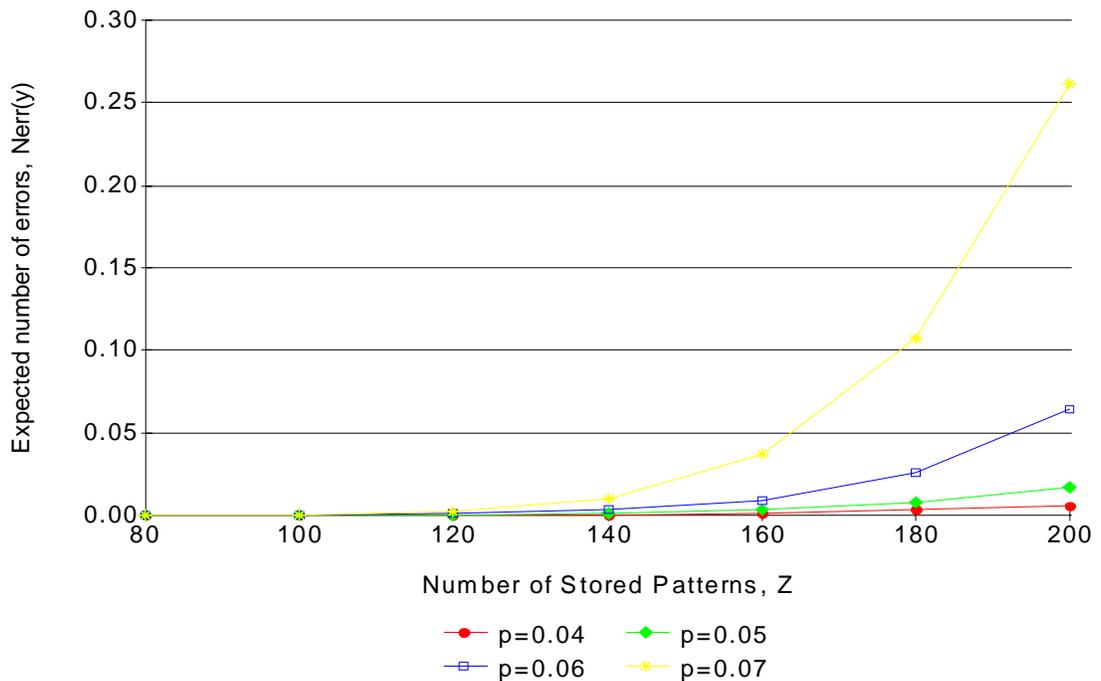


Fig. 7-6 Graph of expected errors per pattern vs. the number of stored patterns

The graph in figure 7-7 shows the expected errors over the entire pattern set, $Nerr(Z)$. Again, a network of $N=200$ neurons was assumed and the graph is plotted for several values of p .

The number of errors as a function of Z grows exponentially. If the goal is to avoid any retrieval errors then even for the low firing level of $p=0.05$, then in this case the network can tolerate no more than 130 patterns before the expected error is greater than 1 bit.

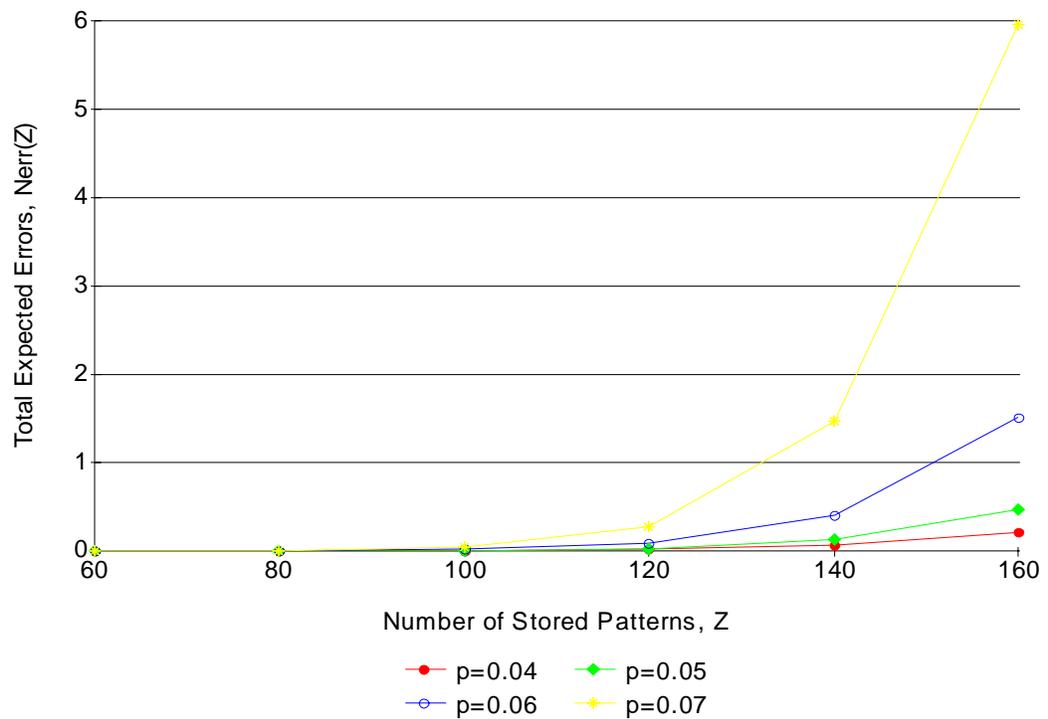


Fig. 7-7 Graph of total expected errors vs. number of stored patterns.

7.6.2 Network State is a Corrupted Version of a Stored Pattern

In this more general scenario, the initial network state does not correspond to a stored memory. This is a common situation in which the network is being used to categorise a new pattern into one of a number of classes specified by the exemplar templates (the stored patterns). The goal would be for the network state to evolve in time so as to converge on the stored pattern which is 'closest' to the input pattern, ideally in the minimum number of updates.

Let the target stored pattern be the vector \bar{G} . Let the network state at time t be $\bar{s}(t)$. The initial vector, $\bar{s}(0)$ is similar to the target vector, \bar{G} , but a certain number of '1's in $\bar{s}(0)$ occupy different bit positions to those in \bar{G} . Let that number of shifted bits be c . Let the set of neurons which are 'on' in target pattern \bar{G} be G_f and let the set of neurons which are 'off' in target pattern \bar{G} be G_{nf} . The example below shows a 5-from-10 target vector, and examples of vectors with c varying from one to four.

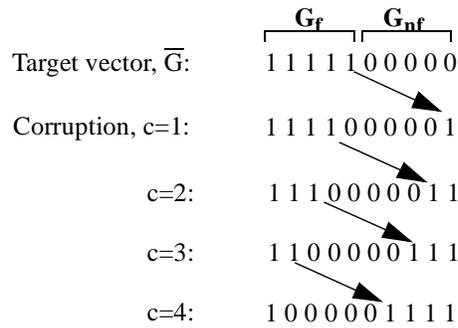


Fig. 7-8 Example 5-from-10 vector at several levels of corruption

As the network state is updated, beginning at $\bar{s}(0)$, the aim is for the number of corrupted bits to be reduced monotonically until the network state coincides exactly with stored vector, $\bar{\mathbf{G}}$. But in this general case there are neurons firing which belong to the set \mathbf{G}_{nf} and thus are a source of slow noise with respect to target pattern, $\bar{\mathbf{G}}$.

The graph below shows the probability curves for the potentials of neurons from sets \mathbf{G}_f and \mathbf{G}_{nf} . Notice that the potential of the \mathbf{G}_f neurons is not constant at $B(K-1)$ as in the previous case. This time there is a random contribution coming from the firing members of the set \mathbf{G}_{nf} which are uncorrelated with target vector, $\bar{\mathbf{G}}$.

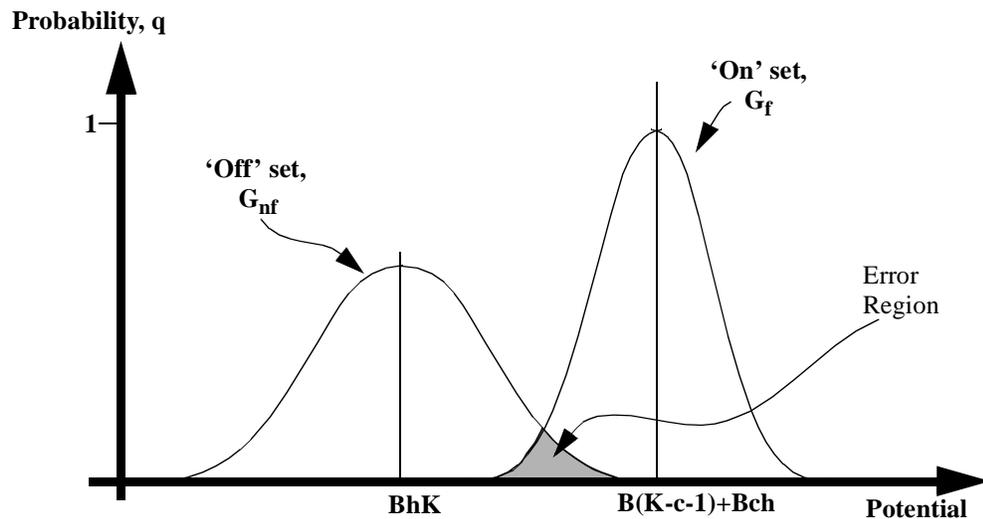


Fig. 7-9 Potential distribution for neurons in a non-learned state

As before, the issue is to calculate the probability that a neuron $j \in G_{nf}$, which does not fire for the target pattern, \bar{p}_0 , in fact has a potential higher than at least one of the set of neurons $i \in G_f$, which is supposed to fire. If this case arises, the network state has a chance to diverge rather than converge on the target memory. By forbidding this case (classifying it as a network error) we prevent divergence of the network state, but by doing so place heavy constraints on the maximum number of uncorrelated patterns that can be stored. These limits will be explored next.

For a neuron, x , which is meant to fire ($x \in G_f$) the probability density function is shown on the right hand side in the figure. For c corrupted bits, the mean, μ , and variance, σ^2 , of the potential of a neuron from this set are:

$$\begin{aligned}\mu(x|x \in G_f) &= B(K-c-1) + Bch \\ \sigma^2(x|x \in G_f) &= B^2ch(1-h)\end{aligned}$$

For the potential, the first term is due to contributions from the $K-c-1$ firing members of \mathbf{G}_f which by definition always make a connection of strength B with every other member. Note that this is the worst case potential for a member of \mathbf{G}_f and strictly is only valid for a *firing* member of the set, which is not allowed to make a connection with itself. In contrast, a member of \mathbf{G}_f which is not firing (but should be) would see $(K-c)$ firing neurons from \mathbf{G}_f , not $(K-c-1)$.

The second term in the mean potential is due to contributions from the firing neurons which are part of the set \mathbf{G}_{nf} . These neurons are firing erroneously in the context of the recall of golden vector p_0 and any potential they contribute to members of \mathbf{G}_f is due to random correlations with other stored patterns. The mean contribution from each such connection is Bh , and there are c possible sources, giving a total mean contribution of Bch .

The variance term for the set \mathbf{G}_f is totally dependent on the random contributions due to other stored patterns. There are no contributions to the variance from the other members of \mathbf{G}_f since these connections are always made (i.e. not random).

Turning now to the gaussian for the members of set \mathbf{G}_{nf} , the mean and variance are unchanged from the previous case where the network began in a state corresponding to a stored memory. Thus:

$$\begin{aligned}\mu(x|x \in G_{nf}) &= BhK \\ \sigma^2(x|x \in G_{nf}) &= B^2Kh(1-h)\end{aligned}$$

We define the potential of a neuron $i \in G_f$ at time t as $U_i(t)$. Its potential is drawn at random from the distribution for G_f . Similarly, the potential of a neuron $j \in G_{nf}$ at time t is defined as $U_j(t)$, drawn at random from the distribution for G_{nf} . The difference between these quantities we define as $U_{diff}(t)$. From elementary statistical theory (Lapin, 1990), we know that the difference between two random variables is also a random variable for which the mean is the difference between the means of the individual variables and the variance is the sum of the individual variances. Thus $U_{diff}(t)$ can be described by a gaussian with mean and variance:

$$\begin{aligned}E(U_{diff}(t)) &= B(K-c-1) + Bch - BKh \\ &= B(K(1-h) - c(1+h) - 1) \\ \sigma^2(U_{diff}(t)) &= B^2ch(1-h) + B^2hK(1-h) \\ &= B^2h(1-h)(c+K)\end{aligned}$$

Neuron j will fire erroneously if its potential is greater than that of neuron i , i.e. $U_j(t) > U_i(t)$, a state which is characterised by $U_{\text{diff}}(t) < 0$. The probability that this occurs is simply the area of the distribution for $U_{\text{diff}}(t)$ which is less than zero, shown below. The area sought is found using the error function, $\text{erf}(Z)$. As before, we translate this distribution into the normal distribution for which this error function has been pre-calculated.

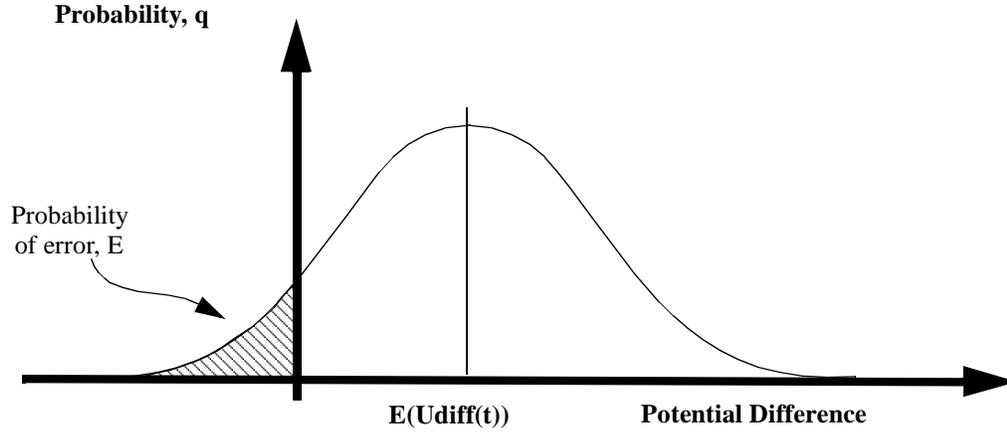


Fig. 7-10 Distribution of potential difference between firing and non-firing neurons

Using the mean and variance for $U_{\text{diff}}(t)$, the formula for the normal deviate, z , becomes

$$z = \frac{x - \mu}{\sigma}$$

$$= \frac{x - B(K(1-h) - c(1+h) - 1)}{\sqrt{B^2 h(1-h)(c+K)}}$$

so that the probability that $x < 0$, as given by the error function, is

$$Pr(x < 0) = \text{erf}\left(\frac{0 - (K(1-h) - c(1+h) - 1)}{\sqrt{h(1-h)(c+K)}}\right)$$

This is the probability for a single erroneous transition between neurons i and j . There are K such neurons in G_f and $N-K$ neurons in G_{nf} . Thus the number of independent events which must be considered is $K(N-K)$. The expected number of errors due to transitions at time t is therefore:

$$N_{\text{err}}(t) = K(N-K) \cdot Pr(x < 0)$$

The graphs overleaf in figure 5-11, for the probability of an error for given neurons i and j , $Pr[x < 0]$ and for the expected number of errors for the whole region, $N_{\text{err}}(t)$ against the number of stored patterns, Z , for several values of error, c . In all cases, a 10-from-200 code is used.

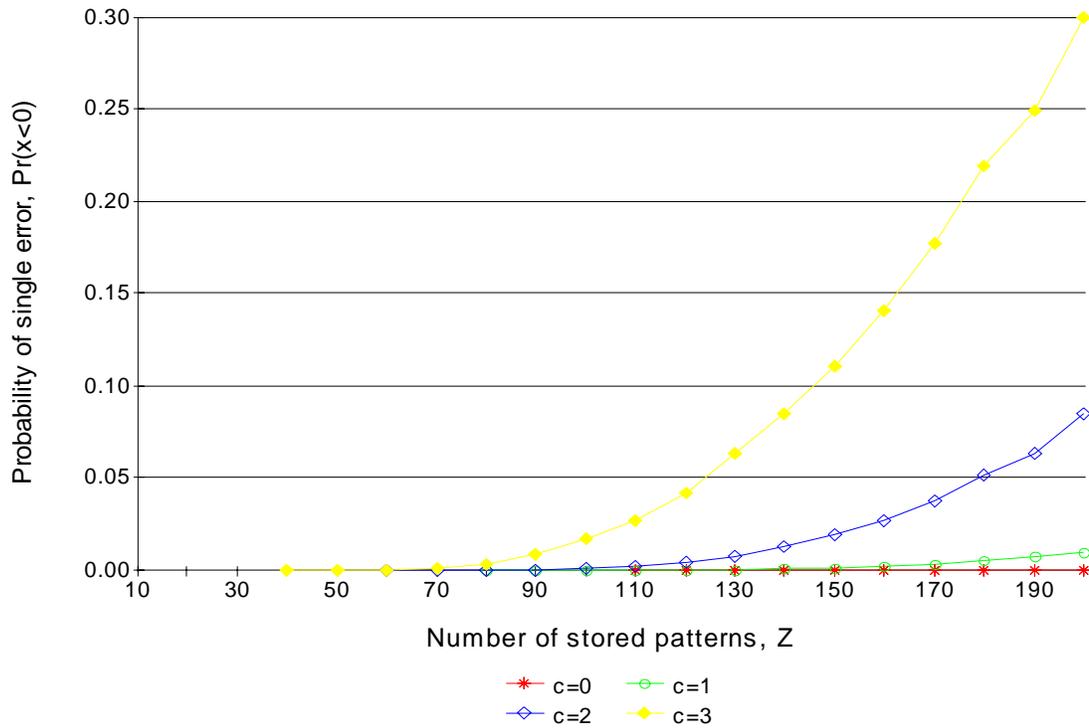


Fig. 7-11 Graph of error probability for two neurons

The total number of errors is the most useful indicator and is shown in the graph overleaf. When $N_{err} > c$, the pattern will diverge at the next update since the value of N_{err} at time t will become the value of c at time $t+1$. For $N_{err} < c$, the expected behaviour is that the network should slowly converge towards zero errors. For $N_{err} = c$, the number of errors should remain constant. Therefore, we should select the maximum number of patterns to satisfy the criteria of the maximum number of bits that must be corrected for.

From the graph it is clear that the storage capacity of a 10-from-200 network cannot exceed 65 patterns if errors as large as 2 bits are to be corrected. To correct 3-bit errors, only 45 patterns can be stored. Notice also that even for the case when $c=0$ (which corresponds to the network in a state corresponding exactly to a stored memory) there is a non-zero value for N_{err} . This implies that even a stored memory is not guaranteed to be a stable point of the system as the number of stored patterns is increased.

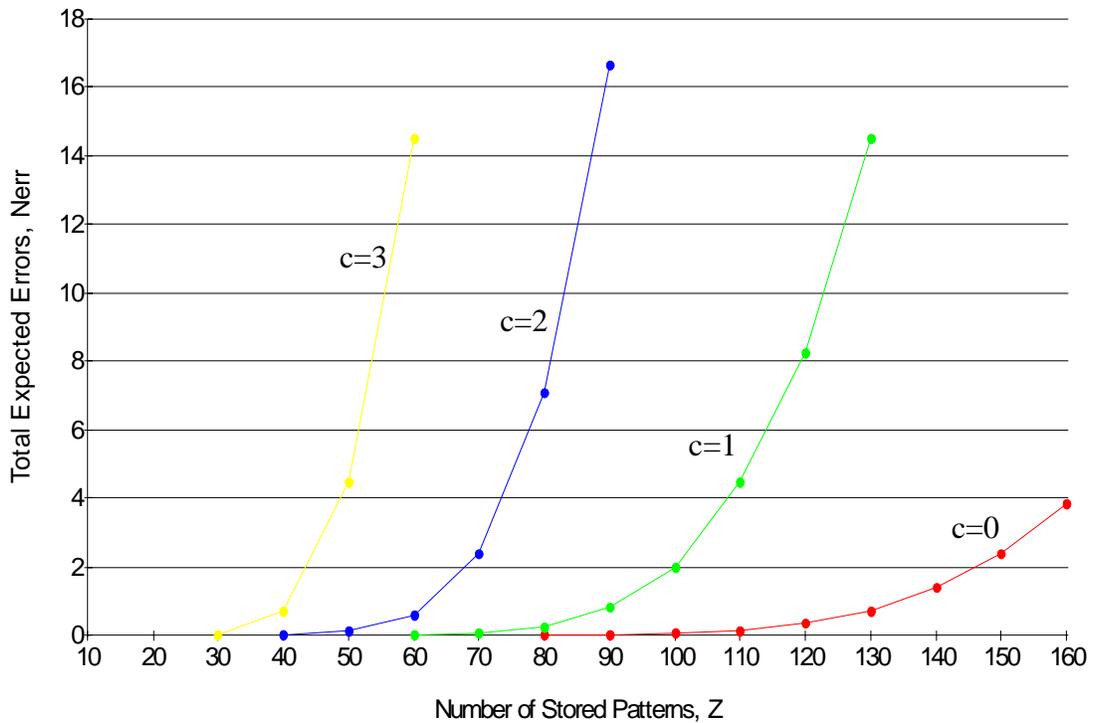


Fig. 7-12 Graphs of the total expected error vs. number of stored patterns.

7.6.3 Storage Capacity

Since the expected number of errors at time t , $Nerr(t)$, is

$$Nerr(t) = K(N - K) \cdot Pr(x < 0)$$

we can define a maximum error per neuron, $Nerr(t)/N$ as say $K/4$. This would permit $0.25K$ error in total, or one quarter of the '1's misplaced. For this limit to apply, $Pr(x < 0)$ is restricted thus:

$$\frac{Nerr(t)}{N} > \frac{K}{N}(N - K) \cdot Pr(x < 0)$$

$$\frac{K}{4} > \frac{K}{N}(N - K) \cdot Pr(x < 0)$$

$$Pr(x < 0) < \frac{N}{4(N - K)}$$

The value of $Pr(x < 0)$ depends on the level of corruption, c . Taking $c = K/4$ gives:

$$Pr(x < 0) = erf(-v)$$

$$\text{where } v = \frac{K(1 - h) - 0.25K(1 + h) - 1}{\sqrt{h(1 - h)(1.25K)}}$$

Using the approximation to the error function,

$$erf(y) \approx 1 - \frac{1}{\sqrt{\pi}y} e^{-y^2}$$

for $y \rightarrow \infty$, together with the fact that $\text{erf}(-y)=1-\text{erf}(y)$ gives:

$$\frac{N}{4(N-K)} > \frac{1}{\sqrt{\pi v}} e^{-v^2}$$

with v defined as before. This can be solved numerically for h and hence Z , the maximum number of stored patterns. The graph below shows the relationship between Z and K for a network with $N = 200$ neurons.

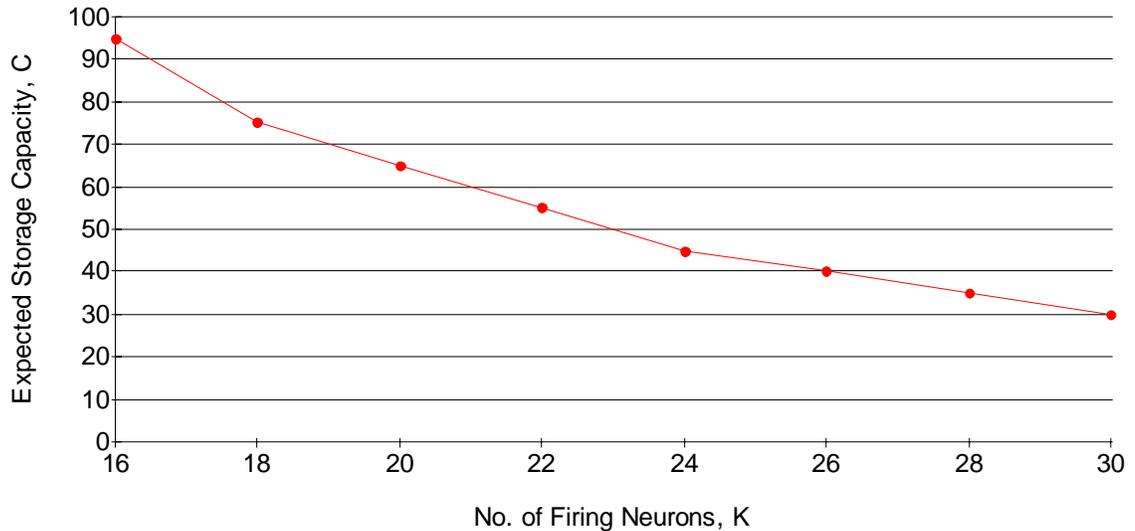


Fig. 7-13 Graph: expected max. stored patterns vs. # of firing neurons for N=200

It is important to understand what assumptions underlie this capacity measure. It is not a convergence analysis of the network, which would require more in depth consideration of the statistical fluctuations of the correlations between patterns. To do this, statistical mechanics would probably be a useful approach, but this is beyond the scope of this work.

Instead, what we have assumed is that the pattern is considered stable if the number of erroneous bits does not increase. Since we have taken the maximum number of erroneous bits as $0.25K$, we consider a pattern to be correctly retrieved if, beginning with $0.25K$ bits corrupted, the expected number of errors is unchanged from one cycle to the next. This analysis does not take individual fluctuations into account and is therefore accurate only in the limit as N tends to infinity.

However, the simulations which will be reported on section 7.8 in take into account the convergence of the network over multiple cycles rather than just the number of errors after a single cycle. This is a more important metric of network performance since it is the convergence properties that are really defining the usefulness of the network in a real system.

Comparison between the theoretically derived results and simulations is possible, but only when we consider those cases for which the theory can provide valid conclusions: this is limited to the cases where the initial level of corruption of

the pattern is equal to the maximum number of erroneous bits, in this case 0.25K bits.

Having understood the limitations of the analysis, the final step is to calculate the total information stored in the network as the product of the number of stored patterns and the information content of each pattern. The graph below shows the total information content of a network of 200 neurons for various values of K:

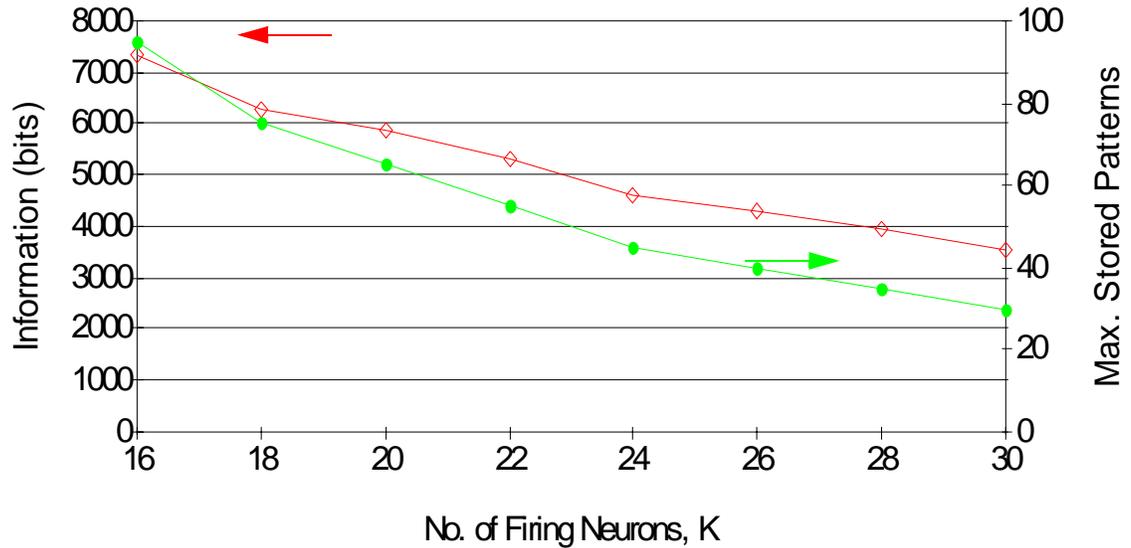


Fig. 7-14 Total information stored for perfect recall in an N=200 network

In the graph, the green line (corresponding to the right Y-axis) is the maximum number of patterns, Z_{\max} , stored by the network before errors occur on recall, while the red line (corresponding to the left Y-axis) is the total quantity of information stored in the network, $I_{\text{total}} = I_{\max} \times Z_{\max}$. From the graph it is clear that the highest total storage is achieved for the lowest K that was considered, $K = 16$.

We know from the basic theory of K-from-N codes that the maximum will occur when $K = \log_2 N$, or just over 7 bits for a network of $N=200$ neurons (see review of Nadal & Toulouse, 1990, beginning on page 51). As discussed in the last chapter, however, the optimal value of K from the point of view of capacity does not necessarily lead to the greatest tolerance to distortion of the input pattern as quantified by its basin of attraction. Simulations to justify this will be presented later (section 7.8.2, page 205).

We note in passing that for $K=0$ (with all neurons silent for each pattern) the maximum number of stored patterns is infinite since each new pattern contributes zero to the noise component during recall. However, the number of legal vectors is only one (the zero vector) and furthermore the information contained in each pattern is zero bits, so this result is not very useful, practically speaking.

Returning to our discussion of the graph, it is clear that the change in total network storage capacity with increasing K is dominated by the falling number of stored patterns. Why should this be so? We know from the properties of K -from- N codes, as outlined in section 6.3.2, page 159, that the maximum information content of each pattern is

$$I_{max} = -\log \frac{(N-K)! K!}{N!} \text{ bits}$$

Using the approximation that $\ln x! = x \ln x - x$ for large x and ignoring the factor of $\log_2 e$ required for a change of base¹, we can approximate I_{max} as:

$$\begin{aligned} I_{max} &\approx -(N-K)\log(N-K) + (N-K) - K\log K + K + N\log N - N \\ &\approx N\log N - (N-K)\log(N-K) \end{aligned}$$

for large N . The information per pattern is a function of K , rising linearly with the factor $-(N-K)$ but falling with the factor $\log(N-K)$. The result is a less than linear rise for low K .

Turning to the number of stored patterns, we see from figure 7-10 that the probability of error increases more than linearly with the potential difference at the tail end of the distribution. The potential difference itself is roughly proportional to K and (if we consider uncorrupted input so that $c=0$), we can reasonably take the number of error bits during recall to be more than linear in K , for low K .

Finally, we note that the storage capacity (which is inversely proportional to the mean number of errors in the limit of large N) falls off more steeply than a linear decent in K . This term dominates the total storage capacity, as shown in figure 7-14, explaining in qualitative terms why it decreases with K .

Overall this analysis has shown that for a network storing K -from- N patterns using the learning and recall procedures outlined earlier in this chapter, the use of a low firing level, K , will produce a higher level of total information storage than that of a network in which K approaches $0.5N$.

Comparing the overall scheme to the learning algorithms presented in the literature survey of chapter three (in particular the review by Casasent & Telfer), we note that the storage capacity seems to be less than the $2N$ of the Ho-Kashyap algorithm. This is not surprising, since the algorithm we use here is essentially based on simple correlated firing between pre- and post-synaptic neurons.

However, the Ho-Kashyap algorithm was applied to purely feedforward memories, not the recurrent type used here. The performance of recurrent networks, in general, was shown to be poorer than feedforward networks although the recurrent nature of the algorithm allowed error correction to be performed at no extra cost. Furthermore, the Ho-Kashyap algorithm was not suitable for one-shot

1. Alternatively we could have converted the quantity in question from bits to natural bits or *nats*. However, we are only interested in the general dependence of I_{max} on K here, so we omit this added complexity.

learning in a dynamic environment, requiring full knowledge of all patterns in advance and even forcing the output patterns to be modified to facilitate storage. The simple scheme outlined here has none of these disadvantages.

7.7 Energy Landscape

In section 5.5 it was shown that the asynchronous neuron update rule would cause the energy of the system, defined by the equation

$$E = -\frac{1}{2N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} O_i O_j W_{ij} + \frac{1}{2N} \sum_{i=0}^{N-1} T_i O_i$$

to change according to $E_{\text{diff}} < W_{BA}/N$ where W_{BA} is the value of the weight between the neuron which is ceasing firing and the neuron which is beginning. In the limit as N tends to infinity (i.e. a very large network) the proportion of the energy change contribution by this positive term tends to zero; the energy is reduced monotonically towards zero, in the limit.

It is well known from analysis of the Hopfield network that local minima in the energy landscape can be a cause of convergence problems which are often avoided by using noise (Amit, 1989). Is the same true in this network? Consider a network in a state A, corresponding to one stored pattern. The energy in this state is a minimum, and ignoring the threshold related term, is equal to $-K(K-1)/N$.

Now consider the path across the energy landscape to a state B, also corresponding to a stored pattern and thus also with energy $-K(K-1)/N$. Each step from state A to state B equates to a single '1' in the state vector shifted to a new position to take it from the vector for state A towards that of state B. Assume for this example that A and B have no '1's in common. The graph below shows an example landscape.

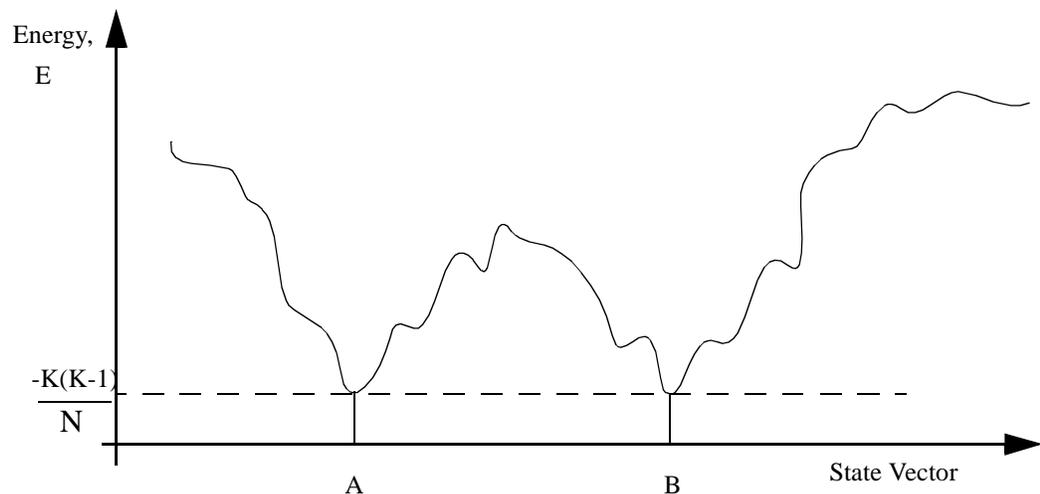


Fig. 7-15 An energy landscape for the simple network.

Since the changes in the state vector always move the '1's to positions which correspond to '1's in the vector for state B we know that the energy will reach a new minimum in at most K steps. After n steps from state A, K-n '1's from the original pattern remain. With the (n+1)th step, the energy increases by $B^2(K-n)/N$ due to the loss of support from '1's which remain in the pattern for state A, but decreases by an amount B^2n/N due to a gain in support from '1's which are already in place for the pattern of state B. Random contributions to the energy also occur, due to cross talk between the K-n firing neurons which remain for state A and the n neurons which are now firing for state B. Each possible contribution will only be realised if a connection between the two neurons had been made for another pattern, which is a random quantity.

Thus the energy change between the two states is not defined by two smooth basins but by a complex set of hills which depend on the other stored patterns. Thus, just as in the Hopfield network case the landscape is littered with local minima and convergence to a non-stored memory pattern is possible.

7.8 Simulation of the Simple Network

To verify the theory presented in this chapter, simulations were performed on a variety of networks using different parameter values. The basic idea was to measure the number of patterns which produced errors during recall, even when the initial pattern presented to the network itself contained errors.

It is important to note that for each network size and configuration only a single run was made. The reason for this is that the run time for each network configuration was extremely long, ranging from six hours for the fastest to over a week for the slowest, running on an IBM compatible PC at 100MHz. As a result it was not possible to produce error bars for the sample data points and all conclusions from the simulations must be drawn tentatively. Future work should include more simulation runs of these networks leading to the production of a box plot illustrating confidence levels for the spread of parameter values.

The scope of the simulations was intended to include a range of different network sizes and configurations. The analysis provided results whose validity increases with N and are strictly valid only in the limit as N tends to infinity. Although we would like to include the large N limit in simulations the simulation time beyond N=300 was deemed to be too great. To cover a set of different network sizes, networks consisting of N=100, 200 and 300 neurons were trained.

For each network, the number of firing neurons, K, was varied. For example, in the N=100 case, values of K from 8 to 20 in steps of 4 were used. The step size was chosen to allow the integer noise margin $M=0.25K$ to be defined.

For each N/K pairing, the network learned a set of 150 patterns, in groups of 10. After each group of 10 was learned, the network was tested on the whole learned set to see if it could recall them correctly. This gradual loading of the net-

work was intended to allow us to establish the rate at which errors creep into the recall process with each new learning event. Ideally, we would do this one pattern at a time. The selection of ten patterns per group is a compromise that allows us to cover the entire set of 150 patterns in a practical length of time.

Recall itself consisted of presenting each pattern with a number of bits, c , corrupted (i.e. shifted to other bit positions) at random. The number of corrupted bits varied over the range from zero to one half of the total firing neurons, K , in the pattern. By testing the network using a single pattern which begins at different levels of corruption it allows us to establish the width of the basin of attraction for each pattern. This corresponds to the parameter e , the radius of the initial target circle, in section 6.5.3, page 169. At corruption level $c > 0.5K$ we can no longer guarantee that the initial corrupted pattern is closer to its uncorrupted parent than to another stored pattern and so it makes no sense to investigate corruption levels beyond this point.

For each corrupted pattern the network was iterated eight times, in which it was expected to reduce the level of corruption and bring the pattern within the final target circle as defined by the parameter d in section 6.5.3. We need to have swift convergence since the output of the network may be used directly by some other network. Therefore, we allow only two cycles for the state to converge to be within distance d of the target pattern.

To be sure that the pattern will not subsequently diverge, we demand that for many subsequent cycles (six was chosen as a compromise between confidence in convergence and excessive simulation time), the distance of the state vector from the target vector does not increase beyond d . Hence, correct recall was assumed if, from the third iteration onwards, the number of erroneous bits in the pattern did not exceed a defined margin, M . The margin was set at $0.25K$, so that even if one quarter of the firing neurons were erroneous the pattern was considered to be correct. Notice that the margin M (measured in shifted bits from one legal vector to another) is analogous to the distance d (a distance metric between two vectors in the same vector space) defined in section 6.5.3, page 169.

Graphs of the results are included at the end of this chapter. Each graph corresponds to one network size, N , and number of firing neurons, K , and plots the number of patterns which were incorrectly recalled against the number of stored memories. Individual traces on each graph correspond to different values of c , the initial number of corrupted bits in the pattern.

[Note that for the network of size $N=200$ neurons, a set of 300 patterns was used, but for the network of size $N=300$ there were insufficient computing resources available to use more than the base 150 patterns].

There are similarities which are visible across all values of N and K . First, there are few, if any, errors for low loading (a low number of memorised patterns, Z). Even for a significant number of initially corrupted bits, the low-loaded network is able to reach a stable state within the error tolerance band. As more and more patterns are memorised, there comes a point when significant (non-zero)

errors first begins to appear in the recall of the whole pattern set. For a given loading, Z , we note that the onset of failed recall occurs first for the highest levels of initial pattern corruption, c , which in line with intuition since increased initial pattern corruption reduces the signal term for all firing neurons.

As the number of memorised patterns is increased beyond the point where errors first begin to appear there is a fairly rapid transition to a state where most or all patterns are diverging from their initial level of corruption, c . The rate (as a function of Z) at which this occurs increases as the ratio K/N increases. We can see this by comparing the simulations for a fixed value of N and of c . For instance, the graphs of $N = 100$ neurons, with uncorrupted initial patterns ($c=0$). For $K=8$, the first errors begin to appear when $Z = 40$. As Z increases the number of failed recalls increases steadily until total recall failure occurs at around $Z = 110$. If we now consider a higher proportion of firing neurons with (say) $K = 16$, we see that errors first appear at around $Z = 20$ patterns but total recall failure occurs with only 20 more patterns, at $Z = 40$. Why should this be?

From the theory, it appears that the increased sensitivity is due to the greater rate of saturation of the weight matrix. Note that the probability of a connection, h , at a particular loading, Z , is defined in terms of p^2 (where $p=K/N$). From Figure 7-3 on page 181, we see that while it is still low, h is itself an increasingly sensitive function of Z for increasing $p=K/N$. Increasing K/N ratio therefore implies a faster rate of saturation of the matrix with memory load, Z .

Furthermore, from Figure 7-10 on page 193, we see that the expected probability of an error is proportional to the area of the gaussian for $x < 0$ (corresponding to the potential difference between firing and non-firing neurons). This is a function of h : the mean is reduced and the variance is increased for increasing h , both effects leading to increasing error rates as h increases. Thus we see a direct connection between K/N ratio and the rate at which the first appearance of errors leads to memory failure as the memory load, Z , is increased.

Moving on, we note that for almost every network configuration, saturation occurs before the full set of 150 patterns have been learned. This is evident from the convergence of the number of errors with the number of patterns stored, indicating complete failure to recall any stored pattern.

The trend in almost all cases is for the number of errors to increase with loading, Z . This is in line with our expectations since the noise component to the potential of neurons which should not fire is increasing with Z . The only anomaly occurs for $N = 200$, $K = 8$. Here we see that for two of the traces the number of errors decreases at around $Z = 120$. For a single bit of initial corruption, the single pattern that failed to converge at loading $Z=110$ succeeded at $Z=130$. Similarly, for two bits of initial corruption, while there were 14 patterns that failed to converge at $Z=120$, only 11 failed at $Z=130$. This is counter to our expectations, but there are two reasons to explain these effects.

First, it is possible for the random, slow noise that is introduced with each new pattern to work for us as well as against. As noted earlier, we are working with

a low sample size (a single trial for each set of network parameters). Second, the selection of which bits to corrupt in a given pattern need not give the same result from one trial to the next. It is possible that one set of chosen bits might result in failure to converge at $Z=120$ while the two different corrupted bits that were chosen for that pattern at $Z=130$ might have resulted in successful convergence if they had been chosen at $Z=120$. In either case, these are statistical effects due to the low sample size. As noted earlier, with sufficient computing resources we could attempt to reduce these effects by averaging over multiple simulation runs.

Looking at any one graph it is clear that in general, as would be expected, the number of erroneous recalls increases as the degree of corruption of the initial pattern increases. For the characteristics for $c=0$ the total number of patterns which will be stable is always greater than or equal to that of the case $c>0$.

For the networks with $N = 100$, around 40 to 50 patterns could be recalled without error, while for the $N = 200$ that limit was pushed up to around 100. In the $N = 300$ case, the limit is increased slightly but the trend seems to be that the rate of change of network capacity is negative with respect to N . This result follows from the definition of the expected number of errors, N_{err} , defined on page 195:

$$N_{err}(t) = K(N - K) \cdot Pr(x < 0)$$

Since for large N the number of errors is proportional to the number of failed patterns, we see that this number is proportional both to N and to K . Note that as the network size is increased in these simulations we have selected p in order to keep the proportion of firing neurons, p , roughly constant and $K=pN$. Thus, the factor $K(N-K)$ increases more than linearly with N , leading to a proportionally larger fraction of errors for larger N .

This suggests that for increasing N we might consider reducing K , the number of firing neurons. Obviously, this will impact the information stored in each pattern, but this is a trade-off that can be made for each network design.

7.8.1 Comparison of Simulation Results with Theory

Next, we compare the storage capacity obtained through simulation with the expected storage capacity as provided by the analysis. As was noted in section 7.6.3, beginning on page 195, this comparison is not straightforward since the theory only provides information on the change in the number of erroneous bits from one cycle to the other and is, furthermore, only valid in the limit of large N . The theory derived is not a convergence analysis: the network state could begin by diverging but then stabilise at or below the maximum allowed number of erroneous bits.

A full analysis needs to consider the dynamic evolution of the probability distribution for the state vector. Furthermore it must also take into account the effects of finite N , which would lead to fluctuations in both the signal and the noise terms for each neuron as a function of the stored patterns themselves. Such an analysis is probably best undertaken using statistical mechanics and is beyond the scope of this work.

However, if we accept the fact that N is finite, we can compare theory with practise in the one case where the network state begins at the limit of acceptable recall (in this case we have chosen $0.25K$) and the expected number of erroneous bits is unchanged from one cycle to the next. In the limit of large N this corresponds to a stable network state, i.e. correct recall. For the simulations, this case is also on the limit of acceptable recall, permitting comparison between the two.

Thus, for each network configuration (defined by N and K) we can compare the maximum number of patterns that we can store such that when recalling with corruption $c = 0.25K$ bits the number of erroneous bits, N_{err} , is also equal to $c = 0.25K$ bits.

Using this metric, the table below compares the theoretical storage capacity with the simulated result for several configurations. Network sizes of $N = 100$, $N = 200$ and $N = 300$ are included.

Table 7-0 Comparison of simulated vs. theoretical network capacity of the simple network

# of Neurons, N	# of Firing Neurons, K	Max. Corrupted bits, $c = 0.25K$	Theoretical Capacity (# of stored patterns), Z_{thry}	Simulated Capacity (# of stored patterns), Z_{sim}
100	8	2	31	33
100	12	3	19	40
100	16	4	13	15
200	8	2	109	101
200	12	3	69	98
200	16	4	47	90
200	20	5	35	65
300	20	5	77	145

We see from the table that it is for the network of size $N = 100$ that the expected and simulated capacities are the most similar. For $K = 8$ and $K = 16$ the results match almost exactly. For the other networks, the simulations show that the network actually performs progressively better than the theory for increasing N and K . How are we to interpret these results?

If we recall that the condition for acceptable recall in the theoretical case was the non-divergence of the expected number of error bits, it is clear that this criteria is more strict than that imposed on the simulated network. For the simulations, we allowed the network state to both diverge from and converge towards the stored vector provided that it was within $0.25K$ error bits after the first two cycles. It is reasonable to interpret the improved capacity for larger networks to the networks increasing ability to rein in a diverging network state to within the error limit.

The close approximation of theory to the simulations for the $N = 100$ neuron network seem to show that (for low N) initial divergence of the network state cannot be reversed in subsequent cycles. In that case, the metric of constant expected error more accurately reflects the ability of the network to converge to within acceptable error tolerances. In other words, when the expected error increases on

the first cycle of the network (an occurrence that corresponds to a failure for the network analysed in the theory) it is unlikely that subsequent cycles will cause the network to converge, leading to a stronger correlation of theoretical and actual capacity for low N than is seen for larger networks.

7.8.2 Basin of Attraction

As a final part of the network simulations, the size of the basin of attraction around each pattern as a function of the number of stored patterns was measured for a network of $N = 200$ neurons. In this experiment, an active set of $K = 12$ neurons was used. After each pattern was stored, the entire pattern set was tested to measure the width of the basin of attraction (the maximum distance the pattern could be from the true stored memory before it was unable to converge). Convergence corresponded to the network reaching a state in which no more than $0.25K = 3$ bits were incorrect. For each pattern under test, multiple trials were carried out in which the network was presented with the pattern at different levels of initial corruption ('1's moved out of place in the initial vector).

The graph overleaf shows the maximum number of patterns that could be stored before the first convergence failure as a function of the initial number of corrupted bits in the pattern. For $c = 0$ and $c = 1$, no failures occurred in the recall of 100 patterns. The size of the basin is inversely related to the number of stored patterns, which is a result qualitatively similar to that produced by a Hopfield network.

This result is in line with our intuition since increasing errors in the initial vector increase the sensitivity of network convergence to noise. Since that noise component is directly related to the number of stored patterns, we expect that the maximum number of stored patterns will be reduced as we demand greater levels of error correction.

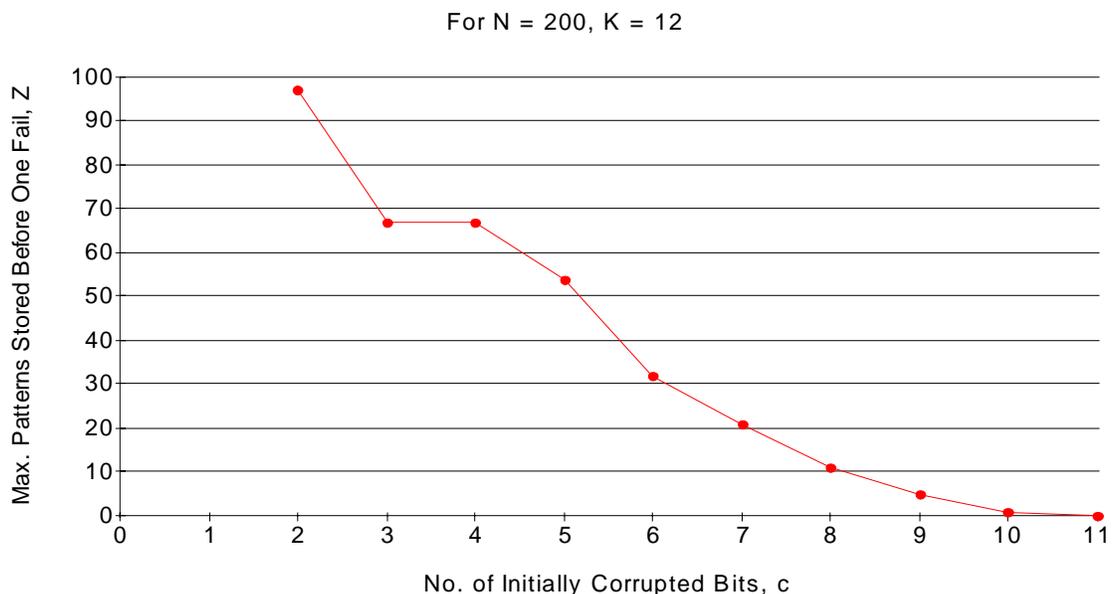


Fig. 7-16 Graph of max patterns stored without error vs. size of basin of attraction

7.9 Discussion

From both the theory and the subsequent simulations it quickly became clear that the selection of p as a constant for various network sizes, N , was not necessarily the best choice. As N increases, the actual number of new connections that are made with each new pattern is increased. These new connections contribute to the signal term in the recall of the new pattern.

But the total amount of signal required to distinguish a single pattern is not necessarily a function of N and it is here that we note a limitation of the fully connected network. A degree of freedom that has been ignored in this analysis is the number of connections that a single neuron can make for a given pattern. There is existing research (for example as reported in Hertz *et al.*, 1991) that shows that for a Hopfield network the capacity can be increased by setting to zero a random subset of the synaptic matrix. This deletion impacts the signal term for each neuron but also reduces the noise. Provided that the selection of synapses to delete is unbiased, the effects can be beneficial.

This idea will be carried into this work in the next chapter. We will restrict the number of connections that each neuron makes with the other firing neurons and investigate the effect that this change has on the network properties.

The other reason for making modifications to the learning algorithm is to try to avoid the problem of increasing energy that was shown to exist for this network in section 7.5, beginning on page 182. The fact that the energy of the system might increase cycle on cycle should be a source of concern. It could lead to the divergence of the state vector away from a stored pattern. The modifications to be made to the learning algorithm should at least address this issue.

7.10 Conclusions

This chapter has presented the basic network which is structurally similar to a Hopfield Network, and described a learning procedure based on a recurrent version of the non-holographic memory model of Willshaw, *et al.*. The dynamics of the network were analysed as a detection of signal in gaussian noise problem.

It was estimated that the number of K -from- N encoded patterns which could be stored in such a network can greatly outnumber the $0.135N$ limit inherent in a Hopfield network, but at a price of lower information content per pattern. Overall, the total information stored by the network is greater for low K than for $K = 0.5$, due to the dominance of the number of stored patterns, Z_{\max} over the information per pattern, I_{\max} .

While it was shown that the network was capable of learning a new pattern with a single presentation, consideration of the energy function shown that, like the Hopfield network, it suffers from local minima due to crosstalk between stored memories which prevents convergence to a stored memory with probability 1.

Furthermore, unlike the Hopfield network it is possible for the energy to increase from one cycle to the next. This effect is due to the fact that two neurons are changing state with each update and the contribution to the energy (positive or negative) of one of them is not known until it has been activated.

The capacity of the network was considered for the case where a recalled pattern was permitted to lie within a margin, $M=0.25$, of the learned vector and still be considered correctly recalled. The capacity, $C(K, N, M)$, of an N -neuron network with this error bound was obtained by solving the equation:

$$\frac{N}{4(N-K)} > \frac{1}{\sqrt{\pi v}} e^{-v^2}$$

$$\text{where } v = \frac{K(1-h) - 0.25K(1+h) - 1}{\sqrt{h(1-h)(1.25K)}}$$

For a network of 200 neurons, the expected capacity was shown to be between 40 and 95 patterns for values of K between 16 and 30. While this is a larger figure than that of an equally sized Hopfield network it is still not sufficient to be used as a building block for the architecture. This capacity analysis was based on the constancy of the number of errors from one cycle to the next and did not constitute a full convergence analysis of the network.

Full convergence simulations of the basic network for a range of values for the number of neurons, N , and the number of firing neurons, K , showed that the network does indeed exhibit the expected properties and that the measured capacity of the network in terms of the maximum number of patterns that could be stored without recall error was close to the estimated value, as a function of K .

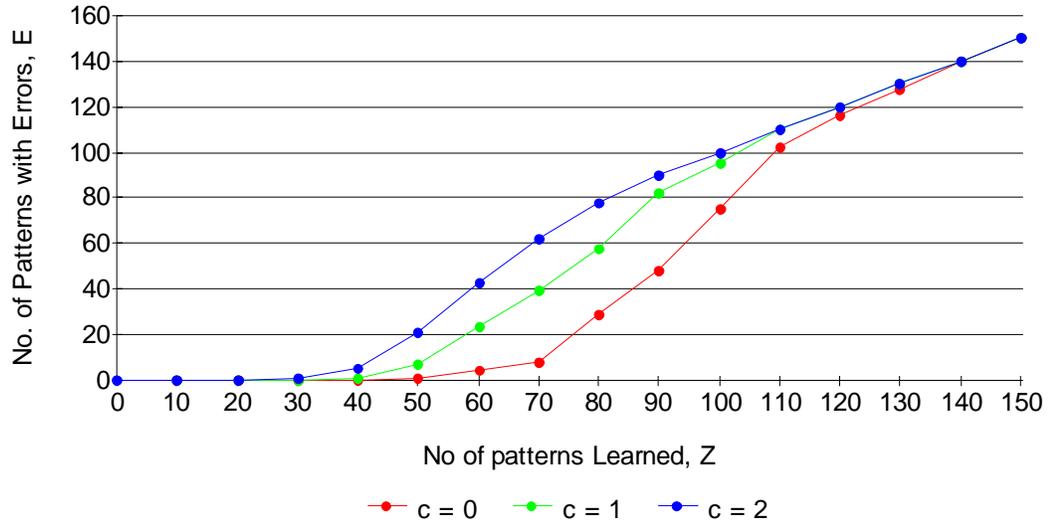
It was asserted that the full connectivity of the firing neurons led to premature saturation of the network due to an excessive signal term for firing neurons. It was suggested that a controlled reduction of this signal term, by reducing the number of connections made for each new pattern stored, might improve the storage capacity of the network by also reducing the noise term in the potential of every neuron. This possibility is explored in the next chapter.

In the next few chapters, several possible avenues of investigation are presented which seek to augment the capabilities of the simple network and to overcome the problems discussed here.

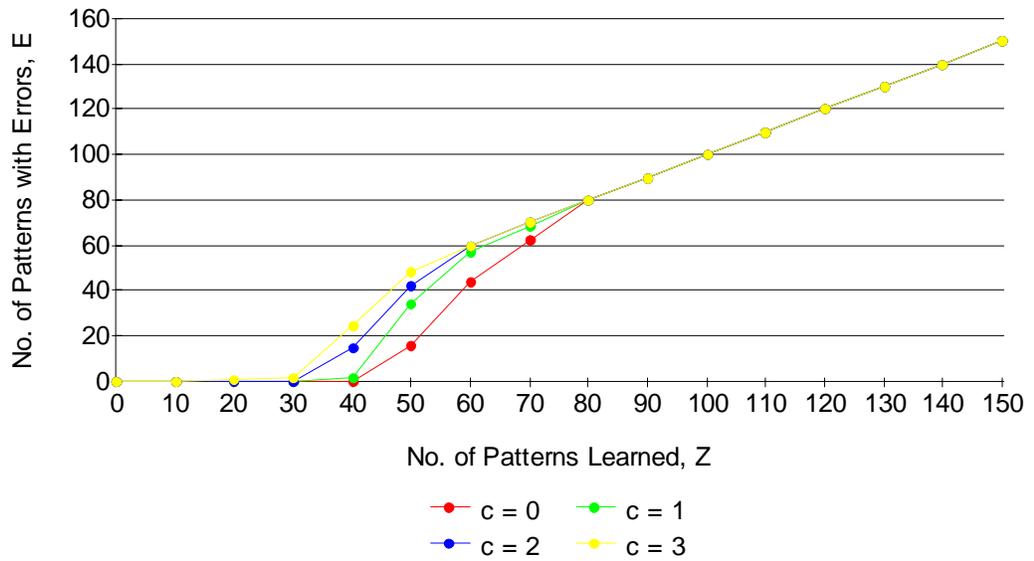
7.11 Appendix: Graphs from Simulations

This appendix presents simulation results for networks of differing N and K .

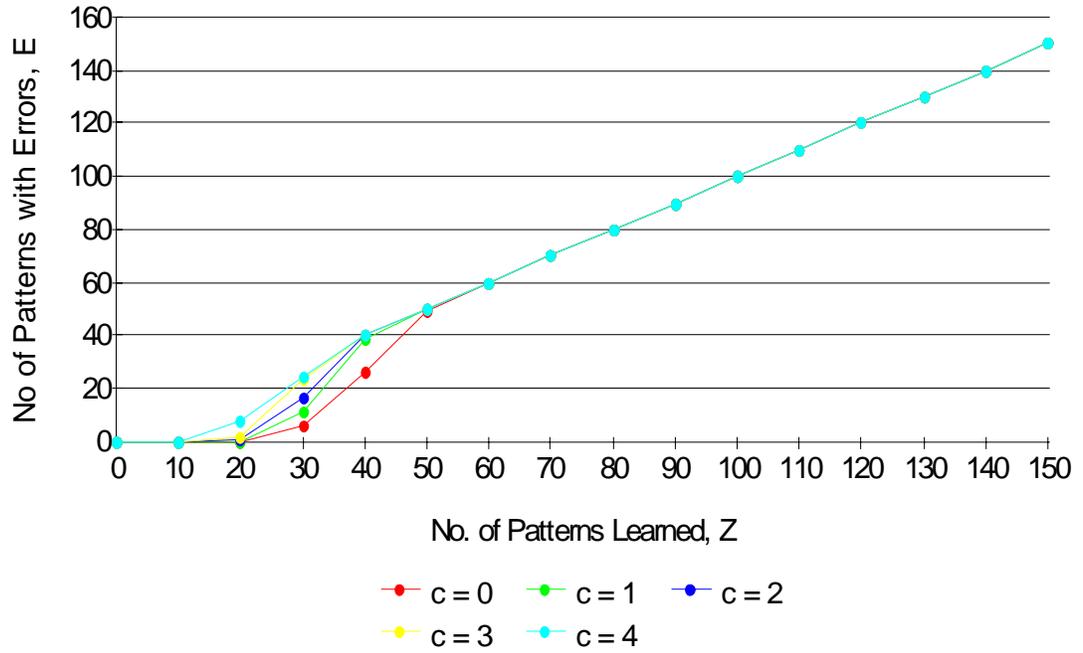
For $N = 100$, $K = 8$



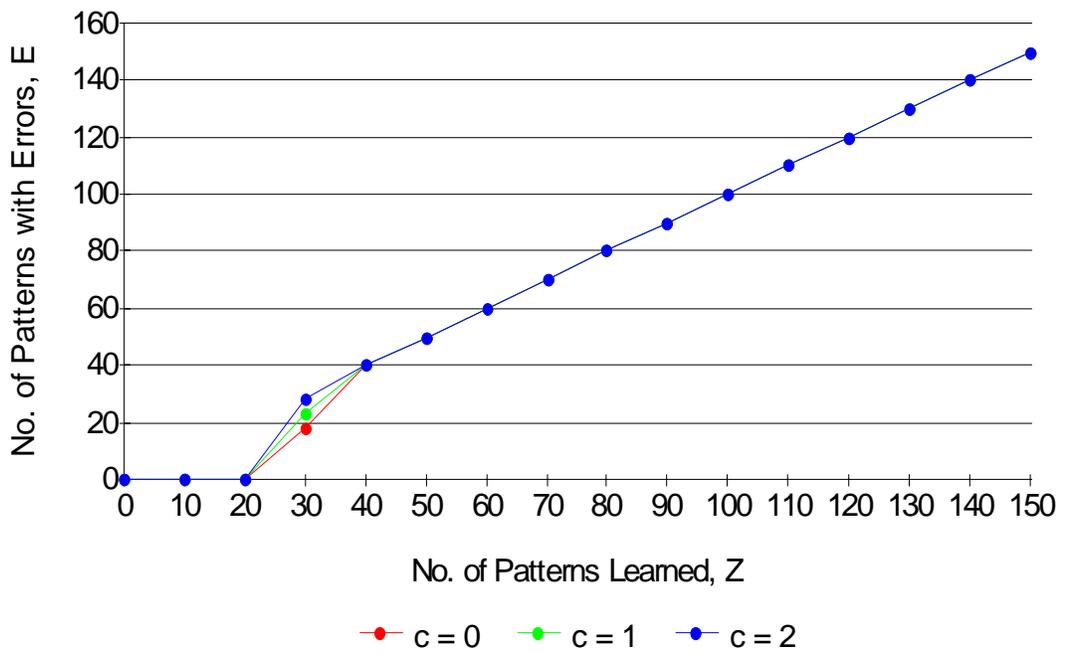
For $N = 100$, $K = 12$



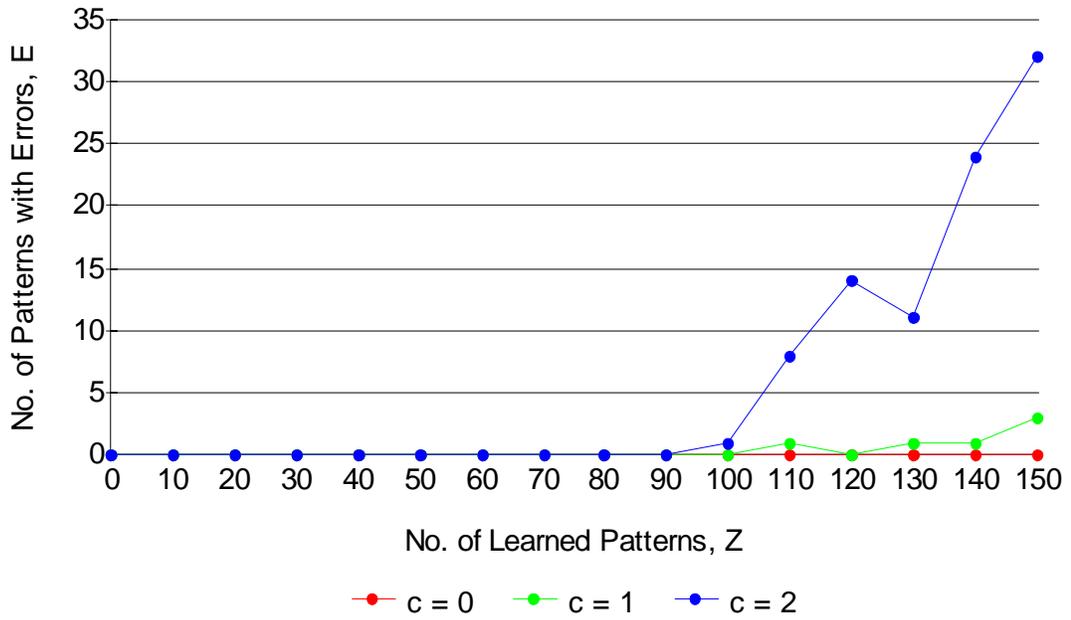
For $N = 100, K = 16$



For $N = 100, K = 20$

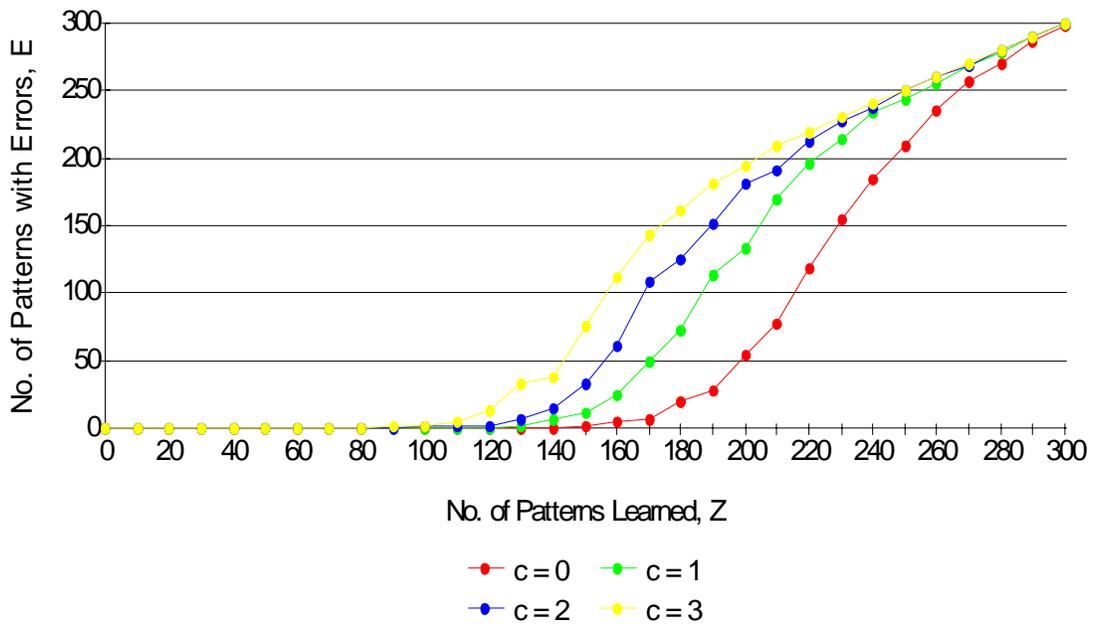


For $N = 200$, $K = 8$

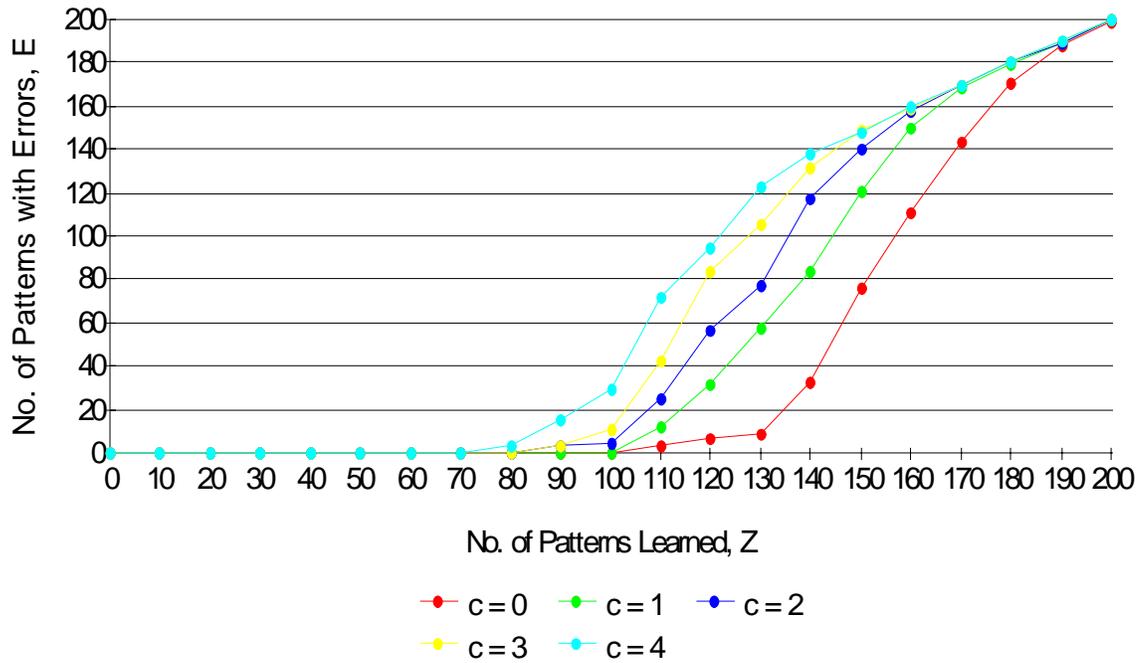


For discussion of kinks for $c=1$ and $c=2$ in above graph, see section 7.8, page 200.

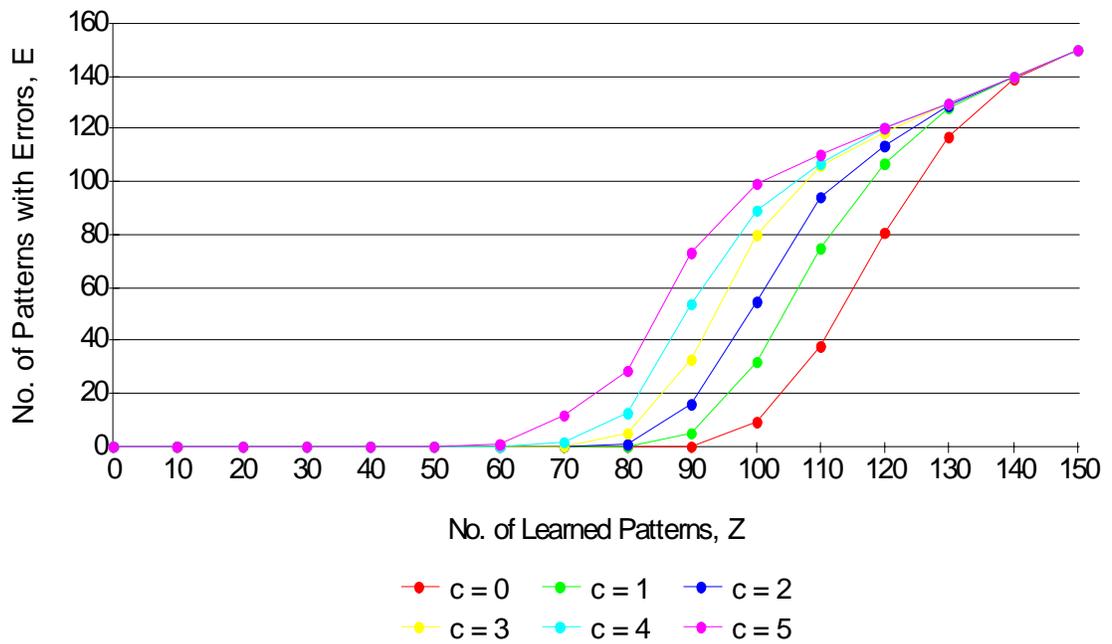
For $N = 200$, $K = 12$



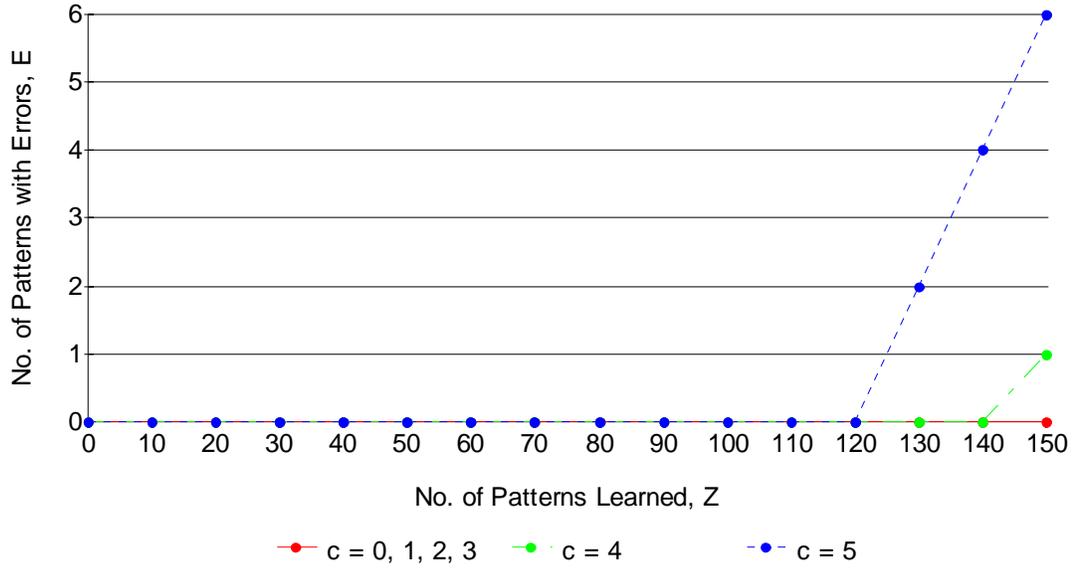
For $N = 200$, $K = 16$



For $N = 200$, $K = 20$

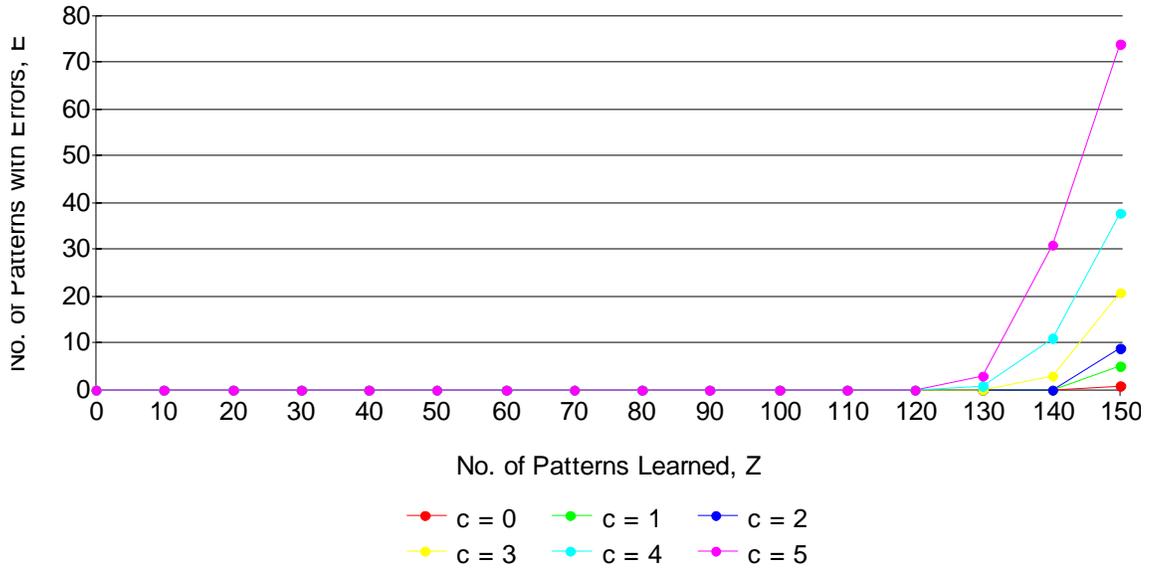


For $N = 300, K = 20$

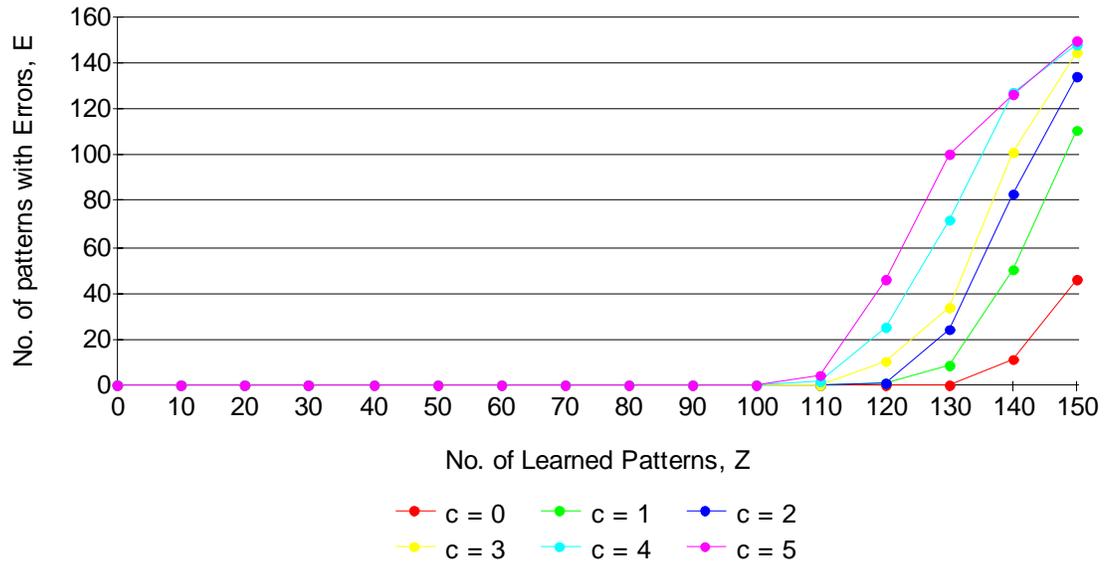


Note: in above graph, $c = 0, 1$ and 2 share same trace.

For $N = 300, K = 24$



For $N = 300$, $K = 28$



8.0 Introduction

The analysis presented thus far has assumed that the stored patterns in the network are fixed points in phase space and that the processing of an input vector consists of applying the vector to the network and continually updating the outputs until no change in the output vector of the network occurs from one timestep to the next. According to this view, when the network is in a stable state, the output vector should correspond to a previously stored pattern. The stability of each exemplar vector was assured by using a symmetrical weight matrix.

Simulations detailed in chapter seven revealed that the memory capacity of the simple network was $O(N)$ where N is the number of neurons. This is similar to the capacity of the range of associative memories as presented in the review by Casasent & Telfer (see section 3.1.2, beginning on page 48). Strangely, for those working in the field of associative memories, being able to store a total number of independent patterns equal to the number of neurons appears to be acceptable. There is perhaps a tacit assumption that if this property can be extended to a brain like machines with billions of neurons then this will be sufficient for real-world applications. We argue that this view of the brain, as a slate in which reams of separate memories are stored like books on a shelf, is one that should be put aside if we intend to make progress with neural networks as reliable tools for mass data storage.

We note the different mindset that brought about the development of the multi-layered perceptron: here the mappings from a set of input patterns to a set of output patterns is not only memorised but the underlying structure of the patterns set is modelled. This is intended to facilitate generalisation from previously unseen patterns and, when the modelling has been performed successfully, to correctly predict the associated output pattern.

Another way to look at this property is that it gives the network the appearance of storing far more patterns than those that have been learned explicitly. This process, we would like to capture for the network we are developing here. Pursuit of this idea is left until chapter ten, when we introduce ‘learning hierarchies’.

An additional problem of the simple network was that local minima are present which can prevent convergence to a stored pattern. In this respect the network is similar to the Hopfield network. If there is a significant chance that the network will not converge, or will converge to an incorrect output, then such a network is not suitable as the building block of a larger system which may contain hundreds or thousands of such modules. Thus, more reliable convergence is necessary.

Turning to the control aspects discussed briefly in the last chapter, it was suggested that a flexible and reliable means of making transitions from one stored memory to another is a basic requirement of the neural building block. How this might be achieved using a learning strategy which is only concerned with stabilising individual patterns without regard to the others is not clear. A better strategy is one in which stability and transition use exactly the same mechanisms. The use of 'dynamic patterns' is intended to achieve that end.

This chapter presents the development of dynamic patterns in the network, which unfortunately renders obsolete a lot of the work already presented and necessitates a re-analysis of many of the properties already studied. It will be shown that the use of dynamic rather than static patterns goes some way toward overcoming the problems of memory capacity, flexible control and reliability of recall.

The exposition begins with the definition of dynamic patterns in the context of the work. Next, possible alternative implementations are presented together with a critique on their merits and limitations. A series of case studies with simulation results follows, before a final discussion on the remaining limitations.

8.1 The Idea Behind Dynamic Patterns

In a dynamic pattern approach, not all of the firing set for a given pattern is active at once. Instead, this set is divided into subsets, each subset firing in turn. Once all of the subsets have fired, the cycle begins again. Thus, a single pattern has both a spatial and a temporal component. In state space, the fixed point attractor of the existing static implementation is replaced by a limit cycle in the dynamic approach. State space schematics of each are shown below.

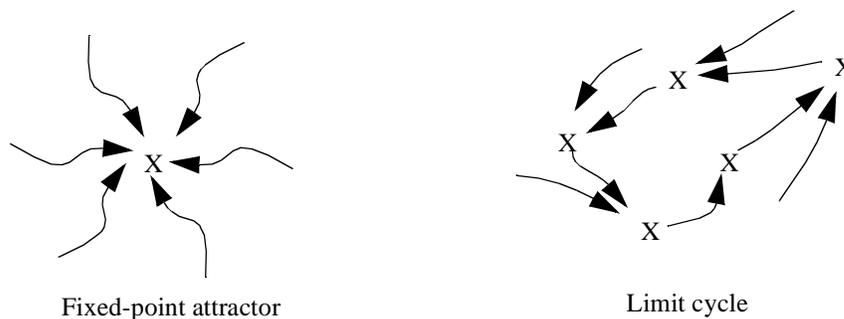


Fig. 8-0 State flow for a fixed-point attractor and a limit cycle.

In figure 6-0, an 'X' marks each state of the network. For the fixed-point attractor, the system in state X remains there indefinitely. In the case of a limit cycle, the state changes on every update of the network but repeats a sequence of states indefinitely (the period is five in this example).

At each timestep, the network must essentially perform a pattern recognition based on the network state that existed in the recent past; how recent depends on the connections made during learning. A stable pattern corresponds to a set of states which would repeat indefinitely in the absence of any change in external stimuli. Changes from one pattern to another correspond to transitions from one cycle of states to a different cycle of states, prompted by external stimuli.

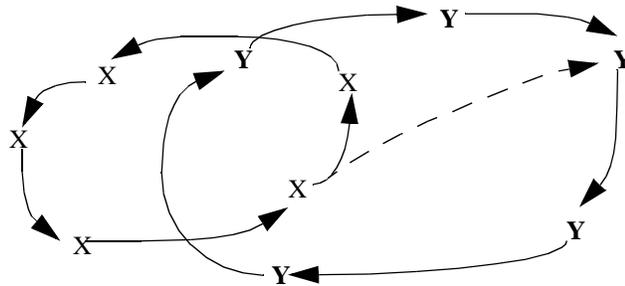


Fig. 8-1 Transition between two dynamic patterns.

8.1.1 Desired Benefits

Clearly, moving to a dynamic pattern complicates the network updating and the analysis of the system. What benefits are expected by making such a change? As discussed in the previous chapter, the simple network in which static patterns are stored suffers from local minima in points of the state space which do not correspond to the stored vectors.

To some extent the local minima problem occurs because of neglect in the specification. The user, by virtue of the learning procedure, demands only that the points given as vectors to learn be global minima of the energy function but there is nothing in that specification to control the energy at any other point. That fact that it corresponds to various combinations of the input vectors should therefore not be surprising.

In contrast, the dynamic pattern approach seeks to control the evolution of the system not just around stored vectors but also during the transition from one to another. It does this by making no basic distinction between a state corresponding to a stable, recalled pattern and an unstable, transitional state. In both cases the state is changing at each timestep, but for the stable pattern the state sequence repeats itself indefinitely.

The goal with the dynamic approach is to be able to program transitions into the network by making minor modifications to the mappings at points around the loop to cause the pattern to exit the loop and form a new stable orbit based around a different stored memory.

8.2 Information Stored in Each Pattern

In chapter six the maximum information content of a single vector using a K-from-N encoding was shown to be

$$\begin{aligned}
 I_{max-KfromN}(K, N) &= -\sum p(x) \cdot \log p(x) \\
 &= \log \frac{(N-K)! K!}{N!} \text{ bits}
 \end{aligned}$$

For the dynamic pattern, the K firing neurons are divided into L subsets, each consisting of D neurons, such that $D = \frac{K}{L}$. At any given time, t, only D neurons now fire. Over L time steps, each subset of D fires once so that during a full cycle of L steps all of the K firing neurons from the original pattern have fired exactly once.

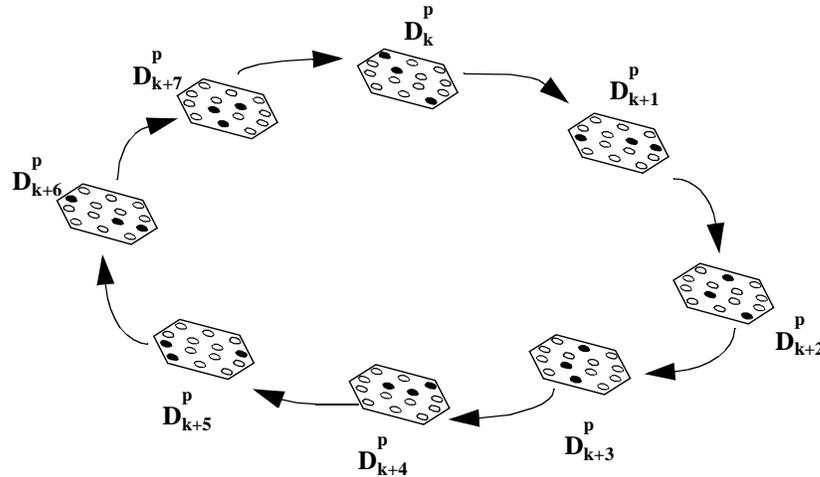


Fig. 8-2 One dynamic pattern cycle, made up of L subsets, describes pattern p.

The maximum information content of a single D-from-N subset of the original pattern is

$$I_{max-DfromN}(D, N) = \log \frac{(N-D)! D!}{N!} \text{ bits}$$

Since there are L such subsets, the total information is L times this quantity. In addition, the ordering of the L subsets is a source of extra information. There are L! possible orders, which require $\log(L!)$ extra bits to specify assuming that all orderings are equally likely. By substituting for D in terms of total number of firing neurons, K, and the number of steps, L, we obtain the maximum total information content of the dynamic pattern:

$$I_{max-KfromN}^{Lsubsets}(K, L, N) = L \log \frac{\left(N - \frac{K}{L}\right)! \left(\frac{K}{L}\right)!}{N!} + \log L! \text{ bits}$$

[In fact, this is a slight overestimate, since the D-from-N subsets are not independent. They are not permitted to overlap, so the nth subset is really a D-from-[N-(n-1)D] code, reducing the true quantity of information that the vector can contain. For $N \gg K$ the error is small and so will be ignored.]

The graph below compares the information content of a straight K-from-N code with that of a dynamic D-from-K-from-N code for various values of L. In all cases, N was set to 200.

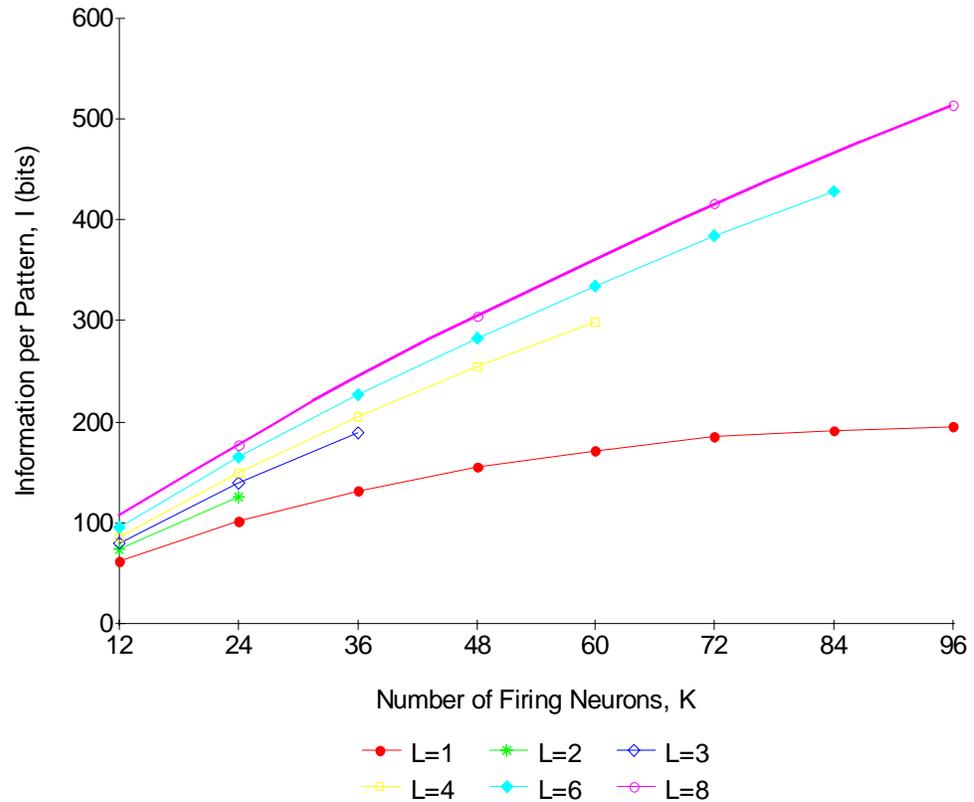


Fig. 8-3 Graph of information content for one pattern vs. number of firing neurons.

The baseline case, $L=1$, corresponds to the non-dynamic pattern in which all K neurons fire simultaneously. For higher values of L , the number of concurrently firing neurons, D , is reduced such that K is held constant. Note that for $L=8$, not all points are defined since $D(=K/L)$ must be an integer.

From the graph it can be seen that the efficiency of the pattern increases as L increases; the information content at $L=4$ is greater than that at $L=2$ for any K . However, the rate at which the information increases with L is itself decreasing, a fact which manifests itself in the way the lines are converging with increasing L . Finally, note that the information has been increased by a factor which depends on K . Multiplication factors as high as four can be achieved for $K=96$.

The drawback of the dynamic approach is that the pattern takes L cycles to reproduce itself. In communication theory, dynamic patterns can be viewed as time domain multiplexing of the information. The total 'bandwidth' required to transmit

the pattern to other networks is effectively L times the bandwidth in the static case. Thus, the information transmitted per cycle is given by:

$$I_{per-cycle}^{Lsubsets}(K, L, N) = \log \frac{\left(N - \frac{K}{L}\right)! \left(\frac{K}{L}\right)!}{N!} + \frac{1}{L} \log L! \text{ bits}$$

In conclusion: one benefit of dynamic patterns is the increase in the total amount of information stored in a pattern, but only by a maximum factor of four for $L < 9$. There are disadvantages to such a scheme, however, which will become apparent later. One problem is that the signal term is now related to D rather than K . Since D is usually much less than K , we would expect the system to be more susceptible to noise.

8.3 Alternative Implementations

There are several possible implementations of the basic dynamic pattern idea. This section presents some of the options that were considered and the choices that were made. The options are broken into three categories. One consideration is the way in which feedforward connections are made from one subset of neurons to the next. A second is concerned with the way that neuron potential is modelled, while a third concerns the form of the input vector and the way it is presented to the network.

8.3.1 Interconnectivity Options

The aim is to store a pattern made up of L subsets, $(D_0, D_1, \dots, D_{L-1})$, each consisting of D firing neurons. The subsets fire in order of increasing index. Using a modified version of the simple learning scheme described in chapter seven, the simplest option is to make a connection of strength B from a neuron i to neuron j if neuron i is active in subset l and neuron j is active in subset $l+1$ (with wrap-around such that $D_0 = D_L$). Thus, at each instant of time, t , the active set D_t is feeding activity to the next subset which will replace it at time $t+1$. This is illustrated below.

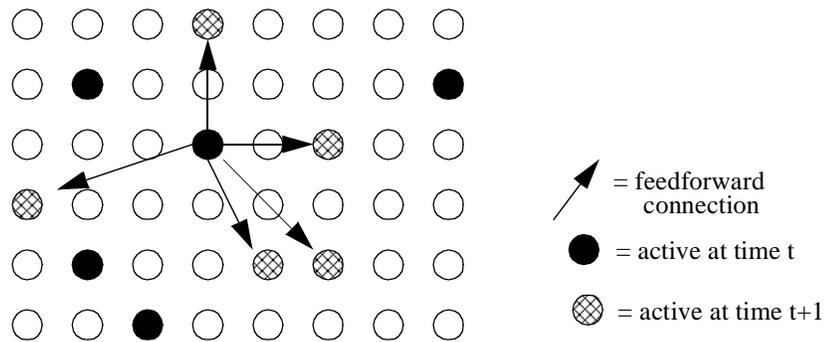


Fig. 8-4 Feedforward connections one subset to the next during learning.

For subsets of size D , D^2 connections are made at each step between the D currently firing neurons and the D neurons which will be firing at the next time step. By extension to the L steps, the total number of connections needed to complete the loop is LD^2 .

How does this compare with the static case? Since $D=K/L$, the total number of connections can be re-written as K^2/L ; the number of connections needed to store a K -from- N encoded pattern is inversely proportional to the number of steps, L . However, the signal term has been similarly reduced, from K to $D (=K/L)$ at each step.

This is the simplest connectivity option but suffers from the reduction in signal strength. For a particular pattern, p_0 , when the network is in a state corresponding to subset D^0_i , say, we want it to move on to state D^0_{i+1} in the next cycle. But with this simple connectivity there will be a problem if this same subset of neurons is encountered in a different pattern p_1 in which the successor subset is D^1_{i+1} .

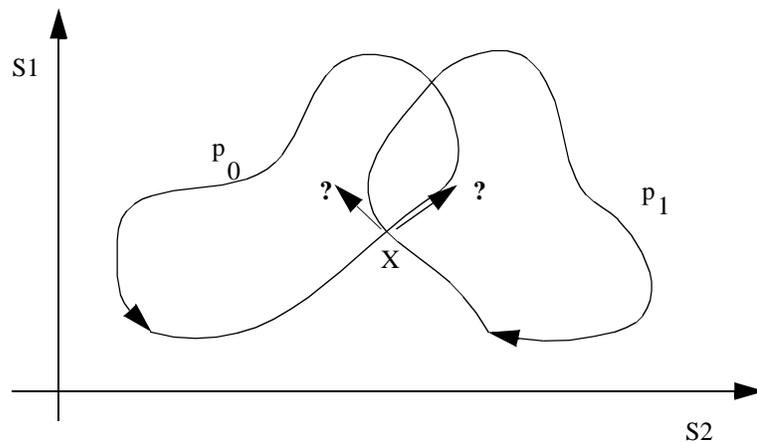


Fig. 8-5 State space showing the trajectories of two pattern, p_0 and p_1 .

In the figure, the two patterns have a point in common, one subset of the firing neurons indicated by the point X . Upon arriving at this point, there is no information to help decide which path the network should now follow. In practice, the potentials of neurons for both successor states will be high and it is likely that the next state will be a mixture of the successor states of the two patterns, leading to a breakdown in the recalled memory.

To escape this dilemma there are several potential solutions. We could arrange that no two subsets are the same so that trajectories never cross. But how would this be achieved in practice? This does not seem like a realistic solution, especially if we are to introduce an element of robustness later on. To allow for noise we would need to specify a guard band around each subset and demand that the guard bands of all patterns must be mutually exclusive. Again, this seems impractical.

Alternatively, the network could be given extra information to help differentiate between states which belong to different trajectories. Such options for the connectivity can be obtained by making feedforward connections two or more time steps further on in addition to the next-state case just described. This increases the signal term for each neuron at a cost of extra connections. This is illustrated below.

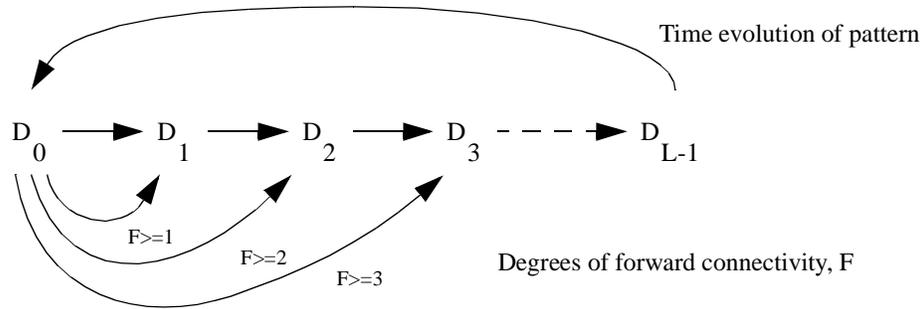


Fig. 8-6 Possible connection schemes over multiple time steps.

The degree of forward connectivity, F , is defined as the number of subsets ahead of the current one, D_t at time t , which make synaptic connections with D_t . As before, all synaptic connections are given a constant strength, B .

Thus, for neurons which belong to subset D_2 , connections would be made to provide input from neurons in subset D_1 if $F > 0$, also with D_0 if $F > 1$ and also with D_{L-1} if $F > 2$, etc. These connections are unidirectional so there is no reciprocal connection from elements of subset D_1 back to elements of subset D_2 (at least in the context of learning this particular pattern; such connections due to other patterns are not prohibited). At each step, the number of connections to the active set, D_t , is now FD^2 , since input is coming from the F previously active sets of neurons. Over L time steps, the maximum number of connections made to stored one pattern is FLD^2 . The signal term at each time step is now FD .

Examining in this case the problem of crossing trajectories, we see that the potentials which will determine the state at time $t+1$ are no longer solely determined by the state at time t . Instead the state for the F previous cycles is involved. Two patterns would need to have been identical over the previous F cycles to create a conflict over the next subset of firing neurons.

Clearly, for $F=1$, this new case reduces to the simpler option of connecting forward only to the next active subset of neurons at each timestep. Conversely, for $F=L$, every one of the K firing neurons in the pattern connects to every other such neuron including other neurons in the same subset. The weight usage is no better than in the simple network. This case is somewhat special, however. As long as the pattern remains in the same cycle, the potential of all neurons would remain constant and it is not clear how the network would decide from one time step to the next which subset should be the next to fire. Thus, it is reasonable to eliminate $F=L$ as a possible candidate for the dynamic pattern case.

As a final point, note that the case when $F = 1$ could be analysed as a Markov process, since each state transition depends only upon the current state and current input (Narendra & Thathachar, 1989). For higher values of F , a Markovian approach would require a super-state vector corresponding to the vector product of the state vectors of the F previous timesteps, so for a network which can be in any of S states, the Markov process would be defined for a state vector of length S^F . Such an approach will not be considered here, since the stochastic behaviour of the system is not required at this time. It might, however, form the basis of future work.

For the interconnect policy, the most general case allows an arbitrary degree of feedforward connectivity, F , and so this will be adopted. Part of the investigation will centre on the choice of the parameter F and the implications of that choice.

8.3.2 Potential Decay Options

The second area which will need clarification for the dynamic pattern case is the manner in which the potential of each neuron is to be handled. In the simple network, the potentials of all neurons were reset to zero and recalculated at each time step. However, for the dynamic case several other options will be considered.

Firstly, the potential could decay exponentially to zero with a time constant to be determined. Thus, the time course of the potential, U , would be governed by an equation of the form:

$$U_i(t+1) = \sum W_{ij} \cdot O_j(t) + v \cdot U_i(t)$$

where v is a decay constant in the range $0 < v < 1$. In this scheme, the potential fed forward from subsets which have already fired will decay slowly to zero. No memory is required of any state other than the current active subset of neurons.

This case is closer to the functionality of real neurons than the simple recalculate-at-each-timestep case (see e.g. Hall, 1992), but renders the analysis more complicated.

A second choice would force the potential to be reset and recalculated as before but with additional terms to account for previous active subsets feeding activity forward over multiple timesteps. In this case, the system is acting like a digital signal processor (DSP): previous state must be held over F cycles (where F , the degree of feedforward connections, was described in the last section) and a delayed version is presented to the network. In this case, the potential is described by an equation of this form:

$$U_i(t+1) = \sum_{f=0}^{F-1} \sum_{j=0}^{D-1} W_{ij} \cdot O_j(t-f)$$

where f ranges over the number of steps ahead a feedforward connection exists. Note that when a neuron fires, its activity, though lasting for only a single cycle, is felt by neurons downstream for F cycles.

At the end of this time, the activity is removed abruptly. There is no slow decay. Despite the benefits such a scheme offers, in terms of a well defined signal and easily defined cut-off point, the requirement to hold the state of the entire network over multiple cycles makes this a less practical option, viewed from the perspective of the implementation.

The decision was to use the simple digital signal processing approach allowing input from the previous F cycles of activity, since this would simplify the analysis. It was envisioned that future work would include an investigation into the more realistic and more realisable potential decay options in the dynamic pattern context.

8.3.3 Interface with Stimulus Options

Up to this point, the manner in which each region will interface with other regions or with external input has not been discussed in detail. With the addition of dynamic patterns, a number of possible options have appeared and so it seems appropriate to mention them at this point. Analysis will be presented in a later chapter.

Consider a single region connected to a source of stimulation, either an external input or another region. The stimulus is either static, or is itself a dynamic pattern. In the simple case it is static. It supplies activity to the neurons in the region in addition to the stimulus they receive from other neurons within the region. We assume that the input stimulus is enough to affect the pattern of activity within the region (if not, then the network is not performing any useful function with the incoming data).

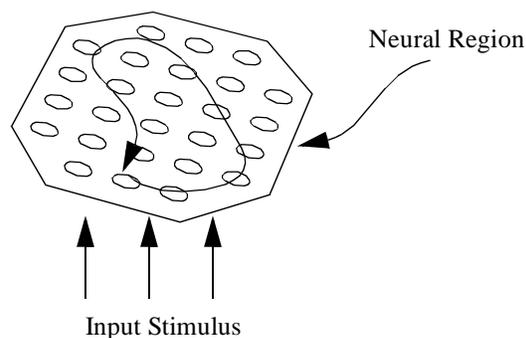


Fig. 8-7 External input to a region should cause changes in the output trajectory.

The change to the trajectory could be brought about in several ways. The simplest scheme has the region devoid of activity until the input appears. The input is used to set up the network in a particular state and then the input is removed allowing the network to proceed from that point, tracing out the stored memory.

In this case, the input is only used to specify a single subset, D_i , of the pattern. After it has been removed, the tracing of the rest of the pattern is in effect an associative recall between the single input stimulus and a pre-specified memory. This will be referred to as 'burst mode' since the input pattern appears briefly as a burst of activity and is then removed.

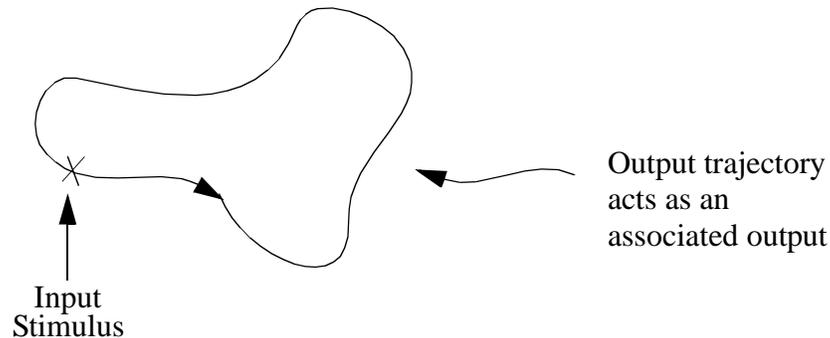


Fig. 8-8 Input specifies point of entry into recall loop.

The output pattern as defined by the trajectory can be totally unrelated to the input vector and is a straightforward way to use such a network for associative recall.

Clearly any input stimulus which triggers a subset of a given pattern could produce the same output pattern, albeit phase shifted by an amount which depends upon the point of entry into the loop. One issue which might prevent such useful functionality is a dependence on feedforward activity from F previous subsets of neuronal firing (as described in the earlier section on interconnectivity). When the input is applied, there is no previous history from earlier sections of the loop and so the transition to the next step around the loop may be prevented due to insufficient signal.

An alternative option has the network initially silent, as before, but when the input is applied it is not removed and forms a permanent input to the region while pattern traces its trajectory. The advantage of such a scheme is that there is now extra information to differentiate two otherwise identical states even if feedforward activity is only from one subset to the next. The disadvantage is that the ever present input signal places constraints on the potential of every neuron in the region at all times and so makes an arbitrary output pattern more difficult to achieve.

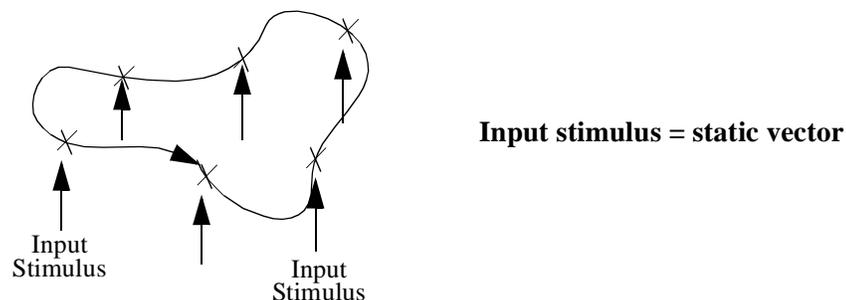


Fig. 8-9 Alternative stimulation scheme with constant input.

The third and final scheme to be considered was one in which the input stimulus itself is allowed to be a dynamic pattern. Thus, the stimulus is present at all times as in the scheme just described but the input pattern itself is changing every cycle, tracing out its own trajectory. If the region is to perform useful work then its output trajectory is not just an image of the input trajectory, but is a function of it.

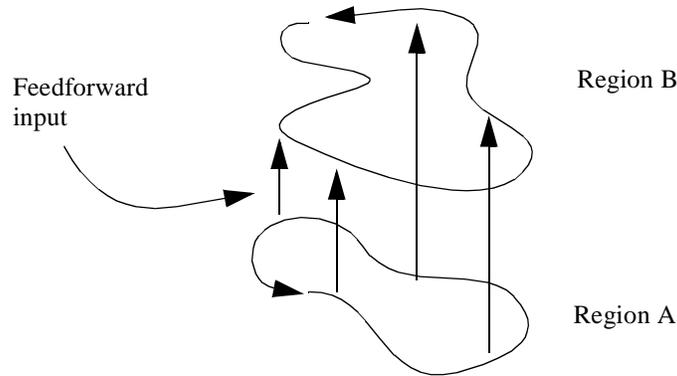


Fig. 8-10 Output trajectories of two regions, with B receiving input from A.

Since the eventual goal is to have many regions interacting using dynamic patterns, the choice for the input stimulus was that of interaction via dynamic patterns. This is the most complex of the proposed schemes but it was expected that justification of this choice would be an extension of the justification of the entire philosophy of dynamic patterns as a means of representing information.

8.4 Learning Dynamic Patterns

The network is to learn a pattern set, \mathbf{P} , consisting of Z patterns which are K -from- N coded and ordered as L subsets each made up of D firing neurons ($D=K/L$). In any such set, a firing neuron has output value $+1$ and a non-firing neuron has output value 0 .

Initially, the synaptic weights between neurons are set to zero. $\mathbf{W}=0$. Each pattern is presented only once. The order of pattern presentation is unimportant, but the ordering of subsets within each pattern is predetermined. For pattern p_0 , one subset, D_k^0 , is selected at random to be learned, k being the subset index. At time t , the subset D_k^0 is made to fire in the network while all others are silent. For all firing neurons, $O_i(t)$, in subset D_k^0 , the synaptic weights are adjusted as follows:

$$W_{ij} = B \text{ if } O_j(t-h) = 1, \quad 1 \leq h \leq F$$

$$= \text{unchanged, otherwise}$$

so that any firing neuron in the previous F cycles will make a connection of strength B with a firing neuron in the current firing subset. The index of the subset, k , is incremented at each timestep, but will wrap around with period L , so that

$D_0^0 = D_L^0$. For the first F subsets, there will be insufficient history of outputs $O_j(t-h)$ to make all of the necessary connections. The problem can be avoided by adding F extra cycles after the first period to add the final few connections.

The learning procedure is repeated for each of the Z patterns in the set, \mathbf{P} . At the end of a single training epoch in which every pattern has been presented once, the training is complete.

8.4.1 The Dynamic Connection Probability, h_D

In chapter seven the analysis of the simple network included a parameter h which was the *a priori* probability of a connection between any two neurons in a network of N neurons having learned Z K -from- N patterns. This parameter was used in the noise analysis of the network and it is useful to calculate a similar quantity for the dynamic network.

Consider the learning of a pattern p_z from set \mathbf{P} . This learning consists of making connections for each subset of the pattern. Consider the learning of subset D_k^z . Given two neurons, i and j , what is the probability that a connection is made from j to i during the learning of this subset? For a new connection to be made, we require two separate events to occur. First, neuron i must be one of the currently firing subset, D_k^z . Given that neurons are selected at random, the probability of this event is D/N . Second, neuron j must have been one of the firing neurons in one of the F previous subsets, which occurs with probability DF/N .

Thus, the probability, c , of making a connection is the product of the two independent events on i and j :

$$\begin{aligned} c &= \frac{D}{N} \cdot \frac{DF}{N} \\ &= \frac{K^2}{L^2 N^2} \cdot F \\ &= p^2 \cdot \frac{F}{L^2} \end{aligned}$$

making use of the relations $D=K/L$ and $p=K/N$. This connection could be made for any of the L subsets of pattern p_z . This the probability, c_z , of making that connection at some time during the learning of p_z is

$$\begin{aligned} c_z &= c \cdot L \\ &= p^2 \cdot \frac{F}{L} \end{aligned}$$

The quantity $c'_z = 1 - c_z$ is the probability that no connection was made between two particular neurons during the learning of p_z . To remain unmade over the whole pattern set (which consists of Z patterns) this event of probability c' must be repeated Z times. Thus, the probability, h_D , of making a particular connection at some time during the learning of the entire set of Z patterns, \mathbf{P} , is

$$\begin{aligned}
h_D &= 1 - (c'_z)^Z \\
&= 1 - \left(1 - \frac{Fp^2}{L}\right)^Z
\end{aligned}$$

In the case when the number of subsets $L=1$ and the degree of forward connectivity $F=1$, the learning algorithm reduces to that of the simple network described in chapter seven. In that case, the connection probability, h_D , reduces to the original value, h , derived for that network.

Comparing the number of stored patterns, Z_S , in the simple (static) with that for the dynamic network, Z_D , for equal weight utilisation ($h=h_D$) we see that:

$$\begin{aligned}
1 - (1 - p^2)^{Z_S} &= 1 - \left(1 - \frac{Fp^2}{L}\right)^{Z_D} \\
(1 - p^2)^{Z_S} &= \left(1 - \frac{Fp^2}{L}\right)^{Z_D}
\end{aligned}$$

Next, an approximation of each side is made using the binomial expansion, using the fact that the value of $p=K/N$ is the same in both the static and dynamic cases:

$$\begin{aligned}
1 - Z_S p^2 &= 1 - Z_D \frac{F}{L} p^2 \\
\therefore \frac{Z_D}{Z_S} &= \frac{L}{F}
\end{aligned}$$

Thus, given identical weight usage the ratio of the number of patterns that can be stored is proportional to the number of steps in the cycle, L , and inversely proportional to the degree of forward connectivity, F , to first order. This knowledge permits a trade-off between the number of stored patterns (increased with *increasing* L to F ratio) and the signal strength of each pattern (increased by *decreasing* the L to F ratio).

8.5 Analysis of Storage Capacity for Dynamic Patterns

As for the simple network, the storage capacity is made up of two terms: the number of patterns which can be stored and subsequently retrieved, Z , and the information content per pattern, I . The storage capacity will be taken as the product of these two terms.

Previous sections in this chapter have discussed the information content of a dynamic pattern. This section will consider the signal to noise analysis of the dynamic pattern case, leading to the conditions for stable recall of stored memories. This will follow the same process as presented in chapter seven.

The first step in analysing the recall of stored dynamic patterns is to decide what constitutes correct recall. The simple fixed point attractor of the static pattern case is now no longer applicable. Instead, a recalled pattern is characterised by a cycle of period L in which any subset D_i^0 of pattern p_0 which fires in cycle i must also fire in cycle $i+L$. If this condition is met then the subsets must fire in the correct order as specified by the original pattern, p_0 and will continue to do so indefinitely unless interrupted by some external event.

Next, since we accept that recall may not be perfect, we must define an acceptable margin of error which requires a suitable definition for the error. Two possible candidates were considered but it was clear that one of them was inappropriate. The simplest solution is to take the total error, given by the number of erroneous bits over the whole pattern. Using this measure, the error would be the same whether individual errors occurred spread throughout the recall of each subset D_i or concentrated in a single subset.

The problem with the total error approach is that it does not take into account the reality of recall: each neuron in the next firing subset, D_{i+1} , must depend upon the current firing set, D_i , and the $F-1$ previous such subsets to specify which neurons must fire next and which must remain silent (external input still being ignored at this point).

The alternative choice (which was adopted as the error criterion) is the sum of the individual errors which are visible to the next set of neurons to fire. Thus, the margin of error, M_D , is specified as the total number of errors present in any F consecutive subsets, D_i , of the pattern while still being able to maintain a stable cycle of period L . The subscript of M_D denotes that this margin is for the dynamic case.

Next, we consider the process of recall itself. Initially, the potential of every neuron is zero. Assume that the network has no activity at time t . At this time a corrupted version of a pre-learned dynamic pattern, p_0 , is applied to the network. The state at time $t+1$, S_0 , consists of D firing neurons but may contain errors due to noise or misclassification.

We define a parameter, $e_n(t)$, to be the number of erroneous bits in the currently firing subset at time t (the n subscript indicates that it is the network error). This parameter will vary with each cycle of the network. Similarly, we define $e_x(t)$ as the number of erroneous bits in the external stimulus. The network is said to exhibit perfect recall if

$$\exists T_{perf} \quad e_n(t) = 0, t \geq T_{perf}$$

In other words, if the value of e_n is zero for all time after a time $t=T_{perf}$ perfect recall has been achieved. During the recall process the desired goal is the fast convergence of $e_n(t)$ to zero.

Finally, note that $e_n(t)$ is also a measure of the noise generated by the subset of neurons firing at time t , as perceived by the next F subsets of neurons. In this analysis, correct recall was assumed only if the value of $e_n(t)$ did not exceed $0.5D$ for the first two cycles and did not exceed $0.25D$ thereafter. This gives the network two update cycles to bring the network state to within $0.25D$ bit shifts of a stored pattern starting from a highly corrupted pattern.

At time t , each neuron recalculates its potential based on the activity it receives from the firing sets over the previous F cycles:

$$U_i(t+1) = B \sum_{f=0}^{F-1} \sum_j W_{ij} \cdot O_j(t-f) - T_i$$

The D neurons having the largest potential are permitted to fire and the rest are silent. Once a neuron has fired, its potential is reset. This prevents a neuron with the highest potential remaining the winner over the course of multiple cycles. In addition, a neuron having fired is not permitted to fire again for a certain period of time, C_{ref} , specified in cycles. The purpose of this restriction is to ensure that any complete pattern made up of L subsets will not contain the same neuron multiple times.

The rule does have an analogy in real neurons: the absolute refractory period is the time after having fired that a neuron is physically incapable of firing again. Clearly, in the biological case this period is demanded by the biochemical process underlying the resting potential of the cell, but it is not impossible that its presence was used by the information processing level of the neural system to ensure that no one neuron was able to dominate in the firing set.

How long should the refractory period be? This issue will be considered in the context of the potential decay model for two reasons. First, it is expected that the network will move to this more complex model of neuron potential at a later

time and second, the refractory period has a greater impact on the workings of this model than it does on the DSP model employed in the current network.

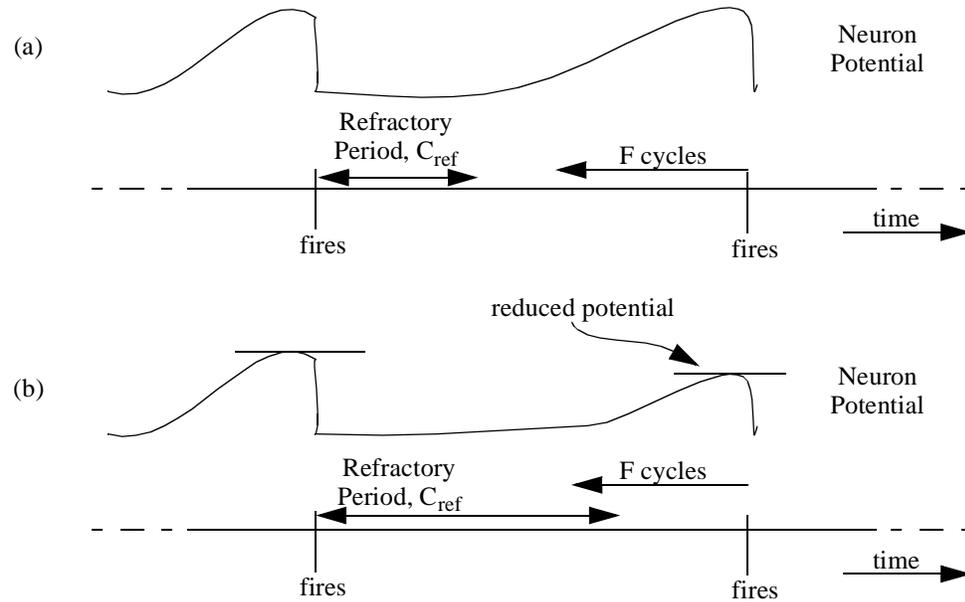


Fig. 8-11 Timelines neuron potential showing influence of refractory period.

Thus, following the potential decay model, if the dead cycles following firing overlap with the F cycles which should precede the next firing of the neuron, then part of the support set of neurons will be ignored. When the neuron is due to fire, its potential will be less than the full amount that it would have otherwise received. This is illustrated below

Thus we can constrain the length of the refractory period, T_{ref} , to fulfil the inequality

$$0 < T_{ref} < L - F$$

where L is the period of the pattern in cycles. Note that if the refractory period is much less than L-F the neuron will have time to build up potential before the start of the F cycles. This potential comes from the activity of neurons which are outside of the F cycle range, but which make connections with the neuron in patterns other than the one being recalled. In other words, noise due to crosstalk. While this potential is undesirable for the recall of the current pattern, it may be involved in the transition to another pattern in later network developments, so increasing the refractory period such that the neuron is only actively involved for the F cycles prior to its designated firing cycle might make network control more difficult.

Consider first a neuron in the firing set for the next time step, t+1. It receives input from two sources. First, from the external stimulus to be classified, and sec-

ond from the internal activity of the previous F cycles. In this case, separating the signal and noise terms in the expression for the potential at time t+1 gives

$$U_i^f(t+1) = B \sum_{f=0}^{F-1} [2D - e_n(t-f) - e_x(t-f) + a \cdot e_n(t-f) + a \cdot e_x(t-f)] - T_i$$

signal
noise

where a is a random variable and depends on the presence or absence of a connection between a neuron j (which fired erroneously during the F previous cycles) and the neuron i in question. The variable a has mean h_D , the mean connection probability for the dynamic case, and variance $h_D(1-h_D)$. Thus $U_i^f(t+1)$ is itself a random variable whose mean and variance depend on the history of errors $e(t)$.

Now consider a neuron which is not in the firing set for time t+1. To find out if it will fire erroneously and create an error bit, we need to form a probability distribution for its potential and use the error function to determine the probability that it has greater potential than one of the firing set for the next cycle.

There are two relevant cases for the non-firing neuron. The simplest is a neuron which takes no part in the firing of the current pattern at any time step. Thus all of its potential is due to crosstalk noise:

$$U_i^{nf}(t+1) = \underbrace{BaDF}_{\text{noise}} - T_i$$

where again a has mean h_D and variance $h_D(1-h_D)$. Thus, $U_i^{nf}(t+1)$ is a random variable of mean $BDFh_D - T_i$ and variance $B^2DFh_D(1-h_D)$.

The second case which must be considered is that of a neuron which, although *not* about to fire, *does* belong to a firing set which is due to fire in a number of cycles less than F in the future. Thus, it has made connections with some of the previous F firing sets with much higher probability than has a neuron which will not fire at all for the current pattern. The worst case is a neuron which will fire at time t+2, since it has received F-1 cycles of full potential as against F cycles for a neuron that will fire at time t+1:

$$U_i^{df}(t+1) = B \sum_{f=0}^{F-2} [2D - e_n(t-f) - e_x(t-f) + a \cdot e_n(t-f) + a \cdot e_x(t-f)] + 2BaD - T_i$$

signal
noise-int
noise-ext

where $U_i^{df}(t+1)$ is the potential of the *delay-firing* neuron, and the noise terms are divided into noise-int for noise due to pattern errors and noise-ext due to crosstalk from other memories at time t-(F-1).

For perfect recall, the potentials of all members of the next firing set, G_f , must have potentials higher than any other neuron. If we demand only than the

number of erroneously firing neurons is no greater than $0.25D$ this will permit a higher storage capacity.

As with the analysis of the simple static network, we next ask: what is the probability that a neuron which is a member of the non-firing subset has a potential greater than one which is a member of the firing subset? Two cases are considered, one for a non-firing neuron which is part of a later firing subset, the other (which represents the overwhelming majority of the neural population) representing a neuron which is not a member of any firing subset of this pattern.

To simplify analysis, all neurons in the soon-to-fire subset were taken as firing on the next cycle. This is, strictly speaking, more pessimistic than the worst case. Also, the number of erroneous bits was assumed worst case, such that $e(0) = 0.5D$, $e(1) = 0.5D$, $e(2)$, $e(3) \dots = 0.25D$.

Thus the mean and variance of a firing neuron, i , are:

$$\begin{aligned}\mu_{uf}^i &= 2B(1.5D + 0.5Dh_D - (1 - e_x)h_D) + (F-2)B(1.75 + 0.25h_D - (1 - e_x)h_D) - T_i \\ \sigma_{uf}^i{}^2 &= B^2h_D(1 - h_D) \cdot [2(0.5D + e_x) + (F-2)(0.25D + e_x)]\end{aligned}$$

where the first term of the mean is due to the two start-up cycles in which the output pattern can be corrupted by up to $c=0.5D$, the second term is for the remaining $F-2$ cycles, and the third term is the threshold, as before. The single external error value, e_x , is the number of bits which are allowed to be erroneous on each cycle of the pattern. For a neuron which should take no part in the firing of the pattern, the mean and variance of the potential are more simple:

$$\begin{aligned}\mu_{unf}^i &= FBh_D D - T_i \\ \sigma_{unf}^i{}^2 &= FB^2 D h_D(1 - h_D)\end{aligned}$$

Finally, for the neuron which will fire in the *next* firing subset the mean and variance of the potential are:

$$\begin{aligned}\mu_{udf}^i &= 2B(1.5D + 0.5Dh_D - (1 - e_x)h_D) + (F-3)B(1.75 + 0.25h_D - (1 - e_x)h_D) + 2Bh_D D - T_i \\ \sigma_{udf}^i{}^2 &= B^2h_D(1 - h_D) \cdot [2(0.5D + e_x) + (F-3)(0.25D + e_x) + 2D]\end{aligned}$$

We consider the two possible causes of error. The first case considers the probability that a neuron that is not intended to fire for this pattern, j , has more

potential than a neuron, i , of the currently firing subset. This is the probability that the difference in potential between neuron i and neuron j is negative.

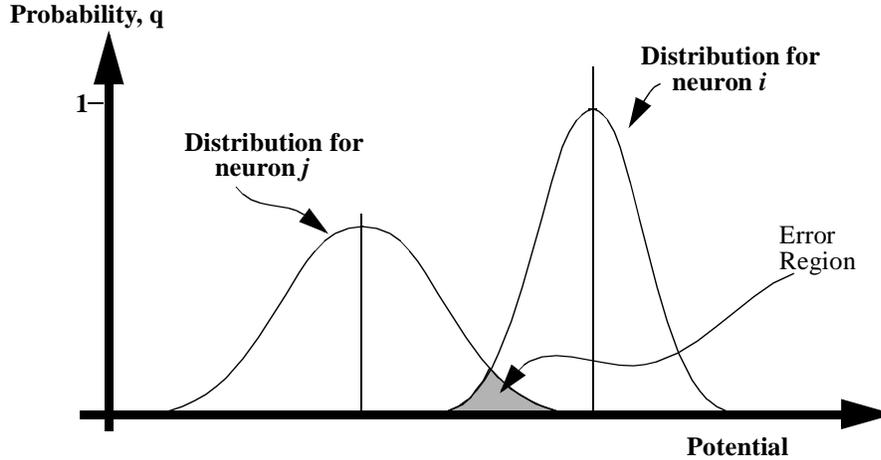


Fig. 8-12 Potential distributions for two neurons.

The difference in potentials ($U_i - U_j$) can itself be represented by a probability distribution, as was shown in chapter seven for the simple network. The mean of this gaussian is the difference of the individual means while the variance is the sum of the individual variances. Thus, the probability of one error from a non-firing neuron, P_{err-nf} , is the area under the curve for which $(U_i - U_j) < 0$. again, this is given by the error function, $\Phi(x)$.

$$Pr[(U_i - U_j) < 0] = Pr\left[\frac{x - \mu_{diff}}{\sigma_f + \sigma_{nf}} < 0\right] = \Phi_{nf}(0)$$

To obtain the expected number of errors, we note that there are D firing neurons, any of which can be surpassed by a non-firing neuron. The number of such non-firing neurons is $N - (1 + T_{ref})D$, where T_{ref} is the refractory period after firing during which time a neuron cannot become active. Thus the total number of independent events is $D \times [N - (1 + T_{ref})D]$.

For a near-threshold neuron, k , similar arguments between neurons i and k give a probability of one error, P_{err-df} :

$$Pr[(U_i - U_k) < 0] = Pr\left[\frac{x - \mu_{diff-(i,k)}}{\sigma_f + \sigma_{df}} < 0\right] = \Phi_{df}(0)$$

In this case, the number of neurons which can be surpassed is still D , but the population which supplies the near-threshold neuron, k , is only $(F-1)D$, corresponding to the firing neurons over the next $F-1$ cycles. Thus the expected number of errors is $D \times (F-1)D$.

Thus the total number of expected errors, N_{err} , at each time step is given by:

$$N_{err} = D[N - (1 + T_{ref})D]\Phi_{nf}(0) + D^2(F - 1)\Phi_{df}(0)$$

and we note that the probability of error, $\Phi_x(0)$, for each term is a function of the number of corrupted bits, e_x , in each subset of the input pattern. Increasing e_x leads to increasing probability of error and hence to increasing total error, N_{err} .

8.6 Case Studies & Simulations

Simulations were divided into two sections based on the form of network input supplied. In the first, 'burst input', the network was set into a state corresponding to one subset of a pattern and then released. The expectation was that the updating of the network state would trace out the rest of the pattern, performing an associative recall.

In the second mode, 'continuous mode', the input was itself a dynamic pattern made up of L firing subsets, each of size D . At each timestep the input pattern corresponded to a corrupted version of one subset of the input pattern (which is assumed to come from a network similar to that which is under test). For both input modes, the network parameters which were varied were the number of neurons N (100, 200 or 300 were used), the number of subsets, L , ranging from 4 to 12, the size of each subset, D , ranging from 12 to 32, and the degree of forward connectivity, F , ranging from 1 to $0.5D$.

The refractory period of each network was set as two so that a neuron, once it had fired, would be unable to fire again for at least two cycles. The only exception to this rule was for pattern with total period $L = 2$, in which case the refractory period was set to one.

Early simulations in which no refractory period was assumed always resulted in immediate failure to recall stored memories. On reflection, this is an understandable result since during the cycle after a neuron has fired the support set it receives from the $F-1$ previous cycles of activity corresponds very closely with the set which made it fire one cycle previously. Thus, there is a high probability that a firing neuron would fire again immediately if permitted to do so.

As for the static network, each network configuration was simulated only once due to the long simulation time (days to weeks on an IBM compatible PC at 100MHz). As before, our confidence in the results cannot be absolute and all conclusions that we draw must be tentative at this stage.

For each network configuration, patterns were learned in groups of ten and the network performance was measured as the number of patterns from the learned set that could be recalled correctly. Correct recall was assumed if the number of erroneous bits in the firing subset at each timestep was never greater than $0.25D$ during one entire circuit of the pattern. A grace period of two timesteps was allowed at the start of recall to give the network time to settle before the convergence constraint was imposed.

8.6.1 Comparison of Burst and Continuous Modes

The graphs included at the end of this chapter compare the network storage capacities of several networks. In each case, the number of neurons, $N = 300$ and the number of subsets, $L = 4$. The graphs show the number of errors recorded against the number of stored patterns.

For burst mode, all cases in which $F > 1$ resulted in immediate breakdown of recall; The network diverged immediately. This was probably due to the fact that the support set for the pattern consisted of at least two subsets of size D (from multiple previous firing subsets), whereas the burst mode input provides only a single subset as the initial state. Thus the initial signal is at most half of that present during learning and was presumably insufficient to initiate the recall cycle.

For continuous mode, in which the input pattern was also dynamic and was continually provided to the network, the storage capacity was greater than that in burst mode in every case. Graphs for $N = 300$, $L = 4$, $D = 12$ show that the capacity (expressed as the number of patterns stored before the first recall error) is almost doubled for any given value of corruption, c .

8.6.2 Comparison of Networks with Different Degree of Forward Connectivity, F

For continuous mode networks, the cases in which the degree of forward connectivity, F , were greater than one did not necessarily lead to pattern divergence during recall. The graphs at the end of the appendix compare the cases of $F = 1$ and $F = 2$, for a network of 300 neurons, using $L = 6$ firing sets of $D = 12$ neurons each.

We note that the higher connectivity option, $F = 2$, leads to a lower limit for storage before saturation and divergence occur, as we would expect. For example, for two bit errors per pattern, $c = 2$, the network can store 100 patterns using $F = 1$ connectivity before the first errors of recall occur. For $F = 2$ this is halved to only 50 patterns. However, we know that the signal term is doubled in moving from the $F = 1$ to the $F = 2$ connectivity scheme. Thus, we see evidence that the designer can trade-off storage capacity against signal-to-noise ratio in a simple way.

8.6.3 Comparison of Network Storage Efficiency for Different network Configurations

Finally, the table in the appendix compares the storage efficiency of the network as measured by the total number of stored patterns per synapse for a range of different networks. The rows are arranged in descending order of storage efficiency, which is taken as the number of stored patterns per synapse in the network expressed as a percentage.

Note that for the continuous dynamic pattern, the number of synapses is double that for the other network types, since there are connections both between neurons in the network and between the network and the source of the input pattern.

In general, the burst dynamic network has the best efficiency, but it should again be noted that in this case the signal terms is low, being based only on the activity of a single subset in the initial state of the network. During updating, only the D previous firing neurons determine the next state, compared to the 2D neurons for the continuous dynamic network and the K (=DL) neurons for the static case.

8.7 Comparison of Static and Dynamic Storage

The table below compares the dynamic pattern case with that of static patterns, not just in terms of the maximum number of stored patterns, but also the information content of each pattern. Only the burst mode dynamic model was considered to provide a fair comparison with the static pattern storage of the simple network

For each value of K from the static case, several possible ways of partitioning it between L subsets of D firing neurons are shown, with the maximum number of patterns that could be reliably stored and recalled, Z and the information content of each pattern I_{\max} . In every case, patterns were learned in groups of ten, so the maximum resolution that was obtained was set at ten patterns.

In each case, no bits in the patterns were corrupted; thus correct recall was assumed if an initial network state corresponding to stored patterns did not diverge beyond a margin $M = 0.25D$ or $M = 0.25K$ (for the dynamic and static pattern cases, respectively) during eight updates of the network.

Table 8-0, below, compares different values of L and D for a dynamic network using burst input against the same number of firing neurons, $K = D \times L$, in a static pattern network.

The three columns on the left are the simulation results for the static pattern case, while the four columns on the right are for the dynamic case. The columns indicating the maximum number of stored vectors in each case have been shaded.

From the table, it is clear that in general the number of stored vectors under dynamic storage is at least twice that of the static case and that the information per vector is also higher for the dynamic case.

Table 8-0 Comparison of information storage for static and dynamic patterns (in burst mode), with $N = 300$ and $F = 1$.

Total Firing Neurons, K	Information Stored per Pattern, I_{\max} (bits)	Max. Patterns Stored without Error, Z	No. of Subsets, L	No. Firing per subset, D	Max Information per Pattern, $I_{D-\max}$ (bits)	Max Patterns Stored without Error, Z_D
24	117	> 100	2	12	140	260
40	166	80	2	20	206	130
56	204	50	2	28	262	100
48	186	70	4	12	283	100

Table 8-0 Comparison of information storage for static and dynamic patterns (in burst mode), with $N = 300$ and $F = 1$.

Total Firing Neurons, K	Information Stored per Pattern, I_{max} (bits)	Max. Patterns Stored without Error, Z	No. of Subsets, L	No. Firing per subset, D	Max Information per Pattern, I_{D-max} (bits)	Max Patterns Stored without Error, Z_D
80	247	30	4	20	415	60
112	282	20	4	28	527	40
72	234	30	6	12	427	70
120	287	20	6	20	625	40
168	292	10	6	28	793	30
96	267	20	8	12	572	50
160	295	10	8	20	835	30
228	240	10	8	28	1060	30

In spite of the increased number of stored patterns in the dynamic case, and the increased information stored per pattern, an equivalent static pattern is more robust since each corrupted bit is a smaller fraction of the entire firing set (K bits as compared with only $D=K/L$ bits for the dynamic pattern). Thus for increasing L , the total signal term is reduced

The graphs overleaf show the fall-off in the number of stored patterns with increasing cycle period length, L , and the total information stored in the network, respectively. Considering the first graph of stored patterns against subset size, we note that more patterns can be stored for low D , all other parameters constant, which is similar to the result for the static pattern network in which low activity, K , permits a higher total number of stored patterns, Z_{max} . Increasing cycle length leads to a reduction in Z_{max} , due to a fall in the ratio of signal to noise.

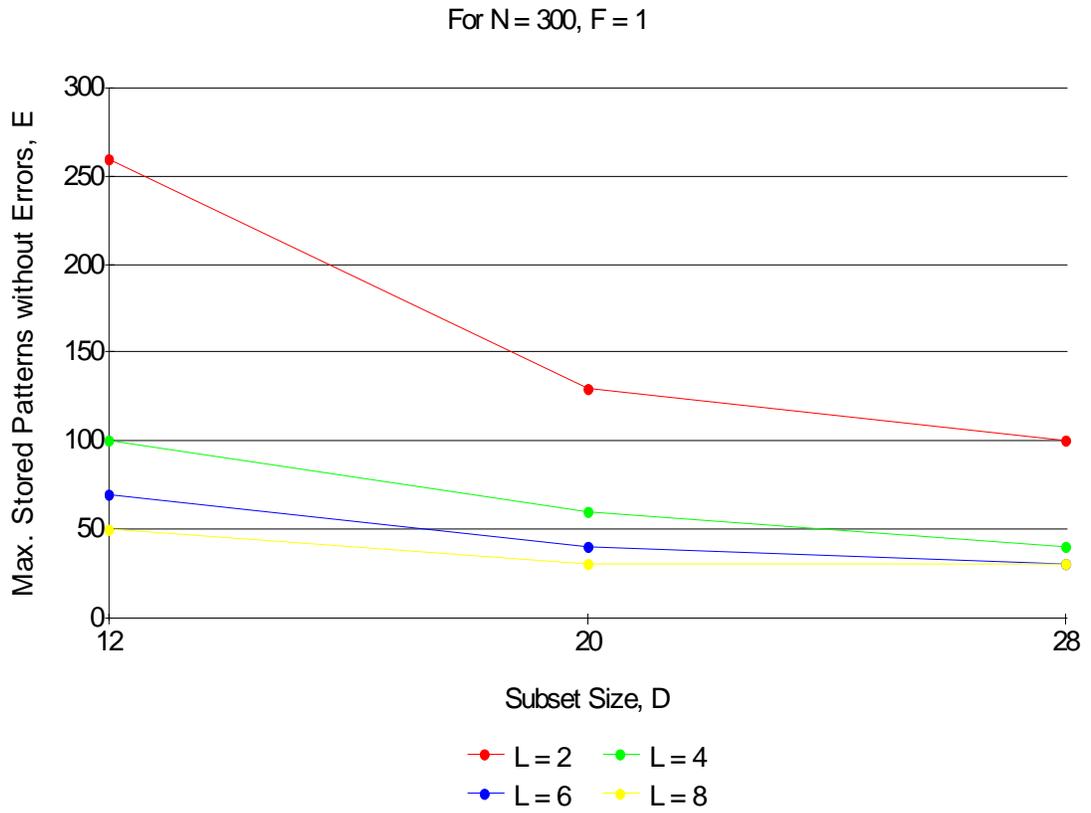


Fig. 8-13 Max stored patterns vs. size of firing subset, from burst mode simulation.

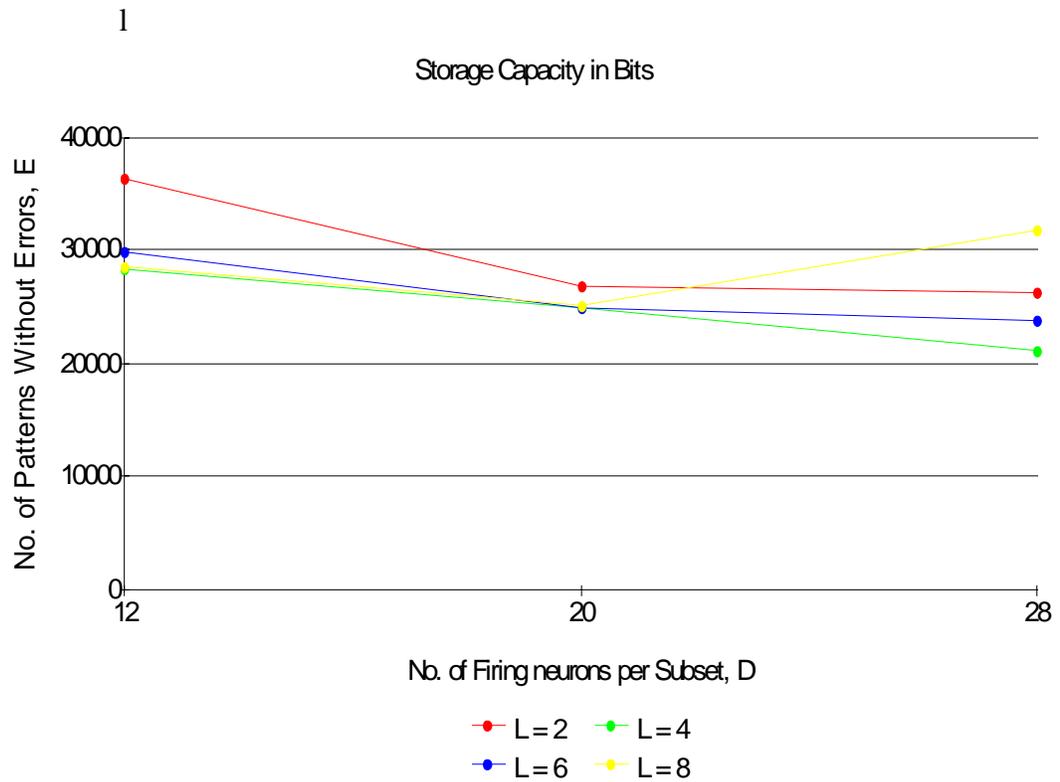


Fig. 8-14 Total storage capacity (bits) vs. number of firing neurons per subset.

We now turn to the second graph, in figure 8-14, which shows the effect of subset size, D , on the total information capacity of the network. Note that, somewhat surprisingly, there is no smooth flow in the lines for each cycle length, L . Instead lines cross each other making it difficult to detect any trend. This is probably due to the effects of noise, coupled with the small network and training set sizes and reminds us that the simulation run was performed only once for each network configuration. It would be useful to run the simulations several times for each such network. It is left for future work and a more powerful computer.

Tentatively, for various values of D and L , we conclude that the total quantity of information stored in the network is approximately constant. Thus, to first order the designer may trade off the maximum number of stored patterns against the information content per pattern for a network of fixed storage capacity.

Finally, we compare the total quantity of information for the dynamic network against a Hopfield network in which the N units have a value $+1$ or -1 with probability 0.5 , so each vector hold N bits of information. It is known that the statistical limit of such a network for large N is that approximately $0.138N$ patterns can be stored. Thus, for a network of $N = 300$ neurons we expect to store 41 patterns of 300 bits each, giving a total storage capacity of 12,420 bits. For values of L between 2 and 8, and for values of D between 12 and 28, we note that the total capacity never falls below 20,000 bits, in a network of 90,000 1-bit connections.

Thus, the storage of low activity dynamic patterns is clearly a more efficient means of storing information although the Hopfield network has a larger signal term (all N bits, compared with a total of $K = L \times D$ for the dynamic network). Thus we would expect the ability of a Hopfield network to correct errors to be greater, in general.

8.8 Conclusions

This chapter has presented the extension of the network to the storage and retrieval of dynamic patterns. The simple static vectors learned by the network in chapter seven have been replaced by a time domain multiplexed pattern of activity which forms a cycle of constant period, L .

The aim of such a development was twofold. To improve the capacity of the network and to exercise more control over the network when in a state which did not correspond to a stored pattern. This second goal was intended to overcome the local minima problem inherent in the simple network and others such as the Hopfield network.

It was shown that the total information content of a K -from- N vector is increased when it is divided into L subsets, each of size D , but must be replayed over L cycles. Thus the overall bandwidth required to transmit such a pattern to other networks is increased by a factor of L . The increase in information content is increased by a factor less than L , thus the transmission efficiency is reduced for the dynamic case.

Simulations showed that the capacity of the network was indeed increased, both in terms of the information stored in each pattern and in the number of patterns stored simultaneously in the network. The total storage capacity using burst mode dynamic patterns was shown to be fairly constant for a particular $K = D \times L$. Thus, the designer could choose to store more patterns at a cost of fewer bits of information per pattern by selecting large, or *vice versa*.

Comparison with a Hopfield network of equivalent size showed that the storage of low activity dynamic patterns leads to a much higher overall information storage.

However, in spite of the improvement in network storage capacity, in absolute terms it is still very limited. In addition, any improvements in the controllability of the network have yet to be demonstrated. A later chapter will build on the dynamic principle by introducing 'hierarchies of learning'. The aim will be to facilitate the assimilation of new data by re-interpreting it in terms of previously stored data. This step will reduce the quantity of network resources needed to store each new association, leading to an overall increase in effective network capacity.

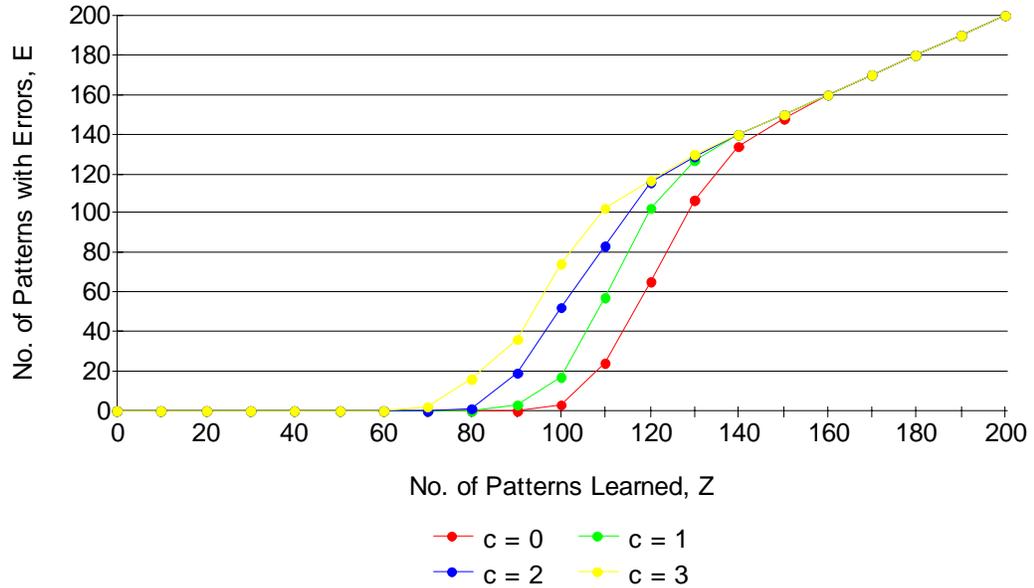
The next chapter on network control will show how the dynamic approach can be used to control the transitions from one stored memory to another.

8.9 Appendix: Graphs of Simulation Results

8.9.1 Burst Mode & Continuous Mode Results

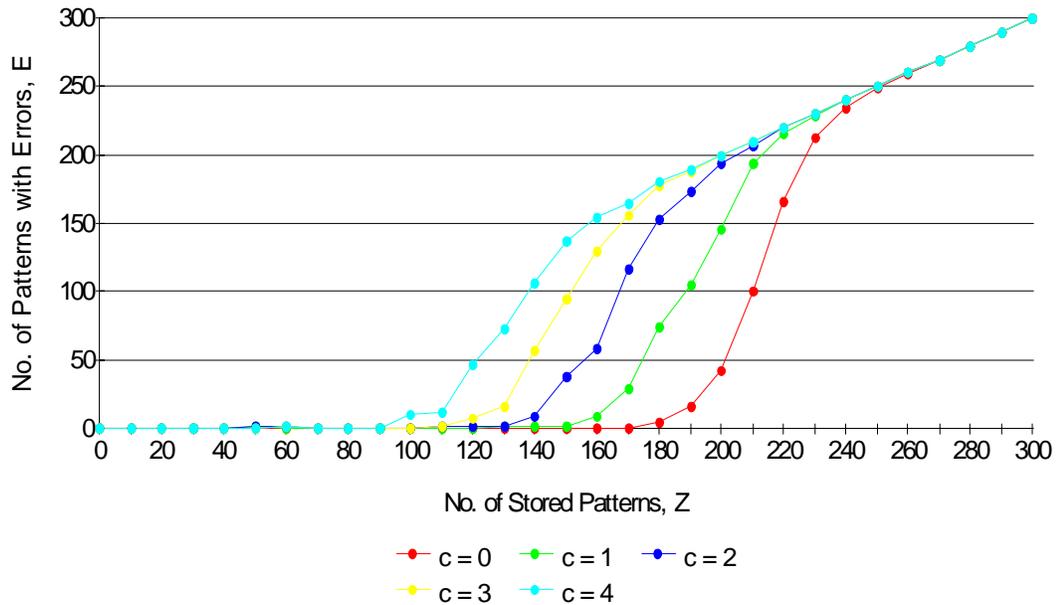
Burst Mode

For $N = 300$, $L = 4$, $D = 12$, $F = 1$



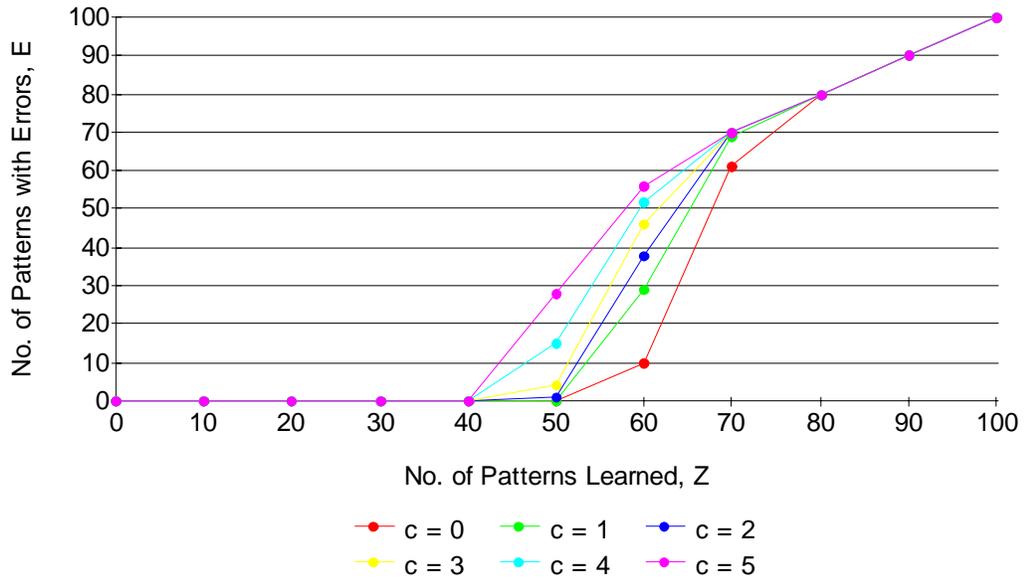
Continuous Mode

For $N = 300$, $L = 4$, $D = 12$, $F = 1$



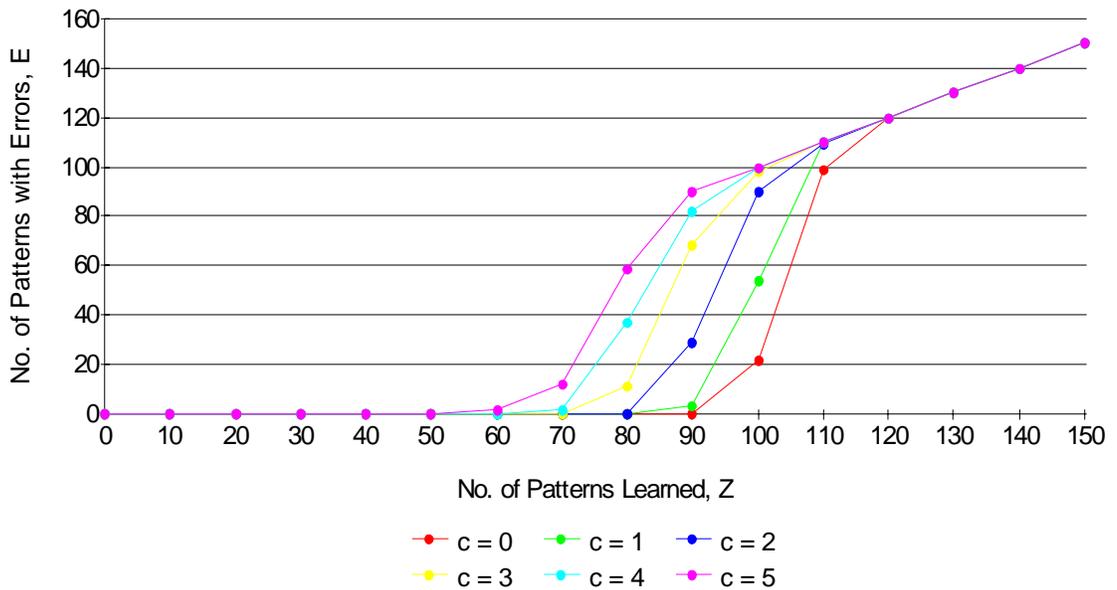
Burst Mode

For $N = 300$, $L = 4$, $D = 20$, $F = 1$



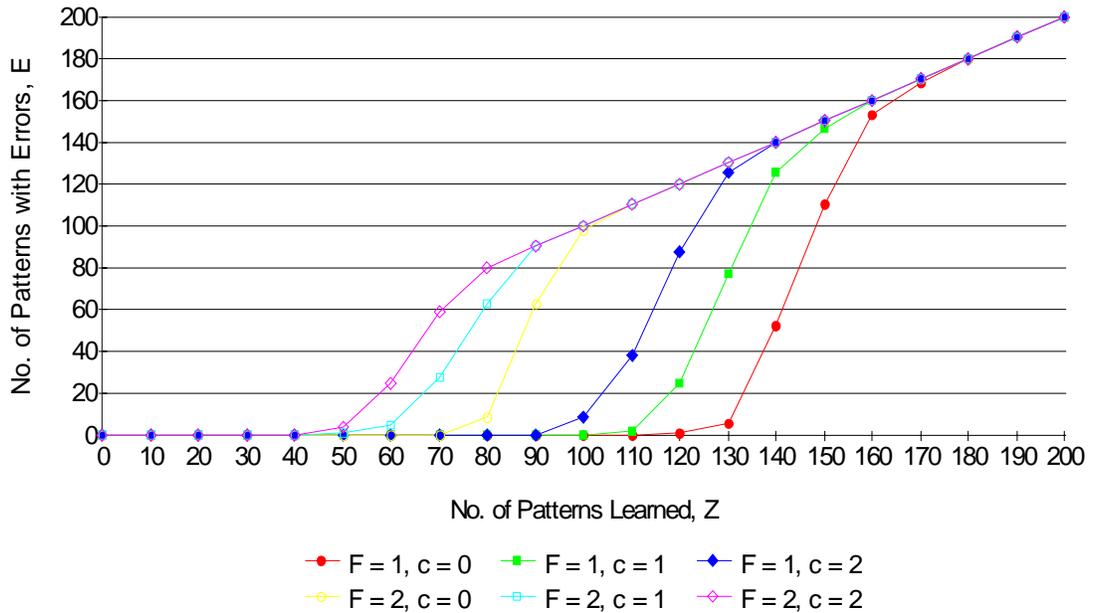
Continuous Mode

For $N = 300$, $L = 4$, $D = 20$, $F = 1$



8.9.2 Comparison of Networks with Different Degree of Forward Connectivity, F

For N = 300, L = 6, D = 20



8.9.3 Efficiency of Storage For Various Network Configurations

Table 8-1 Efficiency of a number of networks, in descending order of efficiency.

No. of Neurons, N	Network Type (Static, Burst or Continuous)	No. of Synapses, s	No. Firing per Subset, D	No. of subsets, L	Degree of Forward Connectivity, F	No. of Stored Patterns without Error, Zmax	Efficiency = Zmax/s x 100%
300	Burst	9×10^4	20	2	1	130	0.14
300	Burst	9×10^4	12	4	1	100	0.11
300	Static	9×10^4	40	1	1	80	0.09
300	Burst	9×10^4	20	4	1	60	0.07
300	Continuous	1.8×10^5	20	4	1	100	0.06
300	Static	9×10^4	56	1	1	50	0.06
300	Burst	9×10^4	20	6	1	40	0.04
300	Continuous	1.8×10^5	20	6	1	60	0.03
300	Continuous	1.8×10^5	28	4	1	60	0.03
300	Static	9×10^4	72	1	1	30	0.03
300	Continuous	1.8×10^5	20	6	2	40	0.02
300	Static	9×10^4	120	1	1	20	0.02

Pattern Association and Network Control

9.0 Introduction

The architecture described in chapter five developed, in qualitative terms, a set of operations required for a symbolic system implemented using a neural network or similar structure. Subsequent chapters developed a coding system for the patterns/symbols and analysed its developing storage and recall capabilities. This chapter focuses on the middle ground between these two parallel threads. Specifically, on the application of the neural building block to the association of patterns and the method of controlling the evolution of the network output using an external control pattern. This control input will permit not only the storage of individual patterns, but of the mappings between them.

The plan for this chapter is roughly as follows. First, the interpretation of control in the context of the network will be discussed. Next, the mechanism for associating patterns will be developed that permits the control network to force transitions from one stored pattern to another by virtue of pre-stored mappings. Finally, simulations will examine the performance of the network for a set of patterns and associations.

9.1 Symbolic View of Network Control

In symbolic terms, the control pattern is intended to facilitate the translation of one pattern into another. In chapter five, this translation was represented thus:

SymbolA $\xrightarrow{\text{SymbolB}}$ SymbolC

For example:

France $\xrightarrow{\text{has_capital}}$ Paris_city

Generically, symbolA is transformed into symbolC by virtue of control pattern SymbolB. The source of the transforming symbol, SymbolB, is the control network. The input and output symbols are represented by the data network. Each association is made between existing symbols, thus we assume in the example that the symbols *France*, *Paris_city* and *has_capital* are already in existence and may already be involved in numerous other relationships.

In the neural network implementation of this model, existing patterns which represent the individual symbols must be associated in such a way as to facilitate the given transformations. The object of this chapter is to propose and analyse potential mechanisms for this process. Two alternative ways of handling control will be discussed.

9.2 Energy Landscape of Network Control

In this scheme the pattern to be transformed is represented on the data network while the control pattern acts as an external input. When the control pattern is changed, the state of the data network is forced to make a transition to a state corresponding to the associated pattern. Thus, network control is achieved by manipulating the shape of the energy landscape of the data network using the external control pattern. Consider the recall of an existing mapping of pattern A onto B when subject to control input d at time t.

$$A \xrightarrow{d} B$$

This process is illustrated in the figure below which begins by considering a pattern as a static vector, before being extended to the dynamic pattern case. The state of the network can thus be represented as a point in N-dimensional space.

Initially, the network state begins in a state, A. It is assumed that the network arrived in this stable cycle due to some previous control pattern, c. At this time the cycle corresponding to B is inaccessible (figure a). At time t+1, the control input changes to d. The desired result is that, due to some previous learning, the network state will now be transformed to that of state B, performing the mapping. To do this, the effect of control input d must be to change the shape of the energy landscape around state A, destabilising it, and thence to cause the network state to evolve towards state B. This is illustrated in figure (b) overleaf.

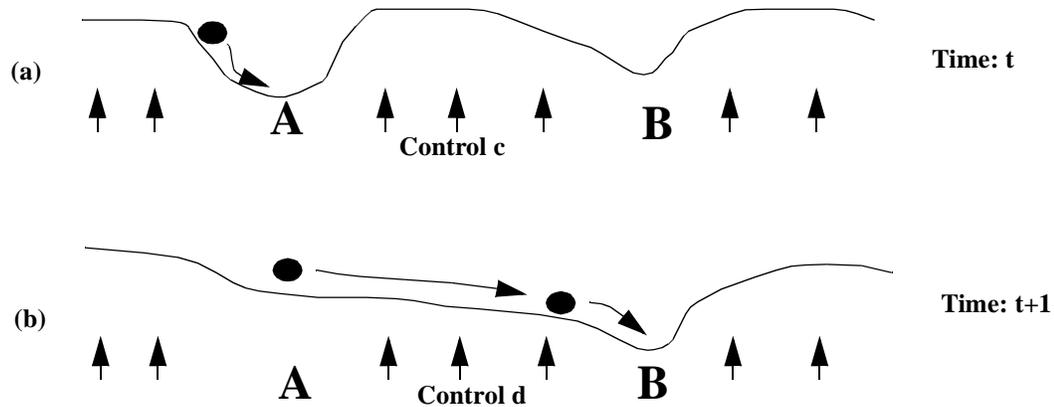


Fig. 9-0 Changes in the energy landscape due to changing control input.

The state of the network will arrive at state B as prescribed. At the next time step ($t+2$) a new control input ('e', not shown) arrives that destabilises state B, and so on. Thus, control inputs act by changing the underlying shape of the landscape. The network state merely has to evolve in an energy-reducing manner to implement the required transformations.

Note that in the preceding discussion, the evolution of the control patterns was not considered. It is assumed at this point that the control network undergoes similar changes in its energy landscape, guided by the output of the data network. In its conceptually simplest form some gating or timing mechanism must exist to ensure that the state of the data network has stabilised before the control network is allowed to change its state, and vice versa. Thus, the focus of activity is 'ping-ponged' between the two networks.

9.2.1 Energy Landscapes using Dynamic Patterns

The basic view of control as the manipulation of the energy landscape can be extended, with some cautionary notes, to the case of dynamic patterns. A stored pattern is now a cycle in state space rather than a fixed point, so a transition now represents a change in the closed set of states which the network will visit rather than a simple change from one particular state to another.

In spite of this change in definition for a stored memory, the essential idea remains unchanged. In this context, an input pattern in the context of one control pattern will settle into a particular cycle of states which will persist until the control input changes. At this time, the closed cycle of states will be exited and the network will enter a new cycle of states, corresponding to the new pattern.

9.3 'Dual Pattern' View of Control

An alternative way of implementing pattern association and control is as a mapping of two input patterns (external stimuli) onto a single pattern on the data network. The input data and control symbols could be considered as a single input pattern, made up of L subsets each of size $2D$.

SymbolA + SymbolB \longrightarrow SymbolC

We could consider two cases, one in which the two patterns were synchronised and another where there may be an unknown phase lag from one to the other. Detecting and correcting for this phase difference would presumably place extra constraints on the patterns themselves leading to a lower storage capacity.

Up to this point in the work the problem of phase difference has been avoided by insisting that a pattern is recalled in phase with the stimulus. An output pattern which is out of phase with respect to the input would thus be considered as an erroneous recall. For a single stimulus this is not a problem since the recalled pattern is automatically in phase with the input. When extending the network to two input stimuli (such as we have in control case) the difficulty arises when the stimuli themselves are not in phase with one another.

For simplicity we assume that there is no phase lag between the input patterns and that the output is therefore generated in phase with them also. By making this assumption we are placing constraints on the rest of the system and in particular on the degree of synchronisation which is maintained between any pair of interacting sub-systems (that is, two networks which receive non-overlapping input perhaps from entirely different sources).

During recall then, the dual pattern SymbolA+SymbolB is applied to the network whose state evolves normally to produce the output pattern, SymbolC, which was associated with that pair of input patterns during training.

The advantage of this scheme, compared to the energy landscape method, is that it is easier to analyse. Essentially, the network acts as a pattern associator. The disadvantage is that there must be a mechanism to transfer a stable pattern from the output of the data network to another network which will act as the source of external input to the data network in some future computation. This could either be handled by the control network as an explicit step or by some non-network based entity which uses the stability in the output pattern as a trigger to make the transfer.

Neither approach seems satisfactory for a number of reasons. In the first case, relying on the control network to initiate the transfer leads to a circular argument, since some other agent would presumably be required to perform the same action for the control network (which was assumed to be based on the same network architecture). Thus, some implicit mechanism must be involved which is outside of the explicit control of either control or data networks. The use of output stability as a trigger during sequential computation was suggested by Amit (Amit,

1989) but suffers from three obvious drawbacks. Firstly, it is an inherently dangerous criterion since there may be low level changes in the pattern during each cycle due to noise which would reduce the detection of stability to a probabilistic event.

Secondly, it may be inefficient to wait until the network has completely converged to a stable point when a semi-stable point may contain enough information for the next stage of computation to begin. The speed of operation of the network might be compromised if the transition between timesteps is based on such a highly conservative criterion. However, it is noted that the extent to which this is true is clearly a strong function of the symbol encoding and the network implementation itself.

The third issue with stability as a trigger for the next computational step is less rigorous and thus more open to criticism, being concerned more with the philosophy of the neural network approach. It is that the digital nature of the transition seems out of character with the highly fluid and dynamic environment we wish to create.

It seems difficult to imagine the human brain using a ‘start-stop’ architecture for each of its computational steps. Of course it is dangerous to assume that this is *not* the case since we have little knowledge today as to how the brain actually performs such sequential computations. It is conceivable that it does in fact undergo a set of digital transitions, but at a level that is difficult for us to analyse using current measuring techniques. More likely however, is that the energy landscape approach is a more accurate model, in which the state of the data network evolving continuously in time in parallel with that of the control network without explicit hand-over of control or the abrupt shunting of patterns from one network to another. This approach is taken by a number of current researchers. A prime example is Kelso who treats the evolution of the brain state as a chaotic attractor and is seeking an explanation of neural activity in terms of the order parameters which characterise its dynamic evolution (Kelso, 1997).

For the purposes of this work however (which is inspired by the human brain without claiming to be representative of it) the solution to be adopted is the simplified dual pattern approach. This will act as a simple demonstrator of the basic ideas presented in the architecture and may form some part of a more powerful and realistic interpretation in future work.

One further assumption that was made was that the cycle time L of all patterns was assumed to be constant for every network. If this constraint is not used, the phase between the dynamic patterns on two connected networks would vary linearly with time, leading to a beat frequency $f_b = \frac{1}{L_1} - \frac{1}{L_2}$ between patterns with periods L_1 and L_2 , with $L_1 < L_2$.

9.4 Implementation of Pattern Association

The objective of this part of the work is to cause the appearance of two given symbols patterns to trigger the recall of a third. The next logical step is to ask how the required associations are made. The figure below illustrates the network set-up required to achieve this.

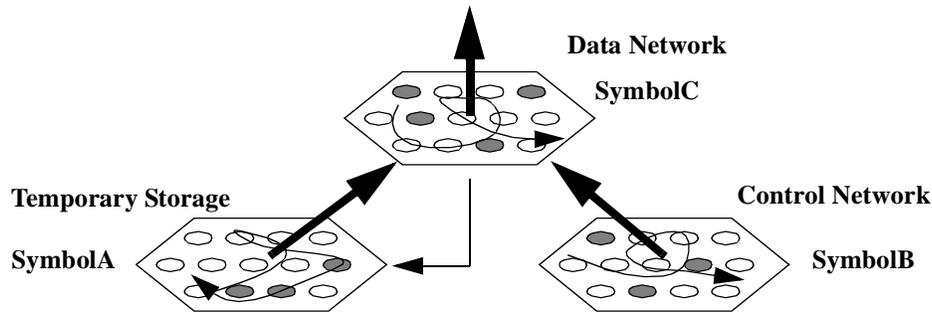


Fig. 9-1 An implementation of control by association.

The figure shows that three networks are required, one for each symbol in the association. The control symbol is held in a control network and is selected by the control mechanism based on the data network output at time $t-1$. The temporary storage, as discussed in the architectural development, contains only a few patterns, all of which have been recently evolved in the main data network. For reasons given in that earlier chapter, only the data network is capable of representing the associations between the symbol patterns.

The association process consists of two steps. First, the three patterns are first set-up on the three networks. Next, connections between all three symbols are made by short-term learning.

9.4.1 Dynamic Learning Algorithm for Association

Association between patterns will be implemented using the same learning algorithm for dynamic patterns as was described in the previous chapter. The output symbol, input data symbol and control symbol will be represented on the data network, N_D , the temporary store, N_T and the control network, N_C , respectively.

Initially, the outputs of each network are set-up with the required patterns to be associated. Thus to make the association:

$$\text{SymbolA} \xrightarrow{\text{SymbolB}} \text{SymbolC}$$

then symbolA, symbolB and symbolC are set-up on N_T , N_C and N_D , respectively. Connections are then made between the firing subsets of the data network

and those of the temporary store and control network, according to the following prescriptions. For the weights between the data and control networks:

$$W_{ij}^C = B \quad \text{if } O_i^D(t) = 1 \quad \text{and} \quad O_j^C(t-f) = 1, \quad 1 \leq f \leq F$$

$$= \text{unchanged, otherwise}$$

where W_{ij}^C is the weight vector between the two networks, O_i^D is the output of neuron i in the data network and O_j^C is the output of the neuron j in the control network. The variable f allows the firing of the pre-synaptic neuron to occur up to F timesteps in the past and still create a new connection, where F is the degree of feedforward connectivity, as defined for the dynamic pattern network.

For the weights between the data network and the temporary store:

$$W_{ij}^T = B \quad \text{if } O_j^D(t) = 1 \quad \text{and} \quad O_j^T(t-f) = 1, \quad 1 \leq f \leq F$$

$$= \text{unchanged, otherwise}$$

where this time W_{ij}^T is the weight vector between the data network and the temporary store and O_j^T is the output of neuron j of the control vector.

As for all networks discussed so far in this work, learning is a one-step operation. The expectation from this association is that after a single presentation of the three patterns together during learning, the subsequent application of the two input patterns will be enough to trigger the production of the output pattern, as required by the architecture.

As the results of simulation will show, this is indeed the case, although the total storage capacity of the network is still not sufficient for the purposes defined in the architecture.

9.5 Analysis using Dynamic Pattern Learning

The analysis begins assuming (i) that a number of patterns have been independently stored in the network and (ii) that a number of associations between triplets of patterns have been established using the learning rule just described. The aim of this analysis is to show that the scheme itself, while sufficient for a few pattern associations soon leads to network saturation and so, in its current form, is not sufficient to implement the architecture.

The potential of any neuron using the control scheme detailed in the last section is given by the following general expression:

$$U^D_{i(t+1)} = B \underbrace{\sum_{f=0}^{F-1} \sum_j W^D_{ij} \cdot O^D_{j(t-f)}}_{\text{Internal connections}} + B \underbrace{\sum_{f=0}^{F-1} \sum_j W^T_{ij} \cdot O^T_{j(t-f)}}_{\text{From temp. store}} + B \underbrace{\sum_{f=0}^{F-1} \sum_j W^C_{ij} \cdot O^C_{j(t-f)}}_{\text{From control}} - T_i$$

where the symbols have their usual meanings. The analysis of storage capacity follows similar lines to that used for the dynamic case except that there are now two sources of external input. We could consider the input as a single vector of size $2N$, which produces a firing pattern of $2D$ active neurons (D '1's in each half of the $2N$ -bit vector) with the pattern repeating every L cycles, as before.

Assume that the network is presented with uncorrupted versions of the input patterns c_0 and p_0 on the control and temporary storage networks, respectively, and that these patterns were associated with the output pattern p_1 due to previous learning. Let the network inputs at time t from each input network be \mathbf{p}^0_t and \mathbf{c}^0_t respectively. From work done on the dynamic pattern network, we would expect that the potentials of the neurons in the next firing set \mathbf{p}^1_{t+1} at time t would be maximum while those of the non-firing set, while being positive, should be less than that of any neuron in the firing set. At each timestep we expect this situation to repeat, so that after a single cycle of L steps, the output pattern p^1 has been expressed by the data network, as required.

Now, assume that the control pattern c_0 has been used to transform a set of input patterns, \mathbf{X} , into a set of output patterns, \mathbf{Y} . Let the number of such associations be n , and include the mapping of pattern p_0 to pattern p_1 . When the network is presented with control pattern c_0 , it will send activity to all of the neurons in the data network with which it has been associated in any pattern mapping. The only way that the data network can decide among its n possible output patterns is by using the input pattern provided by the temporary store to differentiate between them. Thus, for any given input symbol there is already a high degree of correlation between the potentials of the intended firing neuron and those of many of the intended non-firing neurons. Essentially, such patterns have 50% of the firing neuron in common with every other pattern that is the target mapping of a transformation which uses c_0 as the control pattern.

If we assume that the number of control patterns will be much less than the number of data patterns, one remedy would be to reduce the number of active neurons in the control pattern, or even reduce the total number of control pattern neurons. By doing so, the proportion of the potential due to the control pattern would be reduced making it easier for the network to differentiate inputs using the same control pattern. While this might reduce the problem, however, it will not solve it.

The essential problem with the simple association schemes presented so far in the work is that the patterns themselves are uncorrelated. After a fairly small number of patterns are linked using an association, we find that every neuron has

been associated with (and hence made a connection with) every other at some point in the learning process. Using the simple set-to-one principle, this means that every connection has been set to one and the network is entirely saturated.

The next sections presents simulation results to substantiate this hypothesis.

9.6 Simulation and Results

Simulations were carried out using a network of $N = 400$ neurons, with $L = 8$ subsets each of size $D = 16$. Ten control pattern were defined, and learning proceeded by associating a pattern p with pattern $p+1$ using pattern $10 \text{ MOD } p$ as the control pattern. Thus, each control pattern appeared in many associations while each data pattern was used once as an input pattern and once as an output pattern.

Patterns were learned in blocks of ten, with each pattern serving as an input and an output pattern, as described above. After each block of patterns and associations was learned, the network was tested on the entire pattern set that had been learned. Levels of corruption, c , ranging from 0 to $D/2$ were used, the same level of corruption being applied to the control as the data pattern.

In each learning trial, correct recall was assumed if the number of incorrect bits in the output pattern did not exceed $0.25D$, the number of firing neurons, at any timestep after the first two (again, allowing the network time to settle). The graph overleaf shows the storage capacity of the network as a function of the number of pattern blocks learned.

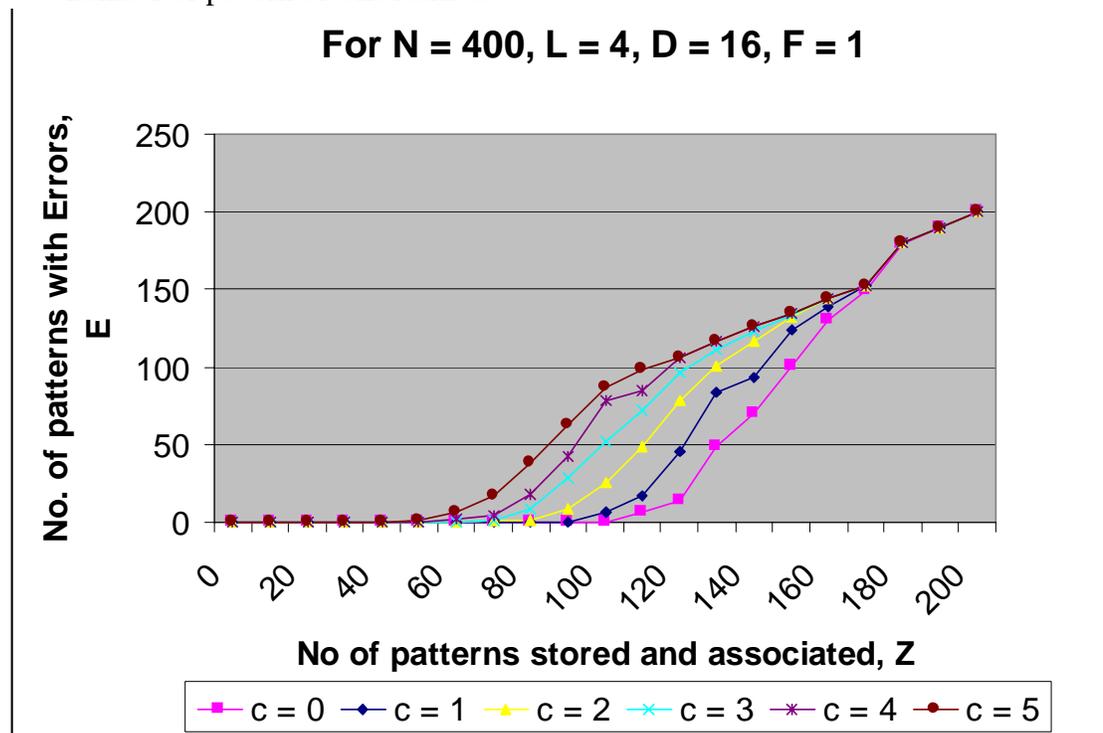


Fig. 9-2 Graph of recall errors vs. storage level for network with control.

We see that for any significant level of initial pattern corruption, c , only of the order of 100 patterns can be stored and associated before the error in recall is greater than the margin $0.25D$. Even so, it should be noted that in this learning scheme, the output pattern is unrelated to either of the two input patterns. Thus the computation performed by the network is not merely to complete a corrupted pattern as was the case in all of the previous networks. Thus, useful work is being done even if the resulting output pattern contains some corrupt bits.

Overall, the network is capable of making an association between two input patterns and a single output pattern with a single presentation of the triplet. The total number of patterns that can be stored in this way is a significant fraction of N the number of neurons in the network. While this capacity is far greater than the $0.13N$ of the Hopfield network, it is still less than the number of neurons, N , and as such is not sufficient to make it a good candidate for the neural building block required by the architecture.

9.7 Discussion

The lack of correlation between any given pair of stored patterns leads to network saturation for a small number ($<N$) of stored pattern mappings. If there were a higher correlation between firing neurons from one pattern to another, this would reduce the number of new connections that were made during each learning event, which we would expect to allow more associations to be made before network saturation.

Unfortunately, the firing subsets for each pattern are established in advance of training and the network does not have any prior information to indicate which patterns are to be associated nor which control pattern will evoke the transformation. In normal operation, we *must* assume that any data pattern can be associated with any other. Thus, schemes which permit the firing subsets of each pattern to be selected with foreknowledge of the associations to be made are not interesting even if they may provide optimal results.

There would seem to be only two acceptable solutions to the problem. First, we accept that the network capacity is limited and must increase the network size N until it is capable of storing the desired number of patterns and associations. For some applications this might be an acceptable solution, but we will proceed on the more reasonable assumption that it is not and that a more efficient implementation is needed.

The alternative solution, and one which is in line with the architectural development of chapter five, is that the patterns themselves must undergo modification to make them easier to store. A subset of the firing neurons in the pattern is replaced with others which are more highly correlated with previously learned patterns and associations. This will reduce the rate of saturation of the network and hence increase the storage capacity.

The disadvantage of this scheme is that the pattern itself is being deliberately corrupted. There is clearly a trade-off between the degree of corruption and the ease of storage that must be evaluated and maintained. This is the subject of the next chapter.

9.8 Conclusions

This chapter has presented the addition of control structures to the simple network which enable the association of data items and subsequent recall of those associations given the appropriate cues. In theory, it represents a possible implementation of the architectural requirements of association between symbol patterns.

Analysis showed that such an approach leads to network saturation even for a numbers of associations less than the number of neurons, N , due to the lack of correlation between associated patterns.

Simulations were carried out on a network of $N = 400$ neurons, using $L = 8$ subsets, each of size $D = 16$. It was shown that after a single presentation of a triplet of patterns (two inputs associated with one output pattern) the network was capable of recalling the associated output even when the inputs were themselves corrupted and the number of associations was of the order $0.25N$.

This result, while not without merit for other less demanding applications, shows that the simple association scheme which has been used so far in the work is inadequate as a candidate for efficient long term storage the neural building blocks needed to implement the architecture.

In the next chapter, methods of increasing the network storage will be considered, based on the extraction of features from the pattern set and subsequent re-interpretation of the patterns in terms of those features. A consequence of this change is that the encodings of the patterns themselves is no longer fixed. This leads to a trade-off between the storage efficiency of the data and the integrity of the patterns themselves.

10.0 Introduction

The advantages of the non-holographic memory of Willshaw *et al.* are the speed with which new memories can be added to the network and the reliability of the network to learn new information “in one shot”. Such a memory was the basis for the first, simple network described in chapter seven. As was demonstrated in the analysis of that network, however, the price that must be paid for simplicity is low storage capacity.

The introduction of dynamic patterns of activity altered the way in which patterns are stored in the network in an attempt to improve network storage efficiency. Yet at heart the principle remains the same: a K-from-N coding scheme with the same properties of one-shot learning, controllability and robustness which stem from the common base in the non-holographic memory.

Finally, the addition of pattern association in the previous chapter highlighted the limitations of the simple association approach upon which the implementation has depended thus far. It was shown that the capacity of such a network was insufficient to make it a good candidate for the neural building block and that a more complex learning algorithm was needed.

To that end, this chapter provides a preliminary introduction to the concept of hierarchies of learning, building on the idea of more complex synapses which was presented qualitatively in the architectural development in the context of addressing the stability-plasticity dilemma (section 3-9). The aims of learning hierarchies are fourfold.

First, to try to increase the storage capacity of the network still further by tuning feature detectors which use higher order correlations between the patterns, with a concomitant increase in storage efficiency. Secondly, to introduce a means of seamlessly integrating memories with a range of persistencies from short-term to long-term. Thirdly, to facilitate generalisation by facilitating *structured* representations for the stored patterns. Fourthly, and perhaps most importantly, to try to

address the stability-plasticity problem which is a major issue in networks which continue to learn in a changing environment.

10.1 Learning Hierarchies: Aims and Issues

10.1.1 Feature Extraction

In the simple network, the storage of each K-from-N pattern (be it static or dynamic) was the result of a first-order correlation between the firing of individual neurons; a connection is made between two neurons when they are both firing for a particular pattern (for the case of static patterns) or when the post-synaptic neurons fires within a short time of its pre-synaptic neuron (for the case of dynamic patterns).

Much of neural network theory, is concerned with the extraction of relevant features from the input patterns (Bishop, 1995). There are a number of reasons for this. First, it is usually more efficient to represent the data in terms of its important features than in its raw form. Also, feature extraction facilitates generalisation by allowing the network output to be generated from an underlying extracted model of the mappings. Refinement of the extracted features corresponds to increased accuracy of the identified model and hence to better generalisation performance.

In a hierarchically organised model, each layer in the hierarchy usually develops a set of features based on the output of the layer below. One drawback with such a scheme for many network architectures is that the different levels take time to develop, over several presentations of the complete pattern set (Rumelhart *et al.*, 1986).

Trying to learn a single new pattern mapping can take many presentations and incremental changes to the weight matrix since the feature detectors at all levels must be changed during learning. Some architectures (such as those based on radial basis functions) use a fixed bottom layer with elements centred on known data points to speed up convergence (Haykin, 1994).

The aim of the adjustments to each feature detector made during learning is to cause the network to be capable of making the given mapping and many others like it. The number of such mappings can (and often does) far exceed the number of neurons in the network. By developing features the network has extracted the underlying structure of the mapping and is able to generate new mappings for previously unseen input data; essentially storing more patterns “for free”.

Thus the development of feature detector can be viewed as a process of model identification which is itself usually a means of reducing the total quantity of data necessary to produce an appropriate response in any given situation by capturing the *state and output generation process* rather than merely the individual outputs.

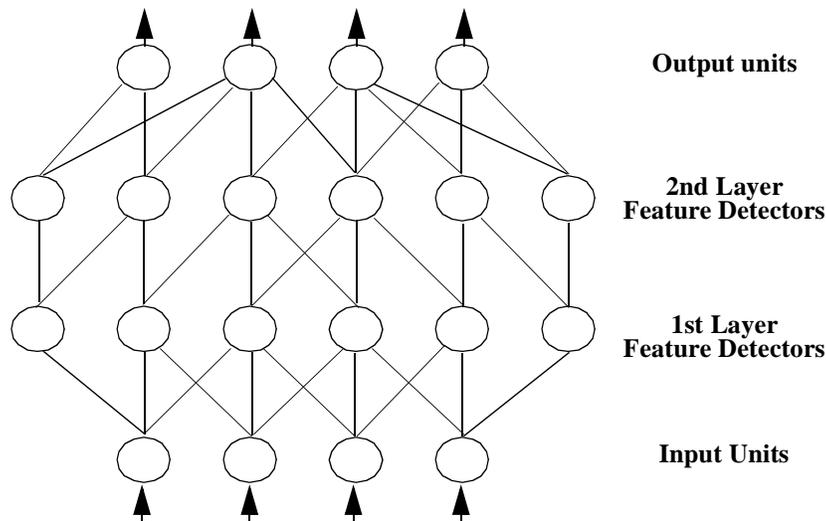


Fig. 10-0 Traditional hierarchical neural network such as an MLP.

10.1.2 Stability vs. Plasticity

The term ‘stability-plasticity dilemma’ was coined by Grossberg to refer to the apparent conflict between the ability of a network to cope with non-stationarity in its environment and its ability to retain knowledge learned in the past (Grossberg, 1988). Networks such as the MLP trained using standard back-propagation suffer from this, as was just described, since it usually takes many learning cycles for the multiple small adjustments to be made to each feature detector to train it appropriately for a new pattern. By making many small adjustments rather than one large one during the learning of a given pattern, the intention is to converge on a feature set which allows the optimal storage of many different patterns and, hence, to try to minimise the disturbance to previously learned patterns.

Both the Hopfield network and basic network described here do not suffer from problems of stability other than as a consequence of saturation (since a learned pattern does not decay) but the price that must be paid for this is the relatively low storage capacity, as has been re-iterated many times here.

Turning to the details of the architecture, we see that any triplet of patterns could be associated at any time. Thus, there would be no opportunity to optimally select the encoding of each pattern based on *a priori* knowledge of the database. Features which were painstakingly extracted for one set of patterns could be disrupted with the acquisition of a single new association. Thus, the stability-plasticity dilemma is a critical issue for this network.

In the architectural development, the evolution of the network during learning was described as the identification of a function which will perform the given mappings. A new mapping, if it brings any new information to the network, must by definition require the network to produce an output which is at odds with that

which would have been produced before the learning had taken place (see the figure below). Ideally, all other mappings made by that function are unaffected, the new mapping forming a discontinuity in the map.

If we require the network to be able to generalise based on this new learning event, then over a period of time the function must adapt itself to integrate the new mapping, distorting the mappings made by similar inputs in the process. This step is clearly an inductive one and the manner in which it is handled is open to many possible interpretations which have a direct bearing of the generalisation performance of the network.

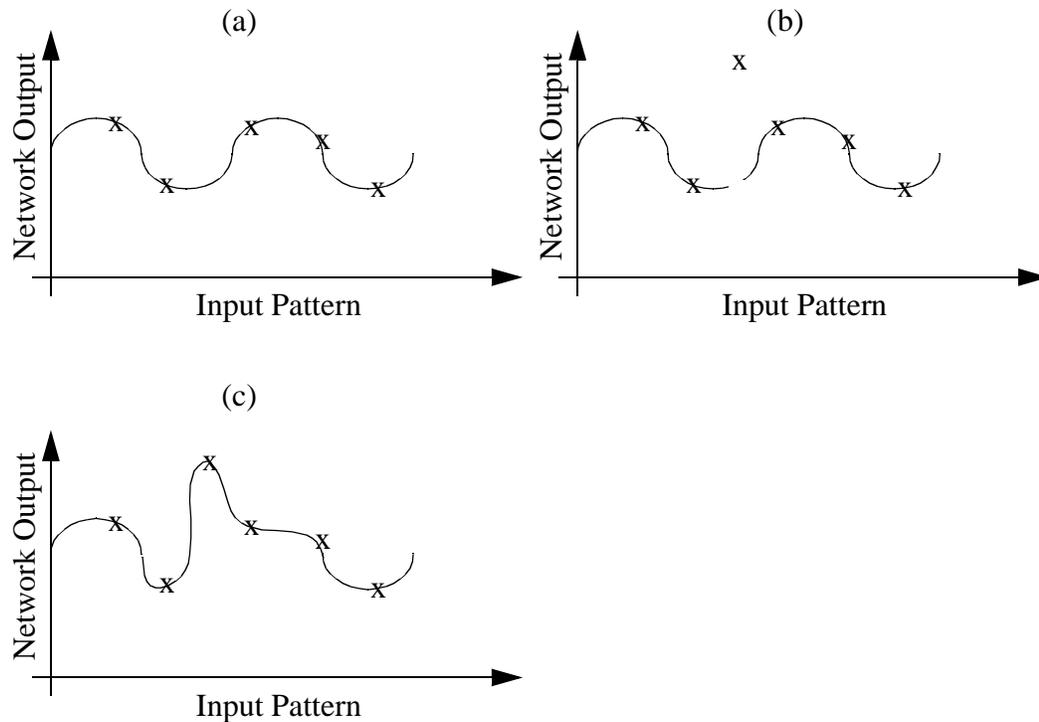


Fig. 10-1 Modification of a function over time to incorporate one new point.

(Note that in this discussion, it is assumed that each new point is correct and that the intention is to learn it. At this stage, we avoid the complexity of handling noisy or potentially erroneous data as well as information which may be required for only a short period before being discarded).

It should seem clear that a major issue with the simple structure of the multi-layered perceptron is that the learning process only admits a single goal: that of updating the feature detectors at all levels simultaneously to slowly converge on a structure which can represent all of the mappings. The failure to form correct mappings early in training is not as important as achieving good performance at the end of the training cycle. Also, each pattern is equally important and has equal persistence. No wonder, then, that the stability-plasticity dilemma is such a problem in the MLP. There is no distinction between old learning which may represent long term features in the data and recent, but highly temporary, data.

An alternative strategy might add additional goals and constraints, potentially leading to much different behaviour. Such a strategy will be presented here and can be summed up as follows. There are two fundamental requirements for the learning algorithm. First, that a learned pattern or association should be accurately recallable even after a single presentation. Second, that data which must be stored for a long duration should be represented in as efficient a manner as possible. The goal is to find a learning scheme in which these two requirements compete as little as possible.

How might such an approach be realised? The network should learn a new pattern in a single presentation using a learning method in which efficiency of storage is secondary to reliability of recall. Subsequent levels of optimisation then *recast* or *re-express* those patterns which will be retained for long term storage within the *same* network but allowing storage efficiency to be an increasingly higher priority. The rest of this chapter will explore the possibilities of this approach.

10.2 Complex Synaptic Structure

As a starting assumption, it was decided that the layered approach to feature extraction would be avoided for the reasons discussed so far in this chapter. Instead, the same regions of N neurons was used to represent features at *all* levels. Changes in the synapses between neurons were made to handle the hierarchical nature of feature extraction. In place of a single valued multiplier, each synapse possessed a set of multipliers, each of which represented a level of persistence in one connection. It remained to be justified that the investment in terms of model complexity would be repaid with a more powerful network in terms of its storage capacity and flexibility.

To begin the explanation of the proposed structure, we first consider the essential characteristics possessed by one synapse. They are: (1) the synaptic weight value, W , which is multiplied by the firing strength of the pre-synaptic neuron and can be modified by learning; (2) the duration, t_s for which the activity is supplied to the post-synaptic cell after the pre-synaptic cell has fired; (3) the rate, α , at which the weight value is changed during learning; (4) the probability, e , that the synapse can have a non-zero value (i.e. that a connection exists).

We could conceive of a network in which the four parameter values could be different for each layer of the network or even for each neuron. In this work, we take the idea to an even lower level of granularity. Here, each synapse possesses multiple parallel ‘units of connectivity’, each of which is a single valued multiplier with a particular value for each parameter. Each synapse then acts as multiple simple synapses in parallel. The figure overleaf illustrates such an arrangement.

The outputs of the parallel multipliers are combined in some way, with the resulting value sent to the body of the neuron for summation with the contributions from the other synapses. Possible options for the ‘combiner’ block include simple

summation and a max function (taking the output of the multiplier with the highest value). These options will be considered in more detail later.

In gross terms, the intended behaviour is that the more volatile synaptic units are capable of capturing a new pattern or association after one learning event, dominating the more slowly varying units whose task is to converge over longer timescales to capture the global characteristics of the data distribution.

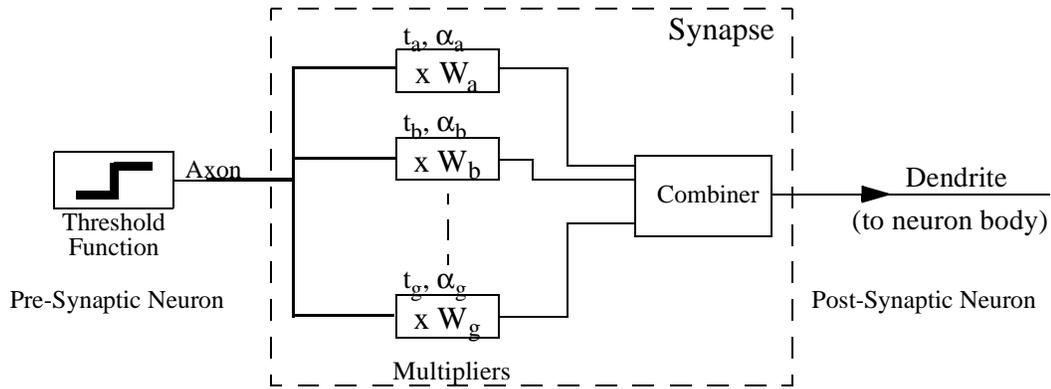


Fig. 10-2 Diagram of the complex synapse studied in this chapter.

Using the four parameters it would be possible to define a wide variety of synaptic behaviours. In this work, the number of possible options will be restricted to supply a small set of allowed synaptic unit types, each of which fulfilling a special purpose during normal operation. We can identify two extremes of synaptic unit behaviour; these extremes characterised the whole concept of learning hierarchies. By interpolating between the parameter values of these extremes, we can produce the other cases.

At one extreme we would like a synapse which facilitates one-shot memorisation of a pattern, as described in the simple and dynamic networks. Such a synapse will have a large learning rate, α , to ensure that only one learning trial is required to transform the weight value from low to high.

Since we want to ensure learning in a single trial, we ensure that a high proportion of synapses has such a unit; thus the probability, e , that a connection can potentially exist is high (perhaps equal to one).

At the other extreme, we would like a synapse which represents global features of the data, extracted over many learning events. It should be relatively slow to change to prevent new learning disrupting long held experience. For reasons that will become clear later on (section 10.3.2, page 268) the probability, e , that a given connection might exist may be much less than one.

Allocating the other two parameters to these extreme synaptic layers offers us the opportunity to consider two alternative scenarios. Notice that, in terms of magnitude, a given layer could dominate the potential supplied to each neuron either by virtue of having a long weight vector W , or a large duration of activity t_s .

The former case corresponds to a high mean strength of connection between neurons via that particular layer, while the latter corresponds to continued activity long after the firing (pre-synaptic) neuron has entered its refractory period. To prevent one layer dominating all others, we could arrange for something akin to inverse proportionality to exist between the value of W for that layer and its value for t_s . The product Wt_s would then be approximately constant. It remains to choose whether it is W or t_s which increases with increasing synaptic layer. Let us consider both options.

Option 1: Increasing weight vector magnitude with level

Here, the fast (level a) synapses have the shortest weight vector. Since it is important for the firing of these synapses to dominate the neuron potential when required, we compensate for this short vector using a large t_s . Once the synaptic a unit is stimulated, it continues generating potential for a long time relative to the refractory period of the neuron.

By contrast, the highest level (level g) synapses have the long weight vectors. These vectors represent the long term history of the neuron's learning and do not change rapidly. To prevent previous learning dominating the neuronal potential, we insist that such synaptic units, once activated, produce potential for only a short time, that is to say they have a short t_s .

The overall philosophy with this option is for the high-level g synapses to dominate the underlying potential on a cycle by cycle basis. The short activation time of each individual synaptic unit allows a fine grained pattern of potential to be maintained. This pattern changes only slowly with learning, since the learning constant, α , is low. Transient modifications using the level a synapses must compensate for lack of individual strength by force of number. As will be presented later, the activity vector 'seen' by each neuron is dominated by the synaptic level which remains active for the longest time.

Option 2: Increasing duration, t_s , with level

In this alternative approach, the low level, a synapses have the longest weight vector but each synapses is active for only a short time. Thus learning not only makes a large percentage change to the level a weight vector, but the absolute magnitude itself is also large.

By contrast, the high-level g synapses are not only slow to modify during learning but, once activated by pre-synaptic firing, remain active for the longest time. The short weight vector of these units prevents them from dominating the network potential. These units therefore represent both long term trends in learning and provide a slowly varying background level of potential during all aspects of processing.

Between the two extremes of synaptic type, from level a to g , we define a range of synaptic parameter values with decreasing learning rate, α_i and probability of existence, e_i . The trends for W_i and t_i depend on the option chosen.

The values for these parameters and the number of different units in the range would be determined empirically based either on a specific environment or generically (i.e. selected as a compromise over a range of environments).

Choice of Option

For the remainder of the work carried out for this thesis in the area of learning hierarchies, we choose option 2 for the allocation of parameters for W and t_s . Thus, the level a synapses have the shortest weight vector but are active for the longest time. This choice reflected the belief at that time that the best use of the higher level synapses was to capture global features of the data. These long term trends could act as a reference for all other activity so that each layer beneath level g would add to its output in a form of *delta encoding*.

[Simulations (to be presented later in this chapter) will show that this philosophy has not yet been shown to be effective. It is possible choosing option 1 at this point might have yielded better results since one of the potential dangers of option 2 is that the weight vectors might slowly converge to averages of all of the input vectors. The benefits of both options remain to be proved by future work.]

For the sake of clarity, we present the trends to be used for the parameters W and t_i :

$$0 < t_a < t_b < \dots < t_g \quad \text{but} \quad W_a > W_b > \dots > W_g > 0$$

During normal operation, new learning events would initially impact only the a units within each synapse. Knowledge encoded only here must soon fade to prevent network saturation. A mechanism must exist to consolidate the new data if it is to be retained for more time than is granted by short-term storage. Options for these mechanisms and the issues involved are discussed later.

Note on nomenclature

In the discussion to follow, synaptic units will be given letters from a to g . Unit a has parameters for short-term memory, while unit g is tuned for long term storage. The synaptic units of a particular level (say a) will be referred to, variously, as *unit a synapses*, *a units* or *level a synapses*. The meaning is the same in each case.

10.2.1 Options for the Synaptic Combiner

The model for the synapse is made up of a number of multipliers in parallel, feeding into a combiner that produces a single value. This value is conceptually propagated down the dendrite to the neuron body for summation with the contribu-

tions from the other synapses. Two possible models were considered for the combiner unit. The simplest is the sum of the outputs of the individual synaptic units. If the output of unit i is designated O_i , then the output of the synapse, $O_s(t)$ is:

$$O_s(t) = O_a(t) + O_b(t) + \dots + O_g(t)$$

When the pre-synaptic neuron first fires, the synapse output will consist of the sum of the outputs of each unit. As time passes, one by one, beginning with unit a , the output of each unit will fall abruptly to zero. This is illustrated below.

The alternative choice is to select the unit which produces the highest contribution to the potential. Here the synaptic output is given by:

$$O_s(t) = \max(O_a(t), O_b(t), \dots, O_g(t))$$

For this *max* scheme to work, we must ensure that the weight value of the shortest duration output (unit a) has the *potential* to become the largest (even though in any particular case it may not be the largest) and so on down the line, with the longest duration output (unit g) having the lowest mean value. If this is not that case, then certain units could never win the competition and their value would always be ignored.

However, if the constraint *is* met then in response to pre-synaptic activity unit a will win first and send potential W_a to the neuron body. But since its activity occurs only for a short duration, its output will drop to zero and unit b , which had the second highest output value, will take over. The synaptic output will now be reduced to W_b . Over time each unit will have its turn until finally (long after the pre-synaptic neuron has fired and gone silent) the long duration unit g will be the only one still active and will have its turn as the provider of the potential. When it has gone silent the synapse will send no more activity to the neuron body until the recurrence of pre-synaptic activity.

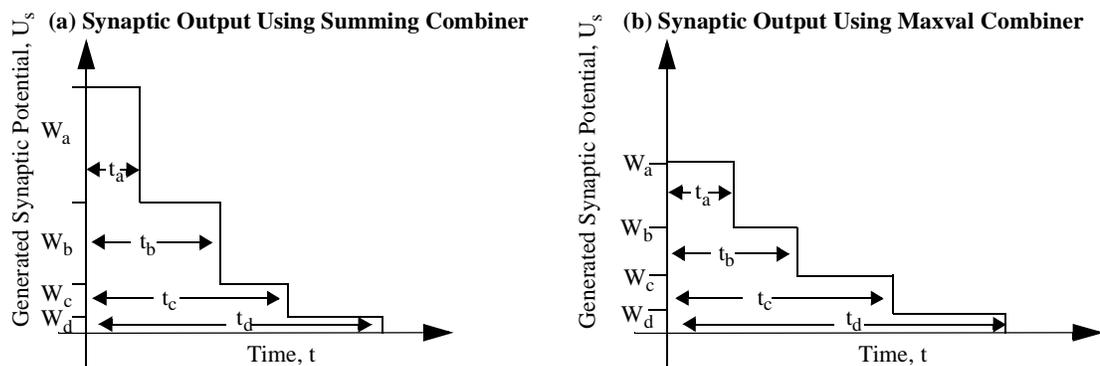


Fig. 10-3 Graphs of synaptic potential against time for two combiners.

One aspect of the summation combiner is that the weight values at all levels have some influence on the potential of the post-synaptic neuron at all times. For the max combiner this is not the case: only a single synaptic unit influences the synapses contribution to the potential. This may have an impact on the learning

algorithm, as will be discussed later in this chapter. As a result, in subsequent analysis the combiner that will be used produces the linear sum of the contributions of the individual synaptic units.

10.3 Options for the Weight Vector Components

This section will consider the breakdown of the weight vector \mathbf{W}_i for each neuron, i , establish properties which depend upon certain parameters of this breakdown and consider some of the trade-offs that could be made in a range of different implementations. We begin with some definitions. First, we write the weight vector as the sum of several independent vectors, one for each set of synaptic units. This is valid since we are adopting the simple summation as the synaptic combiner. Thus for neuron i :

$$\mathbf{W}_i = \mathbf{W}_{ia} + \mathbf{W}_{ib} + \mathbf{W}_{ic} + \dots + \mathbf{W}_{ig}$$

where a to g are the indices of the synaptic units. Note that the actual number of levels within each synapse is yet to be determined. For now, a to g will be used as the default range.

Next, let the length of each contribution to \mathbf{W}_i be l_m , where m ranges from a to g . The definition of l_m is based on the sum of the absolute values of the vector components (in the range $j = 0$ to $j = N-1$). For neuron i :

$$l_{im} = \sum_{j=0}^{N-1} |W_{ijm}|$$

allowing the vector to be approximated as shown in figure 8-7, below.

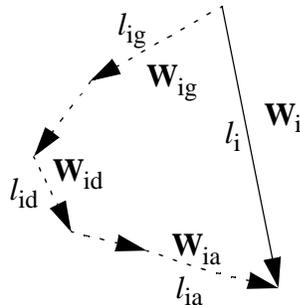


Fig. 10-4 Weight vector for neuron i in terms of components for each synaptic unit.

To allow the dynamics of the network to be controlled as before, we insist that the values of l_m be fixed for each m . This will act as a target for each synaptic unit.

Finally, note that once triggered by pre-synaptic activity a synaptic unit, m , will generate potential for t_m cycles, as discussed earlier in this chapter. Once again, the value of t_m is independent of the neuron index i .

The potential of neuron i at time t will be given by the following equation:

$$U_i(t) = \sum_{m=a}^g \sum_{d=0}^{t_m} [W_{im}^T \cdot \bar{O}(t-d)] - T_i$$

where the two sums range over the synaptic unit types and the time for which they are active. The vector $\bar{O}(t)$ is a record over time of the network activity and is made up of '1's (for active neurons) and '0's (otherwise).

In figure 8-8, the network state over several iterations is shown for a simplified 1-from-8 encoding, together with the output history as seen by three different synaptic units, $\bar{O}_1(t)$, $\bar{O}_2(t)$, $\bar{O}_3(t)$. The index shows the number of cycles for which the synaptic unit is active once stimulated. We see that the effective input vector as seen by the synaptic units of long duration is the sum of the output vectors over several cycles, and is increasingly full of '1's.

Time	Network Output	Output History at time t	
t	0 0 0 0 0 0 0 1	$\bar{O}_1(t)$	0 0 0 0 0 0 0 1
t-1	0 0 0 0 0 1 0 0	$\bar{O}_2(t)$	0 0 0 0 0 1 0 1
t-2	0 0 1 0 0 0 0 0	$\bar{O}_3(t)$	0 0 1 0 0 1 0 1
t-3	1 0 0 0 0 0 0 0		

Fig. 10-5 Output history vectors as seen by several different synaptic units.

Ignoring the threshold term for the moment, the potential of a neuron due to each synaptic unit, $U_m(t)$ is essentially a function of four terms:

$$U_m(t) = f(W_{im}, \bar{O}, D, t_m)$$

respectively the m th component of the weight vector, the network activity vector, the firing subset size and the time of action of the synaptic unit itself, once stimulated. We see that the maximum contribution that each type of synapse can make to the potential of neuron i is:

$$\max(U_m(t)) = S l_m t_m$$

where l_m and t_m are the vector length and activity time for synaptic unit m , respectively, and S is the number of sources of stimulation (which is two for the case of one external stimulus together with the internal connections between the N neurons of the network).

We note that if connections made with synaptic a units are to be capable of dominating the neuron potentials and dictate which neurons will fire after a single learning event (which was the case for the simple and dynamic networks for low enough storage, Z), this implies that for a given neuron the potential coming from

a units must be able to surpass the sum of the potentials from the non- a units of any other neuron, j . In other words for a neuron i :

$$\max (U_{ia}(t)) > \max \left(\sum_{m=b}^g U_{jm}(t) \right)$$

for *any* neuron j . Were this not to be the case, previous learning in an arbitrary neuron j could permit it to have more potential than neuron i even immediately after learning a new pattern in which neuron i is intended to fire at time t . This would prevent neuron i from firing, corrupting (and possibly disrupting) the pattern.

The inequality translates into the following expression for a single pattern stored in unit a synapses, in terms of the activity time and maximum vector length:

$$S \sum_{m=a}^g (l_{jm}t_m) < SB_aD$$

where B_a is the strength of a made connection for a unit a synapse, D is the firing subset size, and S is the number of sources, as defined earlier. Clearly, this argument can be repeated for the case where a pattern stored in unit b synapses (unit a weights having decayed) must be recallable in spite of previous learning in levels c and above. For a pattern stored only in level n synaptic units:

$$S \sum_{m=n+1}^g (l_{jm}t_m) < Sl_{in}t_n$$

which essentially means that the product $l_{in}t_n$ must decrease for higher values of m such that the value at level n is greater than the sum of the values for all levels above it. If we assign the arbitrary value '1' for level g , then it is simple to construct a table of values for the other levels which would fulfil this requirement:

Table 10-0 Relative weight vector component dominance for different synaptic levels.

Level	a	b	c	d	e	f	g
Product, $l_n \times t_n$	64	32	16	8	4	2	1
Justification	$64 > 32 + 16 + 8 + 4 + 2 + 1$	$32 > 16 + 8 + 4 + 2 + 1$	$16 > 8 + 4 + 2 + 1$	$8 > 4 + 2 + 1$	$4 > 2 + 1$	$2 > 1 + 0$	$1 > 0$

Thus for a seven level system (as given above) using a base value of $l_g t_g = 1$, the unit a synapses, would need a $l_a t_a$ of value 64 times as great as a unit g synapse to guarantee one-shot learning, in the worst case.

There are a number of caveats to this result. First, if we relax the simple but potentially over-strict requirement of one-shot learning even in the pathological and improbable situation presented above, this multiplier value can be reduced for *all* levels. To do so requires more complex analysis of the probability distribution of the component vectors and any result would carry with it a probability of error since we cannot guarantee that such a pathological situation will never occur.

Second, we should reconsider the assumption that the network must assimilate each pattern such that it will *always* be perfectly recalled as it is consolidated. With such an assumption, a learned pattern would always be consolidated so as to become a permanent part of memory. But we might not intend this level of storage for every pattern. Here we consider two alternative schemes.

First, we might desire a system in which it is initially important to memorise a pattern after one presentation and be able to recall it perfectly for a short time thereafter, but as it is slowly assimilated into the long-term memory, we do not require such perfect recall.

In another scenario, we might use frequency of presentation as an indicator of importance for long-term storage. Thus, a pattern presented only once may be initially learned but soon forgotten, leaving only a minor trace in the higher levels of storage which is insufficient to recall the whole pattern. Repeated presentation of the same pattern, perhaps widely separated in time, indicates relevance to the system and should lead to an accumulation of the small traces in higher level memory to the point where it can be recalled using them alone.

In both presented scenarios immediate and reliable recall is needed after a single presentation but several pattern presentations (perhaps with time between each for individual consolidation) would be necessary (and even desirable) to initiate long term storage. Consider the alternative table below.

In this example, for level *d* the product $l_d t_d$ has been set to be less than the sum of the products at the levels above. Patterns reaching level *d*, with no support from levels *a* to *c*, may not be stable even for low storage since the contributions to the potential from these synapses may not be able to overcome that of the synaptic units on levels *e* to *g*.

Relaxing the constraint for level *d* has allowed the relative $l_x t_x$ product of each of levels *a* to *c* to be reduced while still leaving them stable after one-shot learning, however. This can be seen from the fact that the products for levels *a* through *c* are still larger than the sum of all products in the levels above them. This example illustrates the flexibility available to the designer: to control, in a very general way, the information preserving-power of each level of the learning hierarchy.

Table 10-1 Modification of l_t product for level d leads to reduction in max. product size.

Level	a	b	c	d	e	f	g
Original Product, $l_n \times t_n$	64	32	16	8	4	2	1
New Product, $l_n \times t_n$	56	28	14	6	4	2	1

Exact knowledge of the learning algorithm itself is needed to refine the trade-offs that can be made, although even in the very high level analysis presented thus far some of the inter-related parameters that are under the control of the

designer when using the hierarchical approach to learning are becoming clear: First is the maximum speed of learning which still facilitates accurate recall. Second is the relative dominance of newer learned patterns over old.

Next, the rate at which learned data can become a permanent part of memory. Finally, the point (or points) in the consolidation process where information learned ‘one-shot’ begins to lose its recall sharpness (blur), requiring multiple presentations to facilitate sufficient reinforcement for permanent storage and recall.

The analysis of the learning algorithm will quantify these trade-offs.

10.3.1 Balanced Contributions from Each Neuron

One of the central tenets in the development of this network has been that each neuron is essentially identical: the *a priori* probability of firing for any neuron in the network is the same and each should contribute the same amount of information to the network output. This idea will be carried forwards into the weight vector itself.

We could envisage a network in which the weight vector of one neuron had all of its sub-vectors aligned so that for certain patterns its potential was very high while for the majority of the time the potential would be very low for the same reason. Such a neuron would always tend to win in a competitive environment, a situation which often leads to misrepresentation of the probability distribution which underlies the pattern set. In addition, such a situation is not in keeping with the philosophy of one neuron participating in the active set for many different patterns.

One way to avoid the dominance of one neuron is to prevent the individual sub-components of its weight vector either from growing without limit or from aligning. Limiting growth can be handled with the periodic normalisation of the vector. Prevention of component alignment could be handled by inhibiting the development of a weight vector at level k when there is also a significant contribution to the potential from the weight vector at level $k+1$. Such a scheme would allow only every second component of each weight vector to align. By extension, a scheme which prevents the development of a vector at level k when it will become too close to the weight vector on levels $k+1$ or $k+2$ would allow only every third level to align. Examples of this are illustrated below.

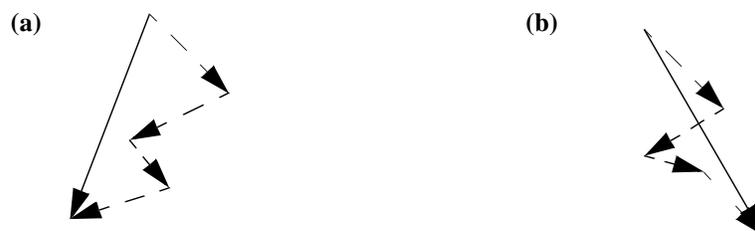


Fig. 10-6 Weight vector components with restrictions on development.

The effects of such restrictions on weight vector development are twofold: first, each neuron is more able to respond to a range of different input vectors since it is not permitted to commit all of its weight vector components to a single direction (leading to a neuron which responds significantly only to a single input vector). This is beneficial in the development of fully distributed representations where we wish to avoid specialisation of individual neurons with respect to an individual pattern.

Second, the potential of each neuron is less subject to fluctuations in response to any given pattern of input. The drawback with this scheme is that it places extra constraints on the learning algorithm in two ways. First, to evaluate a change in a synapse of a level k unit, the algorithm must consider the component values at level $k+1$ and so on, increasing its complexity.

Secondly, in its strictest sense the algorithm is no longer local to one synapse since the effect that one level has on another is in terms of the similarity of the weight *vectors* not the individual *components*. Nevertheless, the algorithm would still be local to each neuron and so retains some degree of 'locality'.

10.3.2 Uneven Distribution of Weight by Synaptic Unit Type

As discussed in the previous chapter, the association of uncorrelated patterns makes it equally likely for any neuron to make a connection with any other, leading to saturation. If it could be arranged that certain connections are used with higher frequency than others, the saturation of the network could be delayed because the infrequently used connections would decay to zero and the highly used connections would grow stronger.

Depending on the algorithm used, a highly used connection might occur between two neurons which are highly correlated over a number of different patterns. This is usually the case in standard competitive learning, as described in chapter two. Thus, what is sought is a correlation between firing neurons, even though the arbitrary association of *a priori* uncorrelated patterns would suggest that such correlations do not exist.

To reconcile these two apparently contradictory requirements, it is necessary to modify the pattern during consolidation such that the correlations between neurons is increased as the pattern is assimilated. Thus, the short-lived connections using a unit synapses have an equal probability of occurring between any pair of neurons due to the uncorrelated patterns which are associated. Each successive layer of synaptic units, however, demands higher and higher levels of correlation between firing neurons. The synaptic g units, at the extreme, must have weight values which are significant only for a small subset of the possible connections. This is illustrated overleaf.

In the graph, the values of all of the weights for a single neuron are shown. The x-axis is the weight number, corresponding to the index of the neuron to which the connection is made. The y-axis is the value of the weight. The target distribution for the weights of each synaptic unit is shown. Note that the order on the x-

axis is different for each synaptic unit, so there is no constraint which insists that the highest value of the g unit synapse and a b unit synapse occur in the same physical connection.

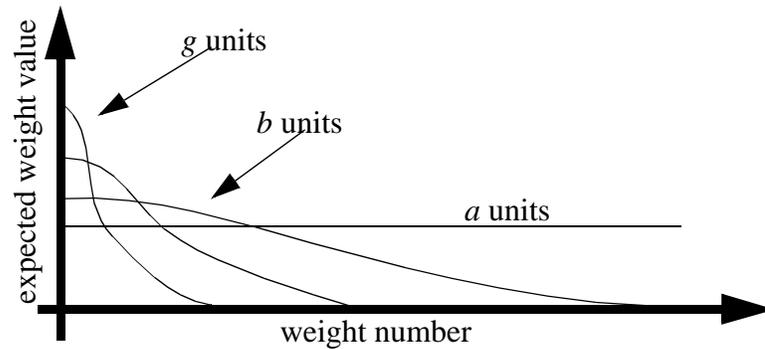


Fig. 10-7 Desired distribution of weights of different synaptic units for each neuron.

What this graph illustrates is that the a unit synapses are equally likely to occur between any pair of neurons whereas we aim for g unit synapses to be limited to only a small number of connections, leading to a higher correlation of activity between those neurons. How this might be achieved will be considered in the exposition of the learning algorithm later in this chapter. The implications for an general algorithm are discussed next.

Note that the parameter e that was specified earlier (section 10.2, page 258) is intended to quantify the concepts described in this section. A low value of e , as possessed by a level g synapse, would correspond to a low probability of the existence of a connection.

10.4 Implications of Creating Correlation Between Neurons

The last sub-section discussed the correlation of firing between neurons during pattern recall or association. Since a pattern or association may connect any neuron with any other with equal probability, this leads to saturation of the network, as was shown throughout chapters seven, eight and nine.

It was stated that by re-using connections that were already existing, the network could delay saturation. Higher level synaptic units would become increasingly polarised so that, for the highest level units only a small subset would have weight values significantly above zero.

For the values of a subset of synaptic connections to dominate, however, there must be an uneven probability of a particular connection being activated. Assuming a Hebbian-like learning algorithm, in which concurrent activity of the pre- and post-synaptic neurons leads to strengthening of the connection between them, then activation of a connection will tend to strengthen it still further and the positive feedback thus established will ensure that the inequality is maintained and

increased. In fact, this is the first principle of self-organising systems, as defined by von der Malsburg (in Haykin, 1994, p353).

Such a scheme brings consequences both for the learning algorithm and for the operation of the network as a whole, however. For a learning algorithm based on Hebbian learning the probability of the activation of a particular connection between neurons must differ from one connection to another despite the fact that the patterns and associations are *a priori* uncorrelated.

To deliberately re-use connections in a situation where the pattern was created independently of existing network structure, the units which fire to encode the patterns must mould the pattern to the existing order of the network: changing the firing subsets to use neurons whose connections are already better suited to activating the firing subset at the next timestep than those current in the pattern. Hence the pattern (or at least a fraction of it) is maintained with minimal changes to the current connections. The figure below illustrates this using three timesteps of a network region.

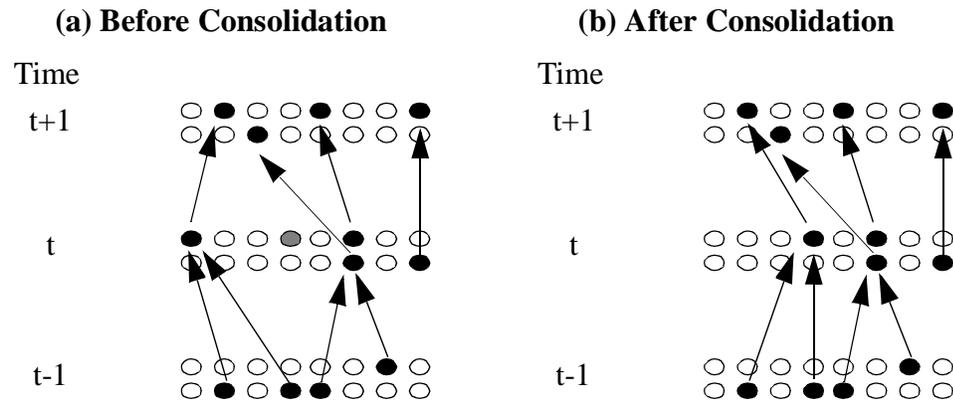


Fig. 10-8 Modifications to the pattern during consolidation make it easier to store.

In figure (a), the firing subsets (shaded) change from one cycle to the next. One neuron (light shaded at time t) has a potential near to firing but is not part of the firing subset at that time. During learning, the algorithm should seek to locate neurons such as the lightly shaded one which are near to firing and can be made to fire with minimal changes to their weight vector. In doing so, one of the firing neurons in the subset at time t falls silent and is replaced.

How the replacement neurons are located is another issue. Noise could be one method, executing many complete cycles of the pattern adding noise of zero mean and fixed variance at each timestep. Those neurons close to firing would do so with much higher frequency than those whose are far from firing. An alternative scheme might allow the level *a* synapses to decay slowly in value: while initially the level *a* synapses alone are enough to dominate the potential of each neuron, as they decay the underlying (i.e. existing) pattern of weights would become more significant and increasingly we would expect to see neurons which should not fire

for the current pattern begin to do so. The learning scheme must profit from the information contained in the changing firing sets to update the weights appropriately. This latter scheme will be discussed in more detail later.

The consequences of modifying the pattern during consolidation are clear: the pattern itself is distorted and is therefore less able to act as an input cue for all of the patterns with which it is associated. There are a number of cautionary points to make, however. If consolidation is occurring at the creation phase of the pattern, then there are no associations in which it is involved, and therefore no penalty if it is modified. However, in normal operation as described in the architecture it is defined as normal practice to create a symbol and then to use it immediately afterwards. This symbol might exist, for example, only at levels a and b of the synaptic hierarchy when first put to use in temporary associations which might become more permanent with the progress of time. The objective at that point would be to consolidate the pattern (with possible changes in its firing subsets, as just discussed) while maintaining the associations already made.

Two solutions are readily forthcoming and ostensibly plausible. First, if all of the associations that the pattern makes are exercised often enough, then each time one is used the connections made during its association could be modified to track the distortions to the pattern. This may be feasible for some applications but for a large memory system with a lot of associations, the probability of exercising all of them without doing so deliberately seems low and somewhat artificial and restrictive.

The alternative solution is to ensure that the modifications are *aligned* in such a way as to preserve the mappings rather than scramble them. Now we are trying to meld the concept of the re-use of connections by creating correlations with the concept of feature extraction (in which common associations between patterns are picked out and new patterns and associations are made in terms of those). While the latter is a process of identification (or existing correlations between patterns), the former is one of creation (deliberately modifying patterns to make them more similar).

As a final point, we note that the problem of the evolving symbol encoding to allow inheritance and generalisation (presented in the context of the architectural development) is a highly related issue and this is no accident.

10.5 Options for Component Interdependence During Learning

During the learning procedure, the individual components which make up the weight vector of each neuron will be updated according to a prescription to be analysed in a later section. While updating level k in the synaptic unit hierarchy, several options remain for the participation of the other levels in the calculation of neuron potential. These options are considered first.

10.5.1 Ordered Sequential Dependence

In this scheme, the levels below that which is being consolidated are disabled. Thus all contributions to the neuronal potential comes from levels k and above. This is shown in the figure overleaf for two levels.

The shaded section of each pyramid shows the levels of synaptic units which are active (generating potential) during learning. Although the levels not included in this set are disabled, the weight values themselves are not affected. However, they are inactive and thus contribute nothing to the neuronal potential.

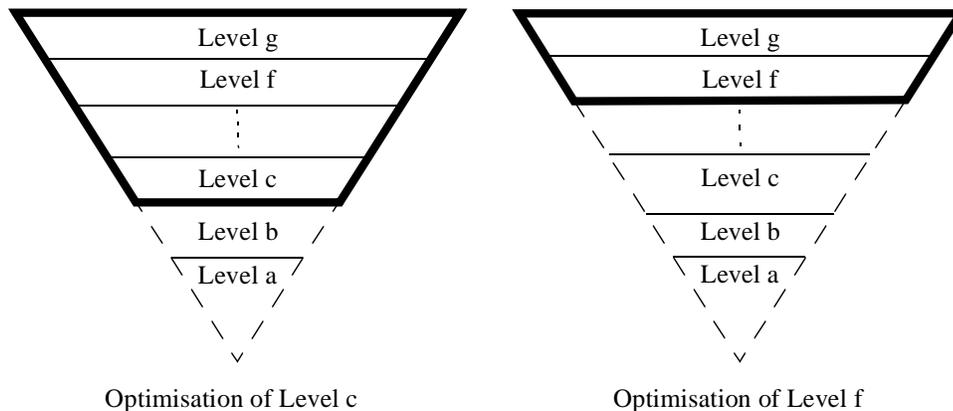


Fig. 10-9 Synaptic unit optimisation using ordered sequential dependence.

What effect would this have on the consolidation of a pattern at level k ? When searching for other neurons to fire and take over the pattern, potential is generated only by synaptic units at levels k and above. Thus, the network is searching for stable features which have been learned by these higher level synapses with the intention of representing the pattern being consolidated *in terms of them*.

By virtue of the lower learning rate, α , for the units at these higher levels, we know that the synapses at levels $k+1$ and above will change little over timescales which will see large changes in synapses at level k . Thus, to all intents and purposes the ordered dependence scheme should provide a stable set of features with which to encoded more volatile patterns.

There are two principle disadvantages of this scheme. First is that a special network mode is needed during consolidation which begins with all of the synaptic units enabled but then disables them level by level, each time consolidating the lowest active level into those above it. The optimum frequency of this special mode and what parameters might govern its operation are also new unknowns.

Second is the emergence of a property which is common to most simple neural networks (including the Hopfield) but which is a source of problems when such networks are applied to symbol structures: the inability to prevent the involvement of a stored memory in recall when it is not relevant.

Consider a basic Hopfield network during recall, given a corrupted version of a stored memory. At every update, the accumulated memories of *every* stored pattern influences the choice of the next state for *every* neuron. This occurs whether or not the memory is relevant to the current computation. In a sense, the memory acts as a monolithic slab and is incapable of breaking its stored data down into a number of independent sets of patterns with only one such set affecting the updating of the network at a time.

By extension, we see that using the ordered sequential update scheme, the modifications to the longest-term synaptic units also occurs outside of the context of any lower level partitioning. Thus the ordered scheme suffers from the same problem as the Hopfield network: the long term behaviour of the g unit synapses is in some way an average over all patterns learned by the network and they have an unconditional impact on all network activity.

10.5.2 Unordered (Parallel) Dependence

In this scheme the individual levels of the synaptic units are updated together, with each contributing in parallel to the neuronal potential, just as in normal network operation. When noise is added to the potential of each neuron to provoke activity from otherwise non-firing neurons, all components of the weight vector contribute to making the firing event occur.

The benefit of this scheme, in contrast to the ordered one, is that no special network state is required. The updating could occur in parallel to normal network operation.

One possible disadvantage is that changes made to high level (long-term storage) synaptic units are made in the context of particular lower level (short-term storage) features. The fact that this may constitute a problem can be illustrated by example. We could envisage a situation in which a positive change in a level g synaptic weight value might be made when the neuron fires even though the majority of the potential in that neuron comes from a recently acquired set of unit b weights.

Soon thereafter (on the timescale of a significant change of the unit g weight) the short term record held in the unit b weights has been significantly modified and the earlier change to the unit g synapse is both insufficient to recall the original pattern and serves little or no purpose in the recall of any others.

The basis of the problem is that the long term connections are being consolidated in terms of shorter term, more volatile features. Building on such unstable foundations would not seem to be a sensible principle. This apparent disadvantage may prove to be illusory, however. If the changes to the synapses at lower levels are uncorrelated, then any minor fluctuations that they create in the higher level weights should be smoothed out.

Furthermore, there may be positive correlation between the lower-level synaptic units and the higher which is in fact *due* to the high level units. In that case the positive feedback during learning will tend to make large weight values get

larger, a central mechanism in competitive learning. Thus, the apparent disadvantage of optimising all of the levels in parallel may be turned to an advantage. To understand this case properly, detailed analysis using stochastic processes is probably required, and is left for future work.

Finally, we note that the very fact of *not* relying on more volatile synapses was quoted as a disadvantage for the case of ordered, sequential dependence since it leads to a monolithic memory. Is there a middle-ground alternative solution?

10.5.3 Ordered Non-Sequential Dependence

This consolidation scheme combines the other two, seeking to form stable memories and, also, to allow subsets of memories to form, breaking up the “monolithic slab”-like memory described earlier. As in the ordered, sequential scheme the layers are optimised one by one starting from the a units, but one low level (b in the figure below) is always involved in consolidation.

The effect of the level b synapses is to act as context for the recall of more permanent information in the higher levels. We could imagine the pattern stored in the b units as a subject area or category, such as peoples names or the capital cities of countries. Using the subject pattern as context, the consolidation of another pattern (“Fred” or “La Paz”) at level c or above occurs, the choice of neurons to carry the pattern being influenced by the context (“names” or “cities”). A short while later, the subject pattern is forgotten and the associated memory, lacking part of the weights which were involved in its consolidation, is more difficult to retrieve.

But being more difficult to retrieve is not entirely a disadvantage: a pattern which has lost some of its support set will have a lesser impact on the recall of other patterns. To a certain extent, the consolidated pattern lies “dormant” in longer term storage; its effect on ongoing computation is much reduced but it is not erased, by virtue of the slowly changing synaptic units which encode it.

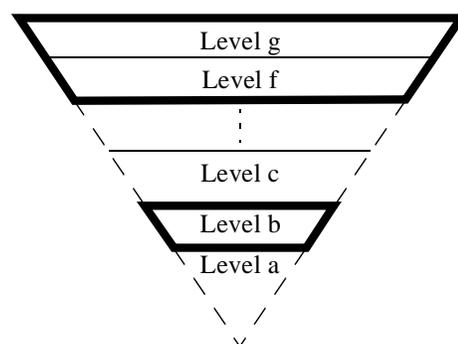


Fig. 10-10 Synaptic unit optimisation using ordered, non-sequential dependence.

How would recall be achieved? When the network decides to recall something which it knows to be a capital city (“Query: what is the capital of Bolivia?”), it only needs to re-establish the b unit connections for that subject pattern (“cities”). Doing so re-establishes the environment in which the high-level synaptic units learned the desired pattern and others of that class. The network can then

retrieve the correct response given an appropriate input cue using a mechanism such as was presented in the last chapter.

Overall, the use of short-term memory to store subject patterns would permit a single memory system to be accessed as if it were many separate memory systems by a technique of advanced priming.

But where does the network store the subject/category patterns? The easy answer is that they are stored in some other memory and retrieved by the control network. While this is a plausible option in terms of implementability, for an autonomously learning system it is somewhat fraudulent since it is essentially “self-bootstrapping”. How does this separate memory store its patterns? Does it group its subjects into higher level subjects? If so, are these stored in a separate memory?

A more integrated solution would include the set-up of one subject class based on the retrieval of an earlier pattern (or patterns) from the *same* memory. This could occur in many ways, two of which seem particularly useful. The first allows the control network to retrieve the subject pattern explicitly and re-consolidate it as an explicit priming step. An alternative and more subtle solution allows every recalled pattern in a particular class to complete itself when it has missing or erroneous elements, followed by short-term consolidation of those missing elements. This completion could be used to re-establish the class context in short-term memory and act as a boost to recall of patterns in the same or a similar category.

To use the human analogy, recall of an almost forgotten memory could be achieved by thinking about other related events/objects until enough context has been established to complete the desired pattern.

In this work, the usage of short-duration patterns as a means of defining sub-memories will not be pursued further, although it is noted that such an investigation would form a valid area of further study. Instead, the unordered (parallel) dependence learning mode will be adopted as a demonstrator.

10.6 A Note on the Generation of New Symbols

Throughout the development of the neural building blocks, the patterns were selected at random, with zero correlation between the selected firing neurons. In consolidating each pattern there was a conflict between the required output and the actual output which was resolved by allowing a fraction of the pattern to be corrupted at each level of assimilation.

We contrast the network view of the patterns with the architectural view: as symbols. In chapter five, the creation of new symbols was highlighted as an important element of the architecture. It was stated that a symbol encoding must incorporate not only a means of accessing the expression that it represents but also

properties of that expression which can be used in computation without accessing the symbol contents. To ensure that each symbol is unique a quantity of noise was postulated: shifting some of the '1's at random.

In spite of the noise, the symbol patterns will not be random. Each must be constructed using some procedure (as yet undefined) which will imbue it with the necessary properties to fulfil its function. Noise would, therefore, reduce but not eliminate in-built correlations between existing connections and those which are necessary to store the new symbol. Further analysis of the consequences of these correlations must wait until the symbol generation procedure has been completely specified.

In its usage, however, the assumption that any symbol can map to any other is still valid. Therefore, any analysis of the mapping process that assumes random associations would still be valid.

10.7 Investigation of One Strategy for Learning

The results presented in this section represent one possible interpretation of the implementation possibilities presented in this chapter. There are many others, using different combinations of parameters and/or making different choices for the options that have been outlined. Exploration of all of the alternatives is left for future work.

To recap, the network which will be considered in greater depth uses the additive synaptic combiner, places no constraint on the angle between weight vectors of consecutive synaptic levels and learns using an unordered (parallel) dependence of weight vectors. All levels of synaptic units are updated in parallel with each new pattern presentation.

The section begins by presenting the learning algorithm and then proceeds to the analysis, which consists of two parts: first, the impact that each learning event has on the networks ability to recall the learned pattern; second, the impact that the learning event has on the recall of other, unrelated, patterns. This provides a means of quantifying several properties of learning and establishing the trade-offs which can be made.

10.7.1 The Learning Algorithm

To begin each learning event, a dynamic pattern is set-up on the region of neurons using standard D-from-K-from-N coding. The pattern is held in short term memory by making direct connections of strength B between each of the neurons firing at time t and those firing at time $t+1$ for all L cycles of the complete period.

The learning takes place by updating the synaptic connections at all levels using the formula given below. Updating occurs during C complete periods of the pattern (i.e. a total of CL timesteps) during which time the pattern held in short

term memory decays (to simulate forgetting). After CL timesteps, the pattern is replaced with the next one to be learned.

At each timestep, the potentials of the neurons are updated as normal. The weight vector, W_{im} , of the level m synaptic units of neuron i is updating using the following rule:

$$\Delta W_{im} = O_i(t) \alpha_m \left(\sum_{d=0}^{t_m} \bar{O}(t-d) \right) - O_i(t) \beta_m W_{im}$$

Consolidation Normalisation

where $O_i(t)$ is the output of neuron i at time t , α_m and β_m are the learning and forgetting constants and $\bar{O}(t-d)$ is the network output vector d cycles in the past. The bracketed term thus represents the history vector of firing neurons which the synaptic units of level m can “see” when updating. The longer the firing time t_m of a synaptic unit, the longer in past can a pre-synaptic neuron have fired and still be visible to a neuron now firing.

Since both terms contain $O_i(t)$ then the weight vectors at all levels of a given neuron are only updated when the neuron itself fires.

The two terms represent the balance between *consolidation* (moving the weight vector towards the input vector) and *normalisation* (removing weight from synapses where the pre-synaptic neuron did not fire recently enough).

It is clear that this particular interpretation of the principles of learning hierarchies is close to basic competitive learning, discussed in chapter two. The essential differences are the multiple synaptic layers (each optimising over different timescales of activity), the D-from-K-from-N coding which allows multiple “winner” at each timestep (using the jargon of competitive learning) and the cyclic, recurrent nature of the patterns themselves.

We note that the changes to the weight values for a given synaptic unit do not have to all be of the same sign. Consider a synapse between two neurons i and j and what happens if neuron j fires first then neuron i fires at a time between t_b and t_d cycles in the future ($t_b < t_d$). The level b synaptic unit would reduce in value since there was no activity from its pre-synaptic neuron within t_b cycles, whereas the level d weight in the same synapse would increase in value. Thus the two weight values are tracking correlations of activity on different timescales.

The changes in the weights at all levels are such to increase the potential of each firing neuron in the face of the current pattern. To ensure that the weight vectors do not become saturated, we insist that the length of each vector be held constant. This establishes a relationship between the learning and forgetting constants, α_m and β_m .

Using the definition of weight vector length as the sum of the absolute values of the components, we demand zero change in the sum of the weight vector changes, that is:

$$\sum_j \Delta W_{ijm} = 0$$

by summing the elements of the weight vector at level m for neuron i . For the given learning algorithm this implies that for a neuron whose weight vector is changing (i.e. for which $O_i(t)=1$):

$$\begin{aligned} \sum_j \Delta W_{ijm} = 0 &= \alpha_m \sum_j \left(\sum_{d=0}^{t_m} \bar{O}(t-d) \right) - \sum_j \beta_m W_{ijm} \\ \alpha_m D t_m &= \beta_m l_m \\ \alpha_m &= \frac{\beta_m l_m}{D t_m} \end{aligned}$$

where in the second line we have made use of the facts that the summations over j and d merely produce the total number of '1's which appeared in firing vectors over the last t_m cycles, i.e. $D t_m$.

The summation over the elements of the weight vector, W_{ijm} , gives the length of the vector, l_m , as defined earlier.

Setting α_m according to this equation will ensure that any increases made to the synaptic weights between neurons are paid for by reductions in weights between the post-synaptic firing neuron and all other neurons (even those that are firing). The cost is divided such that each weight is reduced by the same proportion.

We note that there are several degrees of freedom still remaining to be constrained; the learning rate, α_m , the vector length, l_m , and the relative values of these parameters across the levels of the synaptic hierarchy. Also, the number of layers in the learning hierarchy is itself a degree of freedom which has not yet been addressed. The values of these parameters are crucial to the behaviour of the system, as will be made clear in the discussion presented next.

10.7.2 Effect of Learning on the Recall of a Learned Pattern

During the C cycles in which a pattern is being consolidated, small changes are made in the weight vectors of every firing neuron with a goal of allowing that pattern to be stable even after it has vanished from short-term memory (the a unit synapses).

In this simple version of the learning algorithm (borrowed from well established competitive learning techniques) the winning neurons move their weight vectors so as to be more aligned with the input vector. The amount by which the vector at each level in the synaptic hierarchy moves depends on the learning constant, α_m , for that level and also on the overall length of the weight vector, l_m , at that level. If both of these terms are large then the change in potential that can be

achieved in a single update is relatively large. This should be the case for the low level synaptic units.

Conversely, a small value for both α_m and l_m will result in very little change in potential per update. This is illustrated below in a diagram which is simplified by ignoring the fact that each level of the synaptic hierarchy is responding to a different output history vector which depends upon the duration for which that unit is active once stimulated. The basic idea remains the same, however.

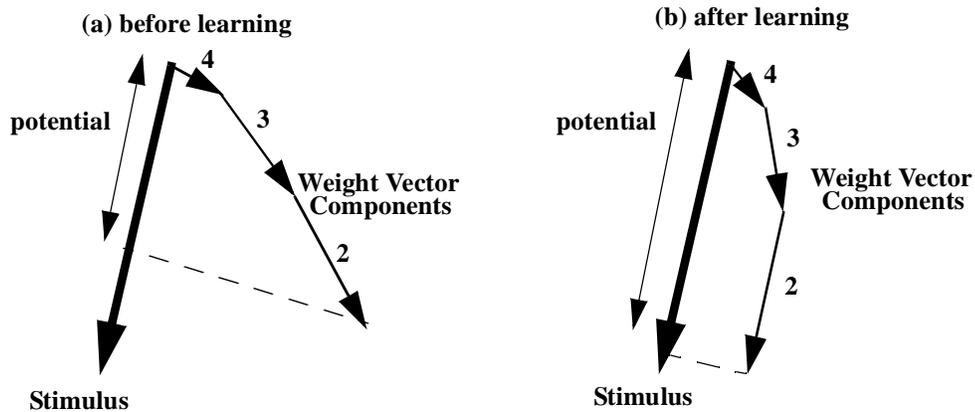


Fig. 10-11 Movement of the hierarchical weight vector during learning.

In the figure, the level two synaptic units produce the greatest change in potential after a learning event. The level three and four units have moved in the right direction but their changes are progressively more modest with increasing level.

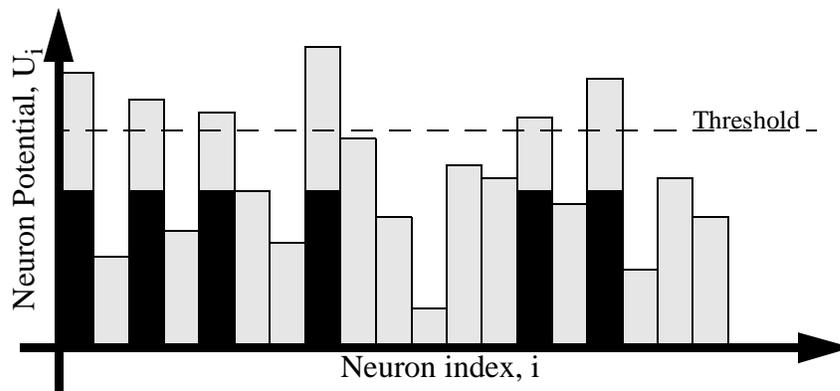
The aim of learning is that the changes should be sufficient to give those neurons which are intended to fire more potential than the others even after the level one synaptic units have decayed to zero (and thus forgotten the pattern being learned). How the potential is distributed between the various layers is irrelevant in the short-term since the network is only concerned with faithfully representing the current pattern of potentials as the pattern is played out.

If the pattern was to be learned with no modifications to the subsets of firing neurons, this could in principle be achieved in a single cycle if the learning parameters, α_m , were sufficient to move the weight vector a large distance towards the stimulus. However, as discussed earlier, the decay of level a synapses during consolidation will permit the network to locate neurons which currently do not fire in the original encoding of the pattern but which could carry the pattern with smaller changes to the weights than would be needed for the original firing subsets. To do this in such a way as to allow both the original pattern and the existing weight distribution to guide the updating process, the changes to the weight vectors are made slowly over many cycles of the pattern.

The potential of each firing neuron will include a term of size DB where D is the number of neurons in each firing subset and B is the strength of each level a connection between firing neurons in the pattern. It is this strength B which is subject to decay during consolidation. The value of BL , though decaying, is identical for all neurons in the firing subset. Thus any variation in the potential of such a neuron is due either to noise in the firing subsets (in other words erroneously firing neurons) or to the potential contributions from other components of its weight vector which themselves are the product of previous learning.

The figure below shows the potential of the neurons in the region at time t and again later at time t' . The learning in the level a synaptic units (corresponding to the pattern to be consolidated) is shaded dark. The light shading corresponds to the potential contributed by the other components of the weight vector.

(a) *Potential Distribution at time t near the start of consolidation.*



(b) *Potential Distribution at time t' at the end of consolidation.*

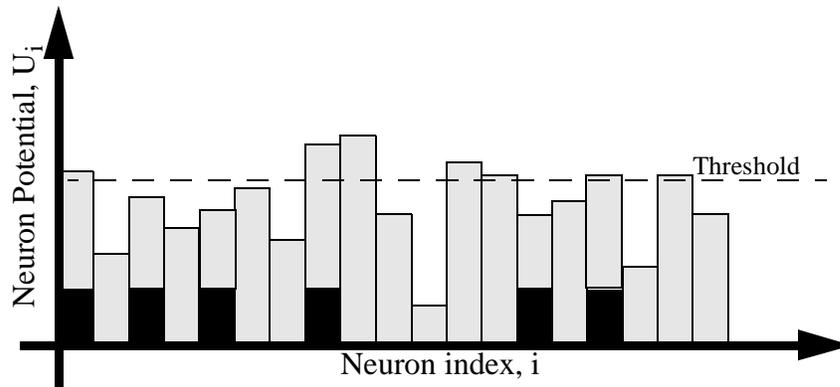


Fig. 10-12 Histograms of neuron potentials (a) before and (b) after consolidation.

The schedule for the decay of level a connections is a parameter of the network. Linear decay has been assumed, though exponential decay is another possibility. In either case, on every successive circuit of the pattern, the fading power of the level a synapses should allow neurons which do not belong to the pattern to become more frequent in the firing subsets. These are all neurons which have high potential even without the level a synapses and, therefore, require relatively small

changes in their weight vector to join the pattern. Small changes equate (in general terms) to little perturbation in the established learning.

The disadvantage of recruiting other neurons to replace the firing subsets, as discussed earlier, is that the information stored in the pattern is eroded as a result. The firing subsets themselves not only support the current pattern but are involved in the associations that it makes with other patterns, so any degradation of the pattern is a source of noise to every association it is involved in.

There are two important points to note here. First, there is clearly a balance between the optimality of storage (which would tend to re-use any neuron which required few changes to carry the pattern) and fidelity of recall (which demands that the same firing subsets be maintained and fire in the same order after consolidation as before). Recall that in all previous analysis of the K-from-N code (and its dynamic variant, the D-from-K-from-N code) there was always a margin for error in terms of the number of erroneous bits which were permitted at any time. We now use this margin as a source of flexibility, permitting errors to enter the encoding up to the limit allowed by the neighbourhood size around each pattern.

The second point to note is that a neuron that has large weight values in the higher levels of its synaptic units biases the neuron selection towards choosing it to carry the pattern. Thus, neurons which win the competition to fire will tend to do so again: positive feedback leading to greater differentiation of the weight vectors.

In the next step in the analysis we consider, in very rough terms, the number of cycles of the complete pattern, C , needed to consolidate the pattern being learned and the relationship with the learning constant and the reliability of the memorisation.

In each cycle, the weight vectors of the firing neurons move towards the stimulus vector by an amount which depends on the learning constant, α_m , for each weight vector component. After only a few such learning events, we must ensure that the levels of the hierarchy from b to g are sufficient to support the pattern reliably. But the weight vectors at level b are themselves subject to change while learning other patterns. For this pattern to survive for a longer duration it must be assimilated into even higher levels.

After a single presentation, the pattern should not have elicited sufficient change in the levels above level b to be stable if the level b weights were to be erased. To quantify these concepts, we define a new set of constants, f_m , as the number of presentations needed for a single pattern to become stable in layer m . For layer a this value is unity while for layer b it must still be very low. Higher layers would permit larger values for f_m and would depend on the system requirements.

We assume that, at the start of learning, the vector W_m is in a direction which is random with respect to the desired (target) direction, W_m^t . For the vector component to fully capture and represent the pattern after f_m presentations each of C

cycles, the total number of modifications is Cf_m . Correlations between patterns may aid in the learning process, reducing the total change necessary for consolidation to level m , but for this simple analysis we assume that no such correlations exist.

To ensure that the weight vector at level m for neuron i can reach the target vector in Cf_m steps, requires the following inequality to be true in the worst case:

$$\alpha_m > \max_j \left(\frac{|W_{ijm} - W_{ijm}^t|}{Cf_m} \right)$$

where the max function ranges over every neuron. Since we assume that the weight vector before and after are independent, the worst case for the difference between them is that a single weight value which is initially at zero must reach the maximum value possible, which occurs when the vector consists of Dt_m weights of magnitude l_m/Dt_m and the other bits zero. Thus:

$$\alpha_m > \frac{l_m}{DCf_m t_m}$$

This constitutes a lower bound on the value of the learning constant for a synaptic layer that must be able to *guarantee* convergence in f_m presentations of a pattern and allows for the worst case situation: a complete transfer of weight to a direction in which there was previously none.

10.7.3 Effect of Learning on The Recall of Other Patterns

While modifying the synaptic weights to store pattern p_i , the changes made during the storage of all other patterns are being eroded. The quantification of this erosion will be considered in this sub-section.

When a pattern is learned, the weight vectors which were put in place for the previous pattern are subject to some disturbance. By careful choice of the parameters of the network, the lower level synaptic units will have made the largest changes during the consolidation of the last pattern while the high level units will have changed only slightly. These are necessary features of the weight vector dynamics to differentiate one extreme of the synaptic levels as short-term and the other as long-term storage.

One aspect of this hierarchy which has perhaps been under-emphasised throughout the development is that the changes in the lower levels of the synaptic hierarchy, though large, should have a lesser impact on the recall of other patterns than the concomitant changes in the higher levels, due to the longer duration of the high level units once activated. This is the opposite of the effect of those same changes on the storage and recall of the pattern being consolidated.

In order to quantify the effects of learning on the existing patterns, consider a neuron, i , which has been taught to fire during cycle k in a pattern, p_0 . Upon learning a new pattern, p_1 , the weight vectors which made neuron i fire will have been

altered only if neuron i fired during the consolidation of p_1 . If not, the neuron is unaffected under the learning algorithm. If its weight vectors were changed, we need to assess how the modifications made at each level of the synaptic hierarchy contributed to that change and to ensure (by selection of parameters) that those contributions become less at each higher level. If this is *not* so then the longer term storage levels (g, f , etc.) are less stable than the lower levels (a, b , etc.) which is counter to the principles of hierarchical learning.

The distortion, $Q(W_m)$, of each component of the weight vector as a result of learning a single pattern is defined as the ratio of the change in the vector to its length. In the worst case this is:

$$\begin{aligned} Q(W_m) &= \frac{|W_m - W'_m|}{|W'_m|} \\ &= \frac{1}{l_m} \sum_i |W_m - W'_m| \\ Q(W_m) &= \frac{C t_m}{l_m} (D \alpha_m + (N - D) \beta_m) \end{aligned}$$

where W_m is the weight vector after the new learning, W'_m is the same vector before learning and C is the number of cycles of the pattern in which learning takes place.

In the final line, the difference between the two vectors is replaced with the worst case value, which occurs when the firing subsets visible to the synapse at level m are constant through the learning process. In that case, $D t_m$ of the input weights increase by an amount α_m , while the other $(N - D) t_m$ elements are reduced by an amount β_m on each cycle of the pattern. Substituting for β_m in terms of α_m gives:

$$\begin{aligned} Q(W_m) &= \frac{C t_m}{l_m} \left(D \alpha_m + (N - D) \frac{\alpha_m D t_m}{l_m} \right) \\ &= \frac{C t_m D \alpha_m}{l_m} \left(1 + \frac{t_m (N - D)}{l_m} \right) \\ &\approx \frac{C t_m^2 D \alpha_m}{l_m^2} (N - D) \end{aligned}$$

We demand that the distortion due to learning a single pattern is lower for increasing level, m . Since N, C and D are constants, then lower distortion for increasing m corresponds to reducing the ratio $\frac{t_m^2 \alpha_m}{l_m^2}$ as the index m is increased.

Complications arise, however, since value of t_m increases with the level index m : this was a consequence of choosing option 2 in the initial definition of the synaptic hierarchy where it was deemed appropriate that the longer term storage elements should provide potential to the neuron for a relatively long period after

pre-synaptic stimulation. The rationale behind this decision was that such units would extract common features in the input patterns and allow other features to be coded in terms of them. Thus, t_m is large for large m .

This implies that the ratio $\frac{\alpha_m}{l_m^2}$ must tend to zero even more strongly with

increasing m to compensate for the increase in the term in t_m^2 . If we define r_m as a positive constant which decreases for increasing m , then the constraint on l_m for reducing the distortion for increasing synaptic level, m , is:

$$l_m > t_m \sqrt{\frac{\alpha_m}{r_m}}$$

If the value of l_m does not fulfil this condition, higher levels of the hierarchy (which are supposed to be more stable in the face of each learning event) may be less stable than the lower levels and the memory system is liable to wander without convergence at each learning event.

10.7.4 Summary of Learning Constant Constraints

Two basic constraints have been developed to restrict the selection of learning constants in the network. To ensure that the network can learn a pattern in a given number of presentations, the first constraint is:

$$\alpha_m > \frac{l_m}{DCf_m t_m}$$

while the requirement of reducing the distortion of a single learning event at each level of the hierarchy leads to the following:

$$l_m > t_m \sqrt{\frac{\alpha_m}{r_m}}$$

Combining both expressions creates a range of values for the learning constant:

$$r_m \left(\frac{l_m}{t_m} \right)^2 > \alpha_m > \frac{l_m}{DCf_m t_m}$$

10.8 Simulations of Simple Learning Scheme

To verify the algorithm developed in this chapter, a network of $N = 1000$ neurons was used to store dynamic patterns consisting of $L = 8$ subsets of $D = 16$ neurons. A four level synaptic hierarchy was used, with the a units providing short-term storage during consolidation, as described in the learning algorithm. Thus, three levels remained for medium and long term storage.

The simulation consisted of tracking the learning of a block of twenty patterns looking at the changes in each of the synaptic levels. During one pass of the pattern set, each pattern was set up in short term memory using the a unit synapses, remaining there for twenty complete periods.

During each period the weights were updated using the learning algorithm already described. After every pass of the pattern set, the ability of the network to recall each learned pattern, but this time with the a unit synapses reset. Thus the short term memory was empty and the network had to rely only on its medium and long term store.

The results obtained for the algorithm were disappointing. While the network adapted at each trial to the pattern currently being learned, any changes made were quickly erased by the next pattern.

The failure could be attributed to one of six causes. First, the network itself could be too small. Increasing the network size might lead to more freedom to create correlations between firing subsets and lead to more optimal storage.

Second, the parameters of the firing patterns, D and L , may be set incorrectly. Either increasing or decreasing these parameters would have both advantages and disadvantages, however. Reducing them would reduce the number of neurons changing for each pattern with a possible reduction in the distortion on each learning event. It would also reduce not only the quantity of information stored in each pattern but also its signal to noise ratio. This in turn would reduce the maximum size of each neighbourhood, demanding more faithful storage of each pattern to preserve enough information to remain distinct. This might, in itself, demand greater distortion of existing learning, leading to worse overall performance.

Conversely, increasing D and L would increase the number of neurons changing in each update but, for the same reasons given, might reduce the distortion by permitting greater overall distortion of the pattern while still leaving it sufficiently distinct to be useful.

The third possible cause of the failure could be that the parameter values are not set correctly. The first place to look here would be the choice made earlier for the trends in the two complimentary parameters W and t (section 10.2, page 258). At that time, the desire to have the level g synapses weak (i.e. short W) but active for a long time, t , (in order to permit a form of delta encoding by providing a stable undercurrent of potential for the lower levels) made the choice of increasing t with level the appropriate one. The analysis revealed that this choice acted against the need to reduce distortion with increasing level. The distortion term was propor-

tional to $\frac{t_m^2 \alpha_m}{I_m^2}$. Repeating the analysis for option 1 might prove more fruitful.

Furthermore, more detailed analysis may reveal that there are further constraints on the learning parameters for the chosen algorithm which have not yet been met and which would allow the simple variant of the learning hierarchies approach to achieve its potential.

Fourth on the list of potential problems is the time course of the decay of level a synapses. A linear reduction was assumed but this may not be optimal. Faster initial decay would bias the learning in favour of re-enforcing the existing synapses rather than imposing the new pattern. Conversely, a slow initial decay followed by faster decay towards the end would favour the new pattern: most changes would occur while the new pattern was still relatively intact.

As a fifth possibility, it may be that, even though the learning hierarchies approach is valid, the simple algorithm chosen (based on standard competitive learning) may be inadequate for the task given the many extra complications introduced using learning hierarchies. This is a likely reason given the simplicity of the algorithm. In its original habitat, the weight vectors of each neuron are permitted to evolve slowly whereas, in the current context of learning hierarchies, they are forced to converge swiftly. Also, the optimisation of feature detectors has not been addressed rigorously and it may be that a greater degree of care is needed to locate features that are stable in the face of new learning using learning hierarchies.

Finally, it is possible that the whole concept of learning hierarchies is fundamentally flawed and no amount of effort to tune the algorithm or its parameters will be enough to make it work. This remains to be proven, but the basic principles of learning hierarchies (using parallel synapses each of which possessing different parameters and properties) not only seem to address the stability-plasticity dilemma in a relatively simple way but also open the way for a whole new range of options for balancing the demands of responsiveness and accuracy which should not be written off based on a single result using an *ad hoc* choice of parameters.

At this time, it is assumed that it is the algorithm and the choice of parameters which needs further work, rather than a fundamental error in the learning hierarchies approach. However, there is no evidence which supports learning hierarchies as a viable concept and so much further work is clearly needed to hone the principles presented in this chapter into a working solution.

10.9 More Complex Learning Schemes

The failure of the learning hierarchies algorithm to display the expected properties has been assumed to be mainly due to the overly simple choice for the learning algorithm and parameter values. In this section, a more advanced scheme will be presented which has not yet been fully investigated, but which may lead to better network performance.

10.9.1 Learning Only To Maintain Potential Within Bounds

Here, the target of learning is to preserve the potential of each neuron within precise boundaries, depending on whether it are firing or not. The weight vectors are no longer fixed in length but grow and shrink depending on the potential of the neuron when it is both firing and silent.

Figure 8-16, (overleaf) illustrates this, showing a number of potential bands which act as targets for firing and non-firing neurons respectively.

When firing, a neuron should have a potential inside the band marked 'firing only'. All non-firing neurons should have potentials below the area marked as guard band. Any non-firing neuron whose potential enters the guard band region may achieve firing potential erroneously, either through further learning or through noise and so it is prudent for the network to reduce the potential by moving the weight vector of such neurons directly *away* from the stimulus instead of towards it. This idea is, in fact, one facet of normal Hebbian learning.

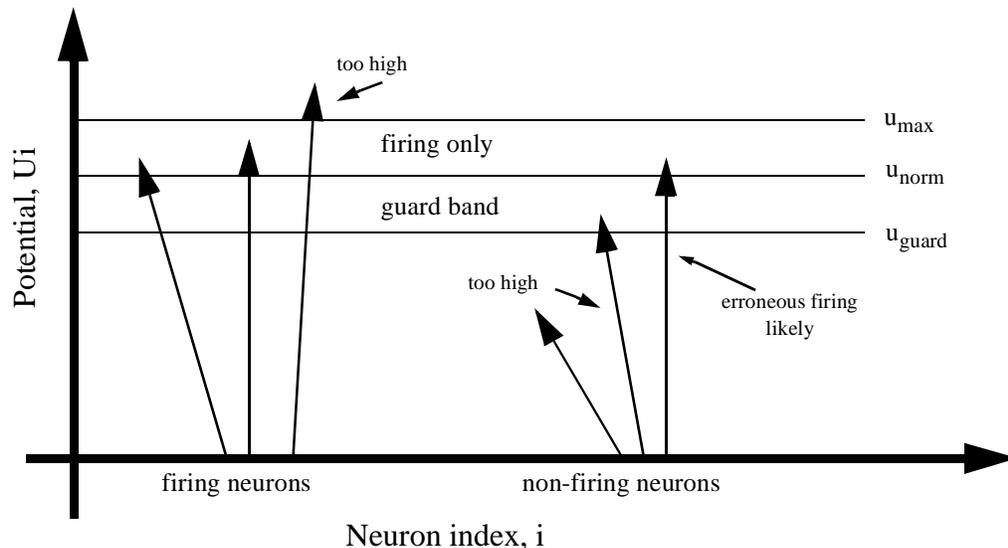


Fig. 10-13 Potential bands in a more advanced learning scheme.

At any time step, a firing neuron should have a potential between u_{norm} and u_{max} . If it is too low it risks being overtaken by a neuron from the non-firing subset. If it is too high (above u_{max}) its weight vectors are too long, which is a source of excessive noise when the neuron in question is part of the non-firing subset at other times. It should reduce its potential, again by moving its weight vector *away* from the stimulus vector.

The effect that this learning algorithm has on the generalisation performance of the network is not clear since the statistics of the input stimuli are being distorted by the network: if the potentials of every neuron fall into the correct band when a new stimulus is presented to the network then no changes are made to the weight vectors. This is satisfying from the point of view of making corrections to the weights only if they are necessary but is a source of distortion to the optimal representation of a given probability distribution.

10.10 More Complex Synaptic Structure

This section briefly considers another level of complexity which might be considered for the synapse. Two ideas are discussed which focus on the time response of the synapse, one for signalling and one for learning.

10.10.1 One Interpretation of the Lynch & Baudry Theory

This scheme is inspired by the Calpain-Fodrin hypothesis of long-term potentiation (LTP) (Lynch & Baudry, 1984) in which it is suggested that LTP is effected by revealing ion channel in the post-synaptic membrane which were ever present but hidden by a web of protein (fodrin). Learning is achieved by action of a catalytic protein (calpain) released during excessive pre-synaptic stimulation which breaks down the fodrin and makes the ion channels available in the membrane.

The theory is interpreted in the context of learning hierarchies as follows: certain levels of the synaptic hierarchy are initially dormant (i.e. contribute nothing to the potential of the neuron) until stimulated by pre-synaptic activity, at which time they become active but still contribute no potential. If subsequent pre-synaptic activity occurs within a short time then the synaptic unit responds at full strength, adding to the neuron potential. However, once a time limit has elapsed without re-stimulation the unit again becomes dormantⁱ. New stimulation will re-awaken it but will not produce any contribution to the potential until a further action potential arrives within the time limit.

What overall effect does this produce? Essentially it allows the neuron to respond *differently* to two pulses from the *same* neuron depending on whether they occur within a certain space of time or not. This may be useful in a network, such as this one, where the patterns are cyclic with a period which is within the 'primed' period of the synapse since it allows a cycle (i.e. a known pattern which has been recalled) to trigger a new behaviour without external stimulus whereas a non-cyclic pattern could permit the network state to wander for a long time without re-stimulating the same connection and initiating the new behaviour.

This is not the same as Amit's investigation into temporal pattern retrieval using a variant of the Hopfield network (Amit, 1989) since he used synapses which consisted of a fast and a slow component. Once activated the synapse is guaranteed to deliver activity in proportion to each weight but with the slow component delayed in time. The effect is not the same, since Amit's scheme did not allow the same pair of neurons to be coupled in two different ways dependent on the stimulation which *is* the case with the idea presented here.

In fact, it turns out that this kind of synaptic behaviour has been proposed before. In his exposition of the searchlight hypothesis for the thalamus, Crick draws upon earlier work by von der Malsburg on a synapse that can take on two

i. The Lynch & Baudry theory does not postulate a return to the dormant state if the synapse remains unstimulated. However, it does not rule it out, either.

values depending on whether or not it has been activated (Crick, 1984; von der Malsburg, 1981). In Crick's hypothesis however, this activation is achieved by virtue of a set of dedicated connections directly to individual synapses and hence it differs from the proposal presented here. Whether or not the scheme as described by Crick is a true explanation of the thalamic processes he seeks to describe, such an approach is impractical for use in every synapse in the network. By avoiding an external agent to activate the synapses, this proposal offers a more practical method of achieving more complex (and hence more powerful) synaptic behaviour.

We could view the two-valued synaptic unit as another level of complexity in the learning hierarchies approach. It is clear that the potential increase in performance that this added layer of complexity brings also entails an increase in complexity of the associated learning algorithm. We must now optimise both the active and the inactive synaptic value, and do so for each level in the synaptic hierarchy. In addition, we now have two extra network parameters. The first is the degree of activity required to active the higher synaptic value and the second is the time delay between activation and deactivation.

Furthermore, if we also model the synaptic unit as being made up of many parallel ion channels, as in a real synapse, the activation and deactivation processes could be modelled as many parallel random processes governed by a time constant. This would transform the abrupt transitions between active and inactive states into a probabilistic mix of the two types. Such a level of detail may add something to the network, but exactly what that might be is left for future work.

10.10.2 Different Learning Response for Each Synaptic Level

Here, we consider the possibility of varying the time course of learning by synaptic level, permitting synaptic units at the low levels not only to respond more strongly than those at high levels but also more quickly. This is important when we take into account that during consolidation the new pattern is initially represented accurately by the level a synaptic units but as time passes and the value that they hold decays, the pattern will be modified even as it is consolidated as a result of the existing connections between neurons.

By altering the time course of learning for each synaptic level it would be possible, for example, to modify the high level units only late in the consolidation phase when the value of these units is more dominant in the pattern. This would increase the tendency for the weight value of the high-level synaptic units to self-amplify rather than being dominated by the randomly distributed patterns.

10.10.3 Conclusions

This section has presented a number of avenues for increasing the capabilities of the network by increasing the complexity of synaptic behaviour. The use of more complex synapses may facilitate a whole new level of complexity in the model but brings with it the burden of more complex analysis.

10.11 Memory Capacity of a Network Using Learning Hierarchies

The analysis of the storage capacity has not been attempted as yet. Compared with the simpler schemes of the static and dynamic networks already presented there are two sources of added complexity in the learning hierarchies model.

The feature extraction performed by the network is intended to increase the storage capacity by extracting essential characteristics from the patterns. Just as would be observed in a multi-layered perceptron (MLP), however, the network is able to generalise (to some degree) by generating output in response to previously unseen input using its extracted features. Thus, the network can no longer be analysed purely on its ability to retain and recall known patterns but rather to make correct assignments on the basis of category, just as an MLP does.

The second source of complexity is the fact that the patterns themselves are deliberately modified as a result of learning. Thus the distinction between an error and a modified pattern must first be made. In the context of the evolving network in which the relations between patterns must evolve with the patterns themselves, the identification of this boundary is not easy to define. But this is a first step to assessing the true quantity of information stored in each pattern vector.

10.12 Discussion

The implementation of learning hierarchies that has been presented in this chapter represents only one possible choice of the options and parameters that are possible. The values of the parameters were chosen in an *ad hoc* fashion, though in depth analysis will no doubt allow more optimal selection even for the simple algorithm. It is possible that alternative algorithmic choices will lead to a learning performance which shows the advantages of learning hierarchies, although it is unlikely that there will be a single set of optimal parameters, merely optimal solutions to a set of constraints imposed by a given problem.

It is also important to note from the architectural development of chapter five that the network equivalent of inheritance (intended as the mechanism by which the network will implement generalisation) is not *necessarily* realised by this learning hierarchies approach. However, it is envisaged that correct tuning of learning hierarchies will facilitate inheritance, subject to additional constraints which remain to be elaborated.

What has been shown in this chapter is a range of new network options, which may be applicable in addressing the stability-plasticity dilemma. The envisaged goal of the approach is that the volatile element of each weight vector can aggressively adapt to represent a novel pattern or mapping, while producing little impact on the network's ability to implement its existing mappings. The less volatile, slowly adapting elements of the same weight vector have a different set of goals, placing more emphasis on global features and optimal representation of data

rather than learning after a single presentation. The range of elements in between these extremes seeks to smooth the transition from short to long term memory. A by-product of assimilation in this scheme is that the pattern must adapt to the network as well as the network to the pattern; a concept which has not received much attention in the literature to date.

Even with the learning hierarchies approach (using an admittedly simple learning algorithm as an example) it has not been shown that the capacity of the network has reached the sort of levels necessary for the type of system envisaged in the architectural development. However, the foundations upon which learning hierarchies are based is viewed here as a starting point for explorations into new and more complex learning algorithms which are capable of the required level of mass storage. More detailed analysis is needed to thoroughly investigate the avenues that have been opened up in this preliminary investigation. It would seem appropriate to bring stochastic analysis to bear on the problem, since the network acts as a state machine subject to noise, which makes the transition matrix itself stochastic in nature.

10.13 Conclusions

This chapter has quantified a number of key ideas that were developed in the context of symbol encoding for the architecture presented in chapter five. In it, the notion of a complex synapse made up of a number of parallel units each with its own set of parameters was developed and analysed. Learning, on both the short and long term timescales was presented as the re-arrangement of synaptic weights between the parallel synapse units of multiple neurons. In the course of this presentation, a range of possible architectural options was considered and contrasted.

The described benefits of synaptic rearrangements were manifold: to optimise the data structure by maximising the reuse of existing connections; to minimise disturbance to existing memories during consolidation of new patterns; to provide a potential solution to the stability-plasticity dilemma and to enable classes of memories to be created, using a priming pattern to access them. The drawback with the scheme was that the patterns themselves were modified as a result of consolidation reducing the information contained in each pattern but permitting more optimal storage and faster consolidation of each pattern.

It was noted that a more detailed study of the concepts of learning hierarchies would be necessary, probably based around the theory of stochastic processes. Several new avenues of exploration based on more complex learning or synaptic structure were proposed which may lead to improved performance and storage capacity but at the same time would require more involved analysis.

The next chapter brings together the threads that have been woven throughout the thesis. It explores some of the questions that remain unanswered and some of the high-level architectural issues that must be addressed to create a general neuro-symbolic processor.

Towards True Neuro-Symbolic Computation

11.0 Introduction

As the final stage of the work, this chapter presents an attempt to draw together the different threads that have been presented and to try to bridge some of the gap between the work as it stands today and the realisation of a general purpose neuro-symbolic computer. By attempting to unify what may appear to be unrelated material the principle disadvantage is that the ideas presented in this chapter are more like pointers to future work than solutions in themselves. Hopefully they form a valid framework in which to accelerate future progress. Like the architecture and learning hierarchies chapters that have preceded it, this chapter describes a number of independent ideas. The serial nature of writing forces them to be presented in linear fashion.

The first section sets the scene by presenting a few key impediments that still stand between the connectionist and his goal of constructing a general purpose intelligent system, which we now tacitly assume will be a neuro-symbolic system. For some of the topics that are raised, the discussion will go no further in this thesis. The exposition of meta-knowledge is one such topic. For others, this section is only a starting point and the remainder of this chapter will try to widen the scope of the discussion. In many cases the goal is to bring together some of the pieces developed earlier that did not seem to fit anywhere, including drawing from the work of others as reviewed in the second, third and fourth chapters. Hopefully by the end of this chapter a more coherent whole will begin to emerge.

To help develop the processing model, the next section reviews the roles of hierarchy and modularity in a general purpose component for an intelligent system by considering the compromises that must be made to an ideal module to take reality into account.

The third section looks at alternative rule-based algorithms to the forward-chaining used in most rule-based architectures. It is particular concerned with highlighting the fact that hooks are already in place in the neuro-architecture proposed in this thesis to accommodate a range of such alternatives in a relatively seamless fashion.

The failure of current neuro-symbolic systems to provide true parallel processing is the subject of the next section. Alternative strategies for parallel processing are considered, leading to further constraints on the symbolic encoding. The importance of managing finite resources is considered and used as the impetus for an alternative theory of intuitive reasoning based on unguided sub-symbolic processing.

The final section presents conclusions and discussion on the main areas for future research arising from work in this chapter.

11.1 Barriers to the Connectionist Dream

Since the publication of the PDP volumes (Rumelhart & McClelland, 1986), a certain air of expectation has existed throughout the connectionist community that at some point it would be possible to construct a connectionist system that would outperform all existing rules-based, symbolic systems in all categories in which they currently excel: problem solving, flexible application of knowledge, systematicity, etc. But more than a decade after the gauntlet was thrown down, so to speak, that day has not yet arrived and despite the considerable progress that has undoubtedly been made there are a number of key issues that are perhaps not as central to the debate as they should be.

This section reviews some of the expectations that have been put forward for neural networks as a framework for reasoning systems and identifies key areas which need further exploration. They are, in the opinion of this author, barriers to the realisation of efficient neuro-symbolic systems. The aim here is to reiterate the importance of these areas of research and to explore the issues in greater depth.

11.1.1 Common Mechanism for Rule & Intuitive Processing

In his 1988 paper, Smolensky defined two types of processing that he expected an intelligence system to be capable of. One of these was a rules processor and the other an intuitive processor. He likened the rules processor to the many symbol processing architectures proposed in the AI community. But to Smolensky the types of operations performed by the intuitive processor could not be easily captured by rules and it was here that he envisaged a connectionist approach to be more appropriate.

But beyond his account of the two types of processing was the even more radical notion that both types of processing could be performed in the connectionist framework, since (among other benefits) such an approach could provide a better explanation of how rule following and rule breaking behaviour could be handled seamlessly together.

This view is also adopted here. It is not enough for us to create a system which merely implements rules following. It is also not enough to create a system which, like a Boltzmann machine, can simulate “feats of intuition” using a settling procedure. Instead we are looking for a single network which has a continuum of

modes of operation, the extremes of which one could describe as strictly rules following and strictly intuitive. Since this is a major topic, it will not be pursued further in this section. However, the requirements for co-existing rules-based and intuitive processing will be developed in later sections (see section 11.3, page 310 and section 11.4, page 319).

11.1.2 Emergent Computation, Even of Symbol Structures

One of Fodor & Pylyshyn's major criticisms of the connectionist approach in their 1988 paper was that even if neural networks were able to demonstrate key properties such as systematicity and productivity (which they saw as crucial to developing truly intelligent machines) then such networks would be merely implementing classical, symbolic algorithms anyway. But central to Smolensky's philosophy was that, on the contrary, it would be possible to construct a neural network that, in addition to demonstrating the necessary properties would have additional (but also vital) properties, undreamt of in the symbolic philosophy, whose identification would vindicate the whole connectionist approach.

In the early years following the Smolensky/Fodor debate several of the researchers working in the field addressed this point. As presented in chapter three, work by Pollack, Chalmers and Chrisman attempted develop the idea of distributed representations from merely an efficient means of compressed storage (requiring symbol structures to be decompressed before use) into an encoding scheme which was intrinsically useful (that is, facilitating direct manipulation of compressed data during processing without the explicit recovery of constituent structure). Why would we want to do this?

There are a number of reasons. First, we assume that operating on a packed vector will be more efficient than unpacking the vector, operating on the individual pieces and then, potentially, re-pack the result into a certain form. If it is possible to achieve all of these individual operations in a single step, this would appear to be desirable. But it pre-supposes that the single step operation requires fewer computational resources than those required to unpack, transform and then re-pack the symbol structure. We must be sure that the format of the compressed vector is such that we can meet this constraint.

Another reason for operating on compressed vectors is that it might be possible to reduce the amount of resources required for storage in long-term memory. However, this is not guaranteed. For example, it is unlikely that the compressed formats proposed by Pollack *et al.* would require fewer resources for storage than the original structures. This might seem to be at odds with intuition since in each case a set of vectors each of size K is compressed to a single vector of the same size. It would be easy to assume that this single vector would require less resource to store it than the complete structure. But the information content of the compressed vector is approximately the same as that of the original structure. The binary bits of the individual vectors have been transformed into multi-valued outputs in the compressed format and so all of the original information has been preserved.

Notice that this stands in stark contrast with the symbol encoding proposed in this thesis. A key point in the architectural development of chapter five was to argue that symbol should explicitly represent only that information which was needed for the type of operation in which it was used, together with sufficient information to retrieve the full content that the symbol replaces. This alternative compressed format need not have the same content as the original structure and so it should require fewer resources to store it than the original. The issue then remains as to whether the content that is not readily accessible can be retrieved in a sufficiently short period of time to be practical. In a well designed and efficient system the more often we need to access the information the less time and energy we should need to spend retrieving it.

This leads us on to the mechanism of processing that we would use with such compressed vectors. Consider the syntactic transform proposed by Chalmers. His network permitted one compressed vector to be associated with another, with the intent of extracting the underlying structures of the two formats as a means to implement generalisation (where generalisation in this case is akin to productivity in the nomenclature of Fodor and Pylyshyn). In this regard he was partially successful although the mechanism was not very robust.

The goal that both he and Chrisman were seeking to attain is a vitally important one. Without achieving it, it is doubtful that the superiority of connectionism as an alternative to the rule-based paradigm will be demonstrated. Ultimately, what we want to achieve is a processing mechanism that uses spreading activation between neurons that are capable of setting on new (i.e. never before seen by the system) patterns that can be unpacked to reveal hidden structure. To do this, the sub-symbolic operations that are causally related to the network's connectivity must somehow implement syntactically correct transforms at the symbol level.

One of the major problems that Chalmers and Chrisman had with their internal representations was in the use of the back-propagation algorithm, which, despite its ability to handle a wide range of association problems, is actually unsuitable for such work for a number of reasons.

The most important of these, as has been stated, is that the structure of the internal representation is not well controlled. The global structure of syntactic rules is inseparable from the rawest association between example vectors. For the purpose of this discussion we will refer to the structure that does develop through back-propagation as *weak structure*. Traditionally, it is almost with pride that connectionist researchers announce that their internal representations are indecipherable by humans. Any hand-crafting in the structure is seen as somehow unclean or sub-optimal.

But it is perhaps erroneous to suppose that having weak, unconstrained, structure in an internal representation will lead to better performance. There should be a way that allows the representation to develop more complex structure without making it necessary to constrain how it achieves it at a high level. One way to do this would be to build structured learning directly into the learning algorithm at the neuron level. Structured learning, leading to what we will refer to as *strong struc-*

ture, could be interpreted as putting richer information at the disposal of the learning algorithm than the firing or silence of bits in the current vector. Although more structured techniques such as adding a momentum term to the learning algorithm have proved successful in MLP simulations, it is often the case that consecutive learning trials are unrelated and it is not clear whether the momentum term is helping in a systematic manner.

This thesis has outlined a number of new approaches to learning structures using a hierarchy of independent synaptic layers. The intention of these approaches, based on the far more predictable performance of associative memories, is to address the limitations of back-propagation in this regard by providing a robust and predictable alternative. The underlying assumption of this approach is that the structure that exists in the multi-levelled weight vectors will allow the internal representations to develop on several level simultaneously. Structure that exists in the input vectors at any particular level will be self-reinforcing and hence will be extracted.

More work is required to justify these expectations and to fully explore the many options that learning hierarchies could offer in this regard. Certainly, the results of chapter ten have not yet shown that trust in this approach is justified.

As a final point about the problems of back-propagation (although less relevant to this discussion) we mention briefly the well known problems of back-propagation requiring a large (but unpredictable) number of learning trials to converge on an acceptable representation, as well as the fact that convergence is by no means guaranteed.

Returning to the main topic, there is other reason for operating on compressed vectors rather than whole structures. It is that such an approach might permit us to process more than one piece of information at a time, leading to an improvement in processing efficiency. This possibility is discussed in more detail later (section 11.4, page 319).

The figure below shows how a given input structure is first translated to a compressed format and then undergoes a series of transformations, T_1 , T_2 , etc. At each step of processing, it would be possible to re-extract a symbol structure from the distributed internal representation.

In Chalmers' network, a single transform T_1 was effected between two previously learned formats and even this on its own was an impressive result. But we require a level of systematicity that goes much further. Not only must we be able to perform transform T_1 and then unpack the results into a separate symbol structure but we must also be able to perform a series of transforms, $T_2 \rightarrow T_n$. For each transform, there may be restrictions on what counts as acceptable input but the system cannot verify in advance that any given series of valid transforms will preserve the syntax of the original structure. We see that performing multiple transforms in series on the compressed vector places demands on the fidelity of each transform: accumulated error must be kept within known tolerances or else we risk losing the structure of the symbols.

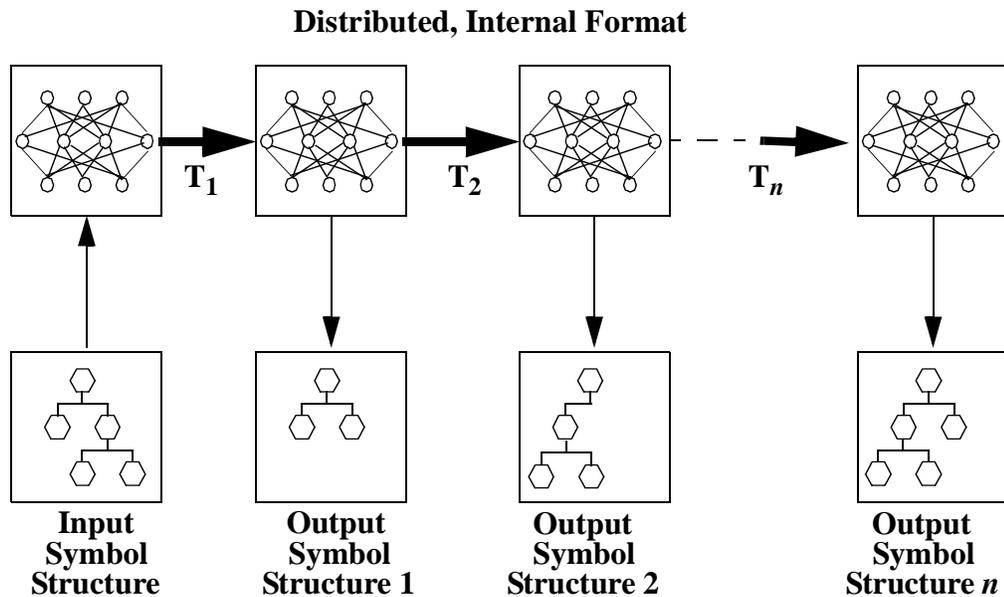


Fig. 11-0 Desired mode of computation in a neuro-symbolic processor.

So the syntax of the compressed vectors must be systematic and productive, as Fodor & Pylyshyn demand, but also robust. It is hard to believe that a system could learn to effectuate whole series of transforms that could occur in any order by slowly learning each one independent of the others. It seems reasonable to depend on a more structured learning that can identify generic and essential features to act as keys in the transformations and to ensure that vital structure is passed on with fidelity.

To ensure this, we must define enforceable rules for the ways in which the compressed formats are composed and transformed. If these rules are specified (and hence enforced) at the symbol level, as they would be for a classic AI architecture, then the battle is already lost. A sub-symbolic architecture would be impractical if some external agency was required to police representations at the symbol level. Thus, we can take it as a constraint that the rules must be specified at a sub-symbolic (i.e. neural) level.

The only way to enforce rules at the neuron level is to build them into the fabric of the network by way of the network update procedure and the learning algorithms which change the strength of connections. This is where the notion of emergent computation comes into play: the rules which govern the systematicity of the whole are specified only at the local level. Global order must emerge from local ordering.

In the work of Chalmers and Chrisman, it was the lack of sufficient rules at this level which leads to a lack of control over the structure of the compressed layer during learning. This in turn is a limiting factor on the ability of the network to generalise.

11.1.3 Meta-information during Processing

An anecdote that seems to reappear from time to time in the public domain is that the main difference between a human being and a computer is that, when given a question for which it does not the answer, a human being will tell you that he does not know, a computer simply does not know that it does not know. This is of course a simplification, but it does bring to light another key issue that must be addressed by connectionist implementations of symbolic systems: dealing with what we might call “meta-information” during processing. Consider the following example.

Imagine a university department that decides to test two different systems to store details of every staff member. The first system is a Hopfield-like network in which each record is stored as an attractor state, while the second is an off-the-shelf database package for a PC, using a normal record format with fields and associated values. When the two systems are given a partially correct name, the network can often complete the name and return the department and office number of that person. The database also has algorithms that allow it to retrieve any name that matches most of the letters and produces a correct answer almost as often as the network. But what happens when the name given is nothing like any of the stored names? The PC checks against all of its records and returns a message like “no matching record”. It knows that it doesn’t know. The network on the other hand, has to settle into an attractor corresponding to the nearest stored pattern and therefore it eventually returns a valid result, unfortunately the wrong answer.

What is missing is information above and beyond the record information that was stored. This meta-information should be able to indicate that the network could not converge, and that the process initiating the retrieval must conclude that it doesn’t know the answer.

This is one example of meta-information. In their work on attractor networks, Amit and his co-workers simulated sequential processing using a Hopfield network as the main body of a state machine. At each timestep the network was required to settle into a new state corresponding to the appropriate output vector and new state. This was fed back for the next cycle. The failure of the network to reach a stable state after a fixed number of network updates was signalled as a “don’t know” state. (Amit, 1989).

There are two problems with this approach. First, the “don’t know” signal did not exist in the same representation space as a normal result. This means that the loop is broken between the network’s output and its input. The system cannot process the fact that it doesn’t know in the same way that it processes normal data. It needs a homonculus to watch the network state and to signal different behaviour if something goes wrong. That is a mechanism that we want to avoid. Secondly, a simple “don’t know” signal conveys little information that is useful in deciding what the system should do next. If we assume that it must be capable of deducing the cause for the failure and taking corrective action (perhaps seeking out new information or trying other memory cues in the case of retrieval failure) then the control process (itself a neural network) should have more information about what went wrong, including the opportunity to dissect a partial result.

[Note that in the architectural development, the use of context information in the symbol encoding (such as ‘Paris_city’) was suggested as a means of detecting failed convergence. The ‘_city’ tag, suitably encoded, provides expected content for the control process so that failure to retrieve something with this tag indicates that the recall is probably invalid. See section 5.4.5, page 126].

A second example of meta-information that needs to be addressed in the same manner as the “don’t know” signal is an indication of the difficulty that the system has in retrieving a piece of information from its memory. In a simple associative network, as discussed in detail in chapter three, each new traced stored in the network reduces the margin for the retrieval of other memories. In more complex storage prescriptions, such as learning hierarchies approach of chapter ten, there may be other impediments to retrieval, such as the inability to set-up the right cue, or degradation of the storage of access pointers. In all of these cases, problems or potential problems with retrieval must be brought to the attention of the system to permit corrective action to be taken. (Such corrective action might include re-learning or re-coding problematic memory traces).

A third and more subtle example of meta-information, related to the second, is becoming aware of the fact that a particular set of facts is difficult to learn. The system may have an expectation of the degree to which it should have forgotten the information after one presentation, say. Due to the re-coding of information in terms of existing information, as outlined in the previous chapter, it is likely that some new facts will be easier than others to relate to known data, leading to differences in learning efficiency. Any divergence from expected tolerances (for better or worse) would be useful information to bring to the attention of the system.

The fourth and final example of meta-information is, perhaps, the most important and its presentation will occupy the rest of this section. It is the opinion of this author that finding a mechanism to capture this class of meta-information in a more useable form is one of the key requirements (if not *the* key requirement) for closing the loop and allowing us to construct intelligent autonomous machines.

The critical focus for our attention should be the comparison operation between symbol structures: traditionally it produces meta-information but it must be remodelled as a subtraction operation over distributed representations of symbolic structures.

By comparison operation we mean a process taking two inputs from the same representation space (usually vectors in connectionist networks) and producing a scalar output indicating their similarity based on some metric. By a subtraction operation we mean a process which takes those same two inputs and produces an output in the same representation space as the inputs that is in some way an image of the exact differences between them. This output could be immediately reused as one of the inputs in an identical subtraction operation. As before, by meta-information we mean an output that is coded in a different (usually less descriptive) representational space than the inputs.

The reasons why the subtraction operation is so important are twofold: first it provides more detailed information on the differences between the inputs that can be used to trigger a more finely tuned set of follow on operations; but more importantly, the result exists in the same representational space as the inputs. This is crucial since it is a necessary condition to allowing *knowledge about knowledge* to be derived in an automatic manner. It permits the system to reason about what it knows. This was the key element that was missing from the marker passing networks of Lange & Dyer, Shastri & Ajjanagadde and Sun (see section 4.3, page 97). So the model of processing is now:

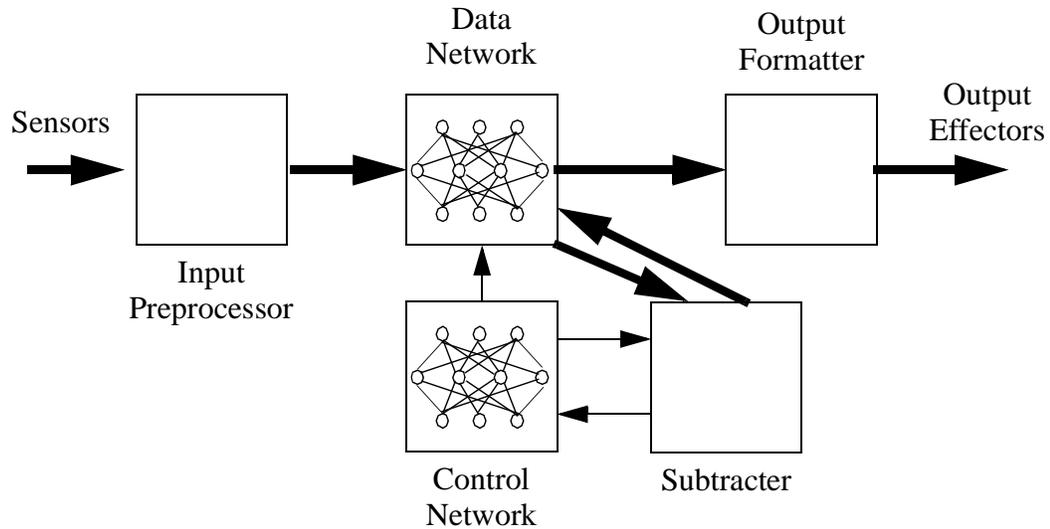


Fig. 11-1 Control of data network using subtraction.

In the figure is a revised diagram for the system. For this discussion we can ignore the input and output processing networks. Their function is to act as appropriate interfaces to the environment. Our concern is with the central loop, consisting of the control network, data network and the subtractor.

The control network produces patterns which guide the transitions of the data network (this idea was pursued in chapter nine although the results obtained thus far are not sufficient for our purpose). We would expect that the control network does not require (and would be confused by) the entire data pattern coming from the data network. In fact, it is concerned only with a set of key elements from the data structure which are needed to generate the next control pattern. Hence we require a notional third network.

The new network is located between the two main networks and acts as both a filter and a subtractor. It must filter the data output to remove the irrelevant details that make no difference to the computations required for control, leaving a skeleton of the structure. Then it must subtract a representation for the expected skeleton structure leaving a new structure, one which represents the differences. It is this structure which must somehow contribute to the decision of control pattern for the next "cycle".

The requirements for subtraction place new demands on the encoding and retrieval operations of the network. For encodings, we now require not only that each vector represents a valid symbolic structure but that even the subtraction of two such vectors represents a valid symbolic structure. Clearly, the choice of representation is now much more constrained. To progress further we need to know more about the mechanism behind the subtraction.

Two scenarios present themselves readily to mind that differ in the onus they place either on the representation or on the subtraction mechanism. At one extreme we could aim for a simple vector encoding and ignore any requirements for subtraction. This leaves all of the work to the “subtraction” algorithm and would probably lead to a highly complex operator, needing a large resource to execute it. For example, it might be required to process the vectors bit by bit, or extract features from each vector and correlate them to ascertain the probable points of commonality. Note that although this operation carries the name “subtraction” the operation it is required to perform may be considerably more complex than a standard subtraction between two vectors.

At the other extreme, we could devise an expansive vector representation, hand-crafted to make subtraction easy. The subtraction operator itself might be as simple as executing true vector subtraction between two multi-valued n -bit vectorsⁱⁱ. The price we pay here is that the vectors themselves might be much longer than those in the other case. They may also be ill adapted for use elsewhere in the network. In the manner of all engineering problems, we would try to select an appropriate compromise between the two extremes as the best balance between competing requirements.

The other constraint that the subtraction operation places on the architecture is in the act of retrieval itself. We view retrieval as the central operation of the network. Even computation is viewed as retrieval, although the patterns retrieved might never before have been produced by the network. If the output of the data network is often gibberish, then the subtraction that follows will presumably produce gibberish as the output. This is certainly an indication to the control network that something went wrong with the retrieval, but offers little insight into how to put things right. Thus, it is important to avoid producing a nonsense result from the data network. Even in the case of failure, we would like the pattern produced to carry some useful information. Enough to guide further useful operations from the control network.

Taking a step back, we note that the control network itself must also transition from one valid structure to another. If the control network fails and produces gibberish then the whole system state would soon collapse into chaos. There appears to be no network to monitor the control network and make changes to its patterns in times of difficulty. (*Quis custodiet ipsos custodes?*)ⁱⁱⁱ

ii. Although the use of K-from-N coding at the lowest level would require even the simplest of subtraction algorithms to be more complex than this.

iii. *Who guards the guards?* Juvenal, Satires, VI, 347

We are therefore obliged to ensure that the control network output remains consistent and coherent. Once we have devised a mechanism to do this, it should then be straightforward to reuse the same mechanism in the data network. This probably necessitates a particular way of working at the lowest levels of the system: the synapses, their dynamics and the learning algorithms that they employ. Earlier work in learning hierarchies, together with speculation of further enhancements to synaptic function, could provide a framework in which to realise the level of dependability and robustness needed to make such a system work (see section 10.1, page 255 and section 10.10, page 288).

11.1.4 Conclusions

This section has discussed a number of the results from the literature in terms of the objectives they were seeking to achieve and why the results they obtained fell short of those objectives. In doing so the aim was to highlight a few key areas in which further work is urgently required if we are to realise the potential of the connectionist approach.

The first such area was the co-existence of both rule-based and “intuitive” behaviour within the same network. Next we considered the requirement to have entire symbol structures emerge fully formed from the co-operative interaction of many neural elements.

The transformation of *meta-information* into a form that can be manipulated by the system is one such operation. The subtraction operation between symbol structures in vector form is a particularly important example of this category.

11.2 Issues in Hierarchy and the Limits of Real Components

The development of a neural building block suitable for the implementation of a general purpose neural computer has occupied most of the thesis. From the simple static K-from-N coding, through dynamic patterns to the complex synapses of learning hierarchies the goal was to optimise the performance of a fully connected region of N neurons with respect to the goals (such as memory capacity, reliability, etc.). Progress was made in this area although the work is far from complete.

The logical next step, having attained a working building block is to consider issues in their interconnection to form a complete system. Some important issues in this development are presented here, albeit in qualitative form.

The intention of this work has always been to view the neural building block as fitting into a general purpose module of two parts, each performing a complementary function. The first part is required to receive inputs from other such modules and transform them into a form that can be used internally by the module. The second part is required to generate an output which is of a form that is usable by other (structurally identical) modules in their own processing. This simple input/output structure is standard and unremarkable as an engineering concept. It also

seems to have relevance as an explanation of the structure of the so-called “hyper-column” of neo-cortical organisation (see, for example, Calvin, 1995).

For the purposes of this discussion, we will refer to the input part as the recogniser and the output part as the action generator. This section considers ways in which both structures must be modified to take into account real world constraints such as learning, maximum fan-in and redundancy. We begin by considering the ideal module, free from these extra constraints.

11.2.1 The Ideal Module

The ideal module consists of two layers of neurons (the white triangles) separated by a 1-from-N selector. Each layer is effectively infinite in size. Each neuron in the input layer receives data from all sensors. Each matches the pattern it receives with its stored vector, as in a Kohonen net. The winning neuron is that which has the highest potential. This neuron is allowed to fire by the 1-from-N selector. In the ideal case, using an input layer of infinite size, this is sufficient to classify any input vector into a unique 1-from-N code.

In the second half of the network the 1-from-N code is transformed into the required output (or action) vector. Output neurons make a connection from every neuron in the input layer (by way of the winner-takes-all network). Since the input to this layer is 1-from-N coded, the translation to the output vector is a trivial one.

Data flow in the module is uni-directional, from input to output. Each neuron in the input layer receives all of the information necessary to make the correct decision, there is no need for feedback or any other type of interaction between neurons (such as interconnections within the same layer).

This module can be replicated and whole sets of such modules can be interconnected to form more complex systems, as required. However, the fact that data from all of the systems sensors could be supplied to one module implies that a single (but ideal) module would be sufficient for any purpose.

This account of the ideal module does not address issues of learning (that is to say how the individual neurons came to exhibit the correct behaviour), as such considerations form part of the limits imposed by reality (which follow).

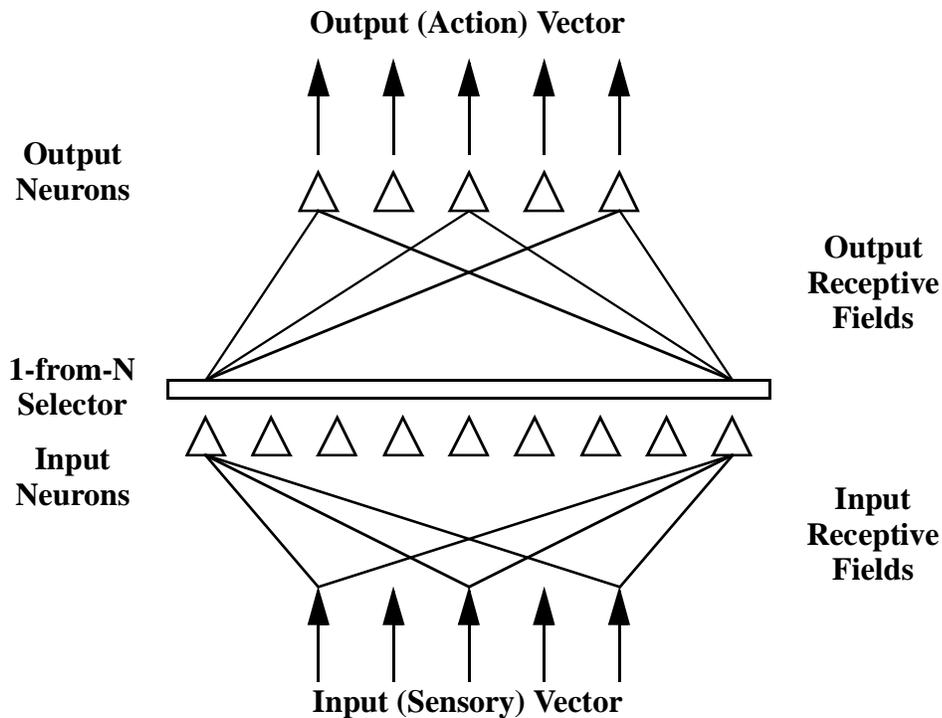


Fig. 11-2 The ideal module.

Now, we consider the changes that must be made to take the issues of engineering reality into account. The main areas that will be considered are layer size, fan-in, locality of data, performance, learning and redundancy. A counter requirement of generality is also discussed.

11.2.2 Limits of Layer Size

The most obvious restriction on the ideal module is that the layers cannot be infinite in size. Indeed, reducing the amount of resources required for the system is usually a key driver in the design process since resources equates with cost.

Therefore, the first area of investigation will be the effects of a finite layer size. Clearly, it may no longer be possible to provide a dedicated detector for each feature in the input space (we will make this assumption for the rest of this discussion, but it is a reasonable one for the type of systems we are considering). Two common solutions, often used together, are distributed representations and hierarchy.

The meaning of distributed representations is well known to most connectionists, but is redescribed here to help raise two important points. In a distributed representation, a single neuron will fire for many patterns, but the collective pattern across the whole layer is usually needed to uniquely identify the object being represented. The advantage of such a scheme is that, from only N possible 1-from- N codes in the original network, the number of distributed (binary) codes is exponential in N (chapter six gave more details on the coding issues).

There are two important problems which distributed representations introduce, however. The first is that modifying a neuron to learn a new pattern disturbs all of the learning that has taken place for other patterns. This is the stability-plasticity dilemma, about which chapter ten had much to say by proposing the learning hierarchies principle as a possible (but non-ideal) solution.

The second issue is that there is no longer a single point in the neural layer which can be used to identify the object being represented. The meaning is spread out. We would need to construct a (perhaps virtual) layer above the distributed layer to detect the pattern of '1's and '0's that make up a given valid code to 'refocus' the meaning back into a single neuron (to rebuild a 1-from-N code). But by extending the argument for this module to all the other modules in the system, it is clear that creating this refocusing layer serves no purpose. Therefore, the network should be composed entirely of interacting layers using distributed representations.

Turning to the other means of better using finite resources, that of adding hierarchy, we see that such an approach depends on finding sub-patterns of commonality between input patterns (features) and using the first layer neurons to extract them. Subsequent layers perform the same task but on the output of the level below. In order for this to work, there must already be structure in the input patterns themselves so the success of hierarchy as a technique for improved efficiency of resource usage depends on the existence of redundancy in the input vectors.

11.2.3 Limits of Fan-in

In real systems, individual neurons can not have infinite fan-in. If we are to build a large network (that is, one with perhaps millions or even billions of neural units) it will be almost impossible to allow every neuron to make a connection with every other. Thus, we will assume that the ideal module, in which every neuron on a given hierarchical level receives input from every neuron on the level below, is unachievable and we must restrict the inter-neural connectivity. This can be done by breaking down a single layer of high fan-in components into a hierarchy of lower fan-in components. Such a technique is employed in electronic design for logic synthesis (see, for example, Weste & Eshraghian, 1992). Reducing the maximum fan-in of each component always leads to an increase in the number of levels of hierarchy.

Restricted fan-in introduces a new problem to the design process. It is now no longer possible for a single neuron at one level to see all of the information at the level below. Therefore, winner-take-all processes must operate only on sub-populations of the entire layer and it is left for neurons higher in the hierarchy to bring together the results of disparate groups in order to make a unified decision.

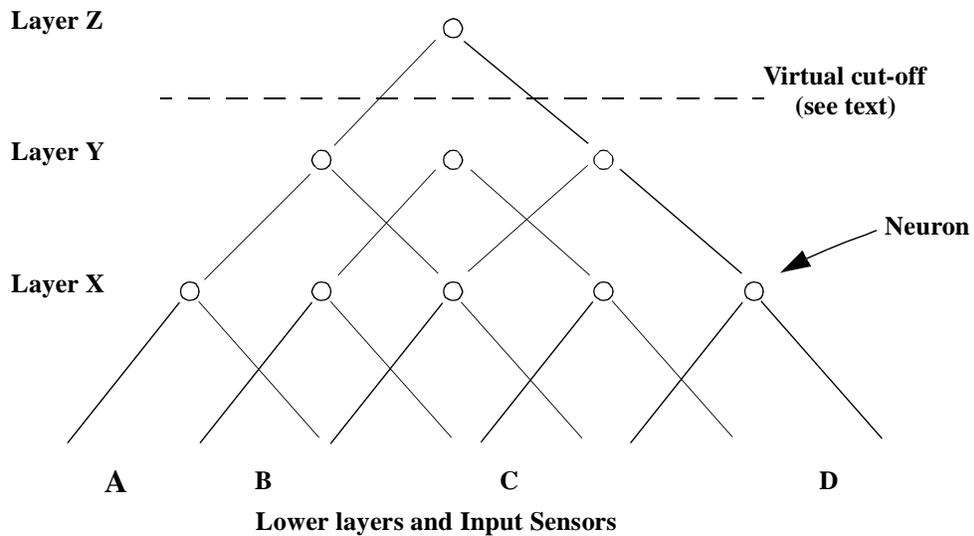


Fig. 11-3 Hierarchical Network with Limited Fan-in Neurons

The figure shows how the receptive fields (as they are referred to in neuroscience) of the neurons are limited (to three neurons in the layer below in this example). The three layers X, Y and Z could occur anywhere in the hierarchy. So the input to layer X could be directly from the sensors or from a lower level. The letters A, B, C and D refer to signals arriving in the different receptive fields. They could be from sensors or from lower level neurons, as noted above.

Note that the hierarchy has been drawn as a triangle, with fewer neurons at each higher level. This need not be so. The actual numbers would depend upon the complexity of the input pattern structure (a point we come to in the next sub-section).

To detect a feature which depends on sensory events A and B, a neuron at level X is sufficient since the two signals are within the receptive field of one neuron at level X. A feature which required the simultaneous presence of events at A and C would only be detected at level Y since the data from two level X neurons must be combined. Similarly, features involving events A and D could be detected first at level Z.

The virtual cut-off represents the fact that the hierarchy does not reach the level at which items are represented as 1-from-N codes, as described for the ideal module. Instead, we must reach a level in which the coding is still distributed but no further merging can occur. Either this layer contains few enough neurons for a single K-winners-take-all mechanism to produce a solo representation of the object (as a K-from-N code) or if not, some way to combine the results laterally in multiple steps must be found. The third alternative is to allow parts of the output code to remain unaffected by other parts so that a form of schizophrenic coding occurs. We take this choice to be unacceptable since the system will not be operating as a single entity.

As it turns out, the problem of handling the virtual cut-off resolves itself as a by-product of resolving the final question in this sub-section. It is this: if we wish the module to be general, how can we decide in advance how many neurons to put in each layer of the hierarchy and how many layers we need?

Trying to add extra neurons as the need arises (an approach taken by the many ART networks of Grossberg, (such as Grossberg (1976a)) is counter to the philosophy of this work since one of the main goals was to build a general purpose system which could be physically implemented. Adding neurons on a module-by-module basis is not in line with this strategy. This dilemma gives rise to the counter-requirement of generality.

11.2.4 Counter-requirement of Generality

The last sub-section described how a hierarchy makes it possible address real-world constraints on the network design but seems to force the designer to allocate resources in the form of the number of layers and the composition of each layer before the network has been constructed, perhaps even before the target domain of application is known. This is something we wish to avoid, if possible.

The alternative solution is to collapse the hierarchy and move the burden for maintaining its function from the neural level to the synaptic level. This can be done conceptually in two steps as shown below overleaf.

In the first step, the hierarchy is collapsed to form a single layer. Note that the function of each neuron is preserved, as is the connectivity (not shown in the figure). This step represents the mixture of neurons into a single layer. Adjacent neurons on the same level may now be representing features at different levels of complexity. But the mix of neurons (i.e. how many are from each level) and the hierarchical connectivity between them are still preset, so the goal is not yet achieved.

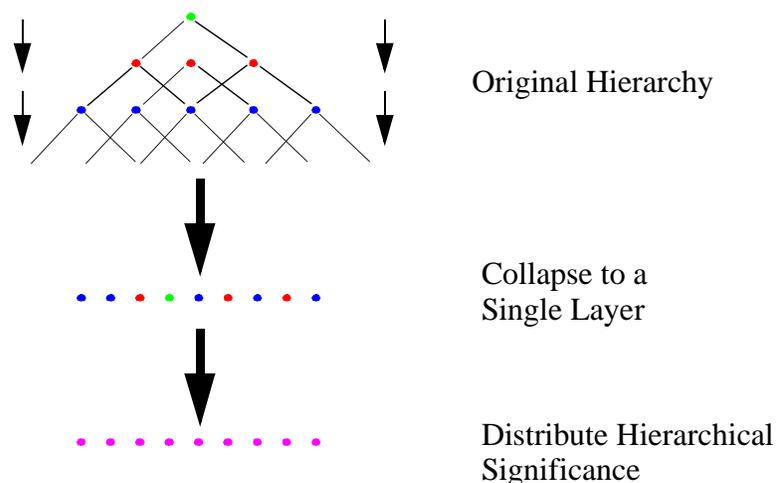


Fig. 11-4 Collapsing the Hierarchy to Preserve Generality

In the second step, the hierarchical label for each neuron is removed, so that any neuron may now represent a feature at any level of the hierarchy. Neurons are now fully inter-connected. An incoming pattern of activity begins by activating neurons corresponding to the lowest level of feature detectors. These, in turn activate neurons on the second level due to internal connectivity, and so on up the virtual hierarchy. To an external observer the act of recognising an input pattern would appear as a spatio-temporal pattern playing across a single layer of neurons.

The final pattern achieved would need to correspond to the output of the highest level of virtual neurons. It might be static or dynamic, depending on the architectural design. Note that for one pattern a particular neuron might fire to represent a low level feature while for another pattern that same neuron might represent a high level feature. Thus, we have extended the notion of distributed representation, allowing a distribution across hierarchical layers to take place. Just as for standard distributed representations there are issues of cross-talk, but in a similar fashion a large enough layer can theoretically take care of any given level of such representational ambiguity.

This approach should be compared with the development of the neural building block in chapters six to ten. A single layer of neurons was developed that performed a pattern recognition and could produce a temporal pattern of activity as the output. The more complex synapses of learning hierarchies should (with further work) provide the means to capture the fact that a hierarchical computation is taking place within a non-hierarchical module of neurons.

Overall, the goal is that a fixed module of neurons can capture of hierarchy of variable size and complexity by modifications of the synaptic connections. It is in the synapses that we capture the variable aspects of the hierarchies structure. This is a much more efficient place to perform this function for two reasons. First, because no new neurons are added and we assume that neurons and their connections are costly to add. Second, because the synapses will already be present and we are merely changing their values with learning.

11.2.5 Requirement of the Locality of Data

In the construction of real neural networks (rather than mere simulations) the high degree of interconnectivity between the neurons causes a considerable design constraint. This is as true for real brains as for artificial ones; the volume of the human brain is dominated by the white matter of axons, while the computational grey matter forms only a thin crust around the outer layer of the cortex (see any textbook on neuroanatomy, such as Martin(1996)).

Even though we have insisted that layers are of finite size and that neuron fan-in must be restricted, so far there has been no restriction on the distance that any given axon may travel to make connections with other neurons. Neuro-anatomical evidence suggests that the pyramidal neurons, which make up the main information processing and propagation circuitry of the neo-cortex, make connections in particular patterned ways (Calvin, 1998). Connections are either local (between neighbouring neurons of mini-columns in the same layer), near-local

(groups of neurons systematically connection to a neighbouring gyrus, perhaps a few centimeters away) or distant (groups of neurons systematically connected to the opposite hemi-sphere or to a sub-cortical structure). The dominant connection type is local or near-local.

Making connections is costly and resources spent in communication do not contribute directly to the computational output of the system. Therefore, a sensible designer would attempt to minimise the distance between communicating entities and to keep such distances in inverse-proportion to the level of communication expected (to first order). This would imply placing functional units that were highly interconnected closer together than units that have little or no communication paths between them.

11.2.6 Requirements for Attaining a Stated Level of Performance

In real-world applications we may find that some tasks can be performed in an acceptable period of time by a general purpose network acting serially. Other tasks will be sufficiently computationally intensive or will be needed so frequently that it will make sense to provide dedicated hardware to subserve them.

Looking at the mammalian brain we see that a large proportion is devoted to vision. Even within that sensory modality there are parallel entities dedicated to colour, movement and line extraction (Hubel & Wiesel, 1977). Clearly, the need to maintain constant visual processing forced nature to dedicate a considerable proportion of brain resources to that function.

We expect that a general purpose machine acting in the real world would also need to dedicate hardware to function such as vision and so we must be prepared to construct systems of many interacting modules with some dedicated to special functions and other more general in purpose. Recall that a similar approach is anticipated by Aleksander in his Magnus project (reviewed briefly in section 4.4.5, page 113). The eventual system is envisaged as a set of networks each dedicated to one sensory modality, interacting with a central network which acts as the general purpose integrator and planner (Aleksander, 1996)).

Creating dedicated modules to handle specified functions demands that care be taken in determining how the modules interact; whether one dominates the others, *etc.* It would seem reasonable to allow the vision module to drive the memory module during the memorisation of a scene but during the recall of that scene the route of data would be in the other direction. Considerations of this sort are left to future work.

11.2.7 Requirements for Learning

The more complex the structure of the input patterns, the more difficult it will be for a simple learning algorithm to capture and internalise it. This places restrictions on the size and complexity of the input patterns to the module.

The development of learning hierarchies in chapter ten was intended to address the main issues in learning for neuro-symbolic systems. The underlying

assumption with the learning hierarchy approach is that the input vectors do have structure, containing a hierarchy of subsymbols that recur in different orders and positions. The algorithms derived from the basic concept have a common base, that of mimicking the structure of the data in the structure of the synaptic connections. Since there are several parallel connections at work on several timescales, there is an opportunity to extract and learn regularities on several levels simultaneously.

If the learning hierarchies approach differs in any way from traditional learning methods it is in that there is no perfect representation to which the network tries to slowly converge. It can quickly learn an association on one 'level' and while simultaneously making progress to learn the association on the others. Thus, complex feature detectors are slowly tuned as they would be for an MLP, but the network does not need to wait for them to converge to 'correct' values in order for the current learning event to take place. In any real-world application this is an important and desirable property.

11.2.8 Requirement for Redundancy

The final facet of the discussion on the limits imposed by reality on the ideal module is an obvious and simple one: that of redundancy of representation. The 1-from-N coding proposed for the ideal module allows no room for the failure of a single neuron. Any such failure results in a completely different neuron winning the competition to interpret the sensory data. As discussed in the review of the Kohonen network in chapter two, this places the onus on the next stage of processing (which uses the result of the 1-from-N unit) to produce similar outputs for similar inputs to attempt a recovery (see section 2.3.3, page 31).

Redundancy allows us to avoid these kinds of problems. This was one of the drivers for the choice of K-from-N coding for the lowest level of coding in the simplest neural building block (chapters six and seven).

11.3 On Escaping the Bottleneck of Forward-chaining Rules

The use of neural networks was inspired, at least in part, by the desire to escape from the straightjacket which is simple rule-following behaviour. Many AI researchers who turned to connectionism in the eighties did so because, no matter how many rules were stored and applied in parallel, they could not bring themselves to believe that a thinking machine like the brain could be practically realised in a purely rule-based system (one notable holder of this view being Rumelhart as described in Rumelhart & McClelland, 1986).

And yet, more than a decade after trying to break away from the rule-based production system architectures the focus of attention in reasoning systems is still exactly that: rules-based, but now implemented in neural networks instead of purely symbolic computers. This was a major criticism of the marker passing architectures of Lange & Dyer, Sun and Shastri (section 4.3.4, page 105).

The intent of this section is to re-iterate the fact that alternatives exist and to suggest that far more powerful (and flexible) reasoning systems could be constructed simply by being more bold in bringing together alternatives drawn from the wealth of already existing approaches. The assumption is that by careful study of some alternatives we may find much commonality that will lead to an efficient implementation. However, this section does not constitute an in depth survey of such alternatives (which is left for future work).

11.3.1 Inverting Rules: From Effects to Causes

The most obvious place to start is to lift the restriction of using rules only in a forward chaining mode. The original restriction made sense when we demanded that the rules of logic be followed strictly. It is also useful as a means of generating actions, as in production systems. Forward chaining, as noted in chapter two, corresponds to the deductive inference, also called *modus ponens*. Consider the following rule:

$$\text{fatherOf}(X, Y) \text{ AND } \text{fatherOf}(Y, Z) \Rightarrow \text{grandfatherOf}(X, Z)$$

Deductive inference allows us to reach a conclusion relating X and Z if the two conditions on the left are fulfilled for some X, Y and Z. In this case, the inverse rule is also safe under *modus ponens*, but could introduce an unknown. If we had a concrete example:

$$\text{grandfatherOf}(\text{John}, \text{David}) \Rightarrow \text{fatherOf}(\text{John}, Y) \text{ AND } \text{fatherOf}(Y, \text{David})$$

which is similar to a case studied by Sun (section 4.3.3, page 103) in which making the inference has introduced an unknown. We know that if David is the grandfather of John then there is someone else (Y) who is the father of one and the son of the other. We need to be able to represent that fact, and if it is relevant, dedicate computing resources to identifying who Y is.

In most cases, however, inverting the rule does not lead to valid inference. The general rule, such an inference is an inductive one, since we are making a statement that may or may not be true. For example, the first rule holds but the second, created by inverting antecedents and consequents does not:

$$\text{TRUE: } \text{drop}(\text{glass}) \text{ AND } \text{material}(\text{floor}, \text{concrete}) \Rightarrow \text{broken}(\text{glass})$$
$$\text{FALSE(?): } \text{broken}(\text{glass}) \Rightarrow \text{drop}(\text{glass}) \text{ AND } \text{material}(\text{floor}, \text{concrete})$$

We can be fairly certain that dropping a glass when the floor is made of concrete will result in a broken glass. However, working the other way, the fact that the glass is broken does not imply either that it was dropped or that the floor is made of concrete.

It seems to be often the case with reality that many causes can lead to the same effect, so that reasoning from cause to effect is safer than reasoning from

effect to cause. Even so, it would be wasteful to ignore information that we could obtain from the inductive leap from a given outcome back to its root causes.

Each of us, drawing on our own everyday experience, knows that we make inductive leaps of this kind as a matter of course. Upon learning that John will be late arriving by plane from Aberdeen, we might assume that his plane was late or there was a traffic jam from the airport. An intelligent system might benefit from generating possible hypotheses such as this and using them to help plan new courses of action (such as asking further questions to clarify assumptions).

How might it do this? If we take a rule as being represented as a pattern of activity then the normal flow of processing (as used in deductive inference) is to identify which conditions on the left hand side of the rule match for a given set of variable bindings and use this match to retrieve the pattern for the right-hand side. (This approach was used in Austin's AURA network, as described in section 4.4.1, page 108 and is also the basis for production systems and expert systems). We can easily extend this model by allowing matches to take place on the *right hand side* of the rule instead, using this to retrieve the corresponding left-hand side.

When a match occurs, the resulting symbol structure is the LHS of the rule, with the appropriate variables bindings propagated from the RHS to the LHS and some measure of the degree of confidence that the system has of the truth of the result. As noted above, there may be unknowns generated for some variables, but this is something we must deal with in the forward chaining case anyway (as discussed during the review of Sun's CONSYDERR system, section 4.3.3, page 103).

There are two points to note here. First, the idea of attaching a confidence value to the output of an inference can also be applied to the normal forward chaining case, replacing strict deductive inference with a softer representation based on degrees of belief.

Secondly, the whole proposal being made here is already described at the symbolic level in belief networks. The idea behind belief networks is the model the probabilistic cause-and-effect relationships between predicates leading to a quantitative theory to explain and predict events (Russell & Norvig, 1995, chapter 15). All that remains for us is to properly integrate them into the neural framework. Using a pattern matching technique to identify matches (or potential matches, in the case of missing elements) seems like a reasonable way of linking the two but there has been little work in this area to date. Further development of these possibilities is left for future work.

11.3.2 Modelling Multiple, Exclusive Consequents

Another area of rules-based logic which seems to be generally ignored by the research community is the modelling of multiple possible conclusions. For example consider the following instantiated rule:

won(Fred, \$30,000) => buy(Fred, sports_car) OR repay(Fred, mortgage)

Here the fulfilment of the conditions (Fred winning \$30,000) allows us to conclude either that he will buy a sports car or that he will pay off his mortgage. Such multiple, exclusive conclusions present a major problem as they lead into the multiple worlds scenario that will be discussed in more detail later (section 11.4.2, page 321).

If this rule were part of another intelligent agent's knowledge base, it could be used to assess the probability with which Fred might take either course of action (perhaps requiring a search for other information commensurate with that agent's goals). If it forms a part of Fred's own reasoning system, the rule could be viewed as an option generator to allow Fred to decide what to do with his money.

As a final point, note that modelling of multiple consequents must also include the distinction between inclusive and exclusive OR. In this example, Fred might be able to buy a car and also to pay of his mortgage with the money he has won. In other examples, such as how Fred will spend his evening, he might be restricted to choosing a single answer from a list of alternatives.

11.3.3 On Not Hardcoding First-Order Logic

Many AI systems, such as have been described at several points in this work., use first-order logic at the core of their processing. The benefit of hardcoding such logic is that the results are always logically valid. Such concepts as 'for all' and 'there exists' are formally defined and there are well defined transformations that can be applied to transform sentences of one type into equivalent sentences of the other type.

But as has also been noted many times, the restrictiveness of first-order logic has caused it to be abandoned as the core of our system development. In the connectionist approach predicates (encoded in a certain way) are associated with other predicates to form rules. But concepts such as 'there exists' and 'for all' have now been lost at the lowest level of processing. Is it possible to recapture them, but at a higher level? And if so, what would that mean?

There is already an example of work in this area, once again that of Aleksander in his investigations into artificial consciousness as reviewed in chapter four (section 4.4.5, page 113). He postulates that the universal and existential quantifiers should not form a fundamental basis for representing relationships between predicates and objects, as they do in first-order logic. Instead, concepts such as 'all men' must be learned by the system in the course of many trials, perhaps by induction with many examples involving objects all belonging to the category 'men'.

The system creates an icon which refers to the concept of the group of all men and can use this icon in other relationships such as 'all men are mortal' in the same way that it can represent 'Socrates is mortal'. Thus, unlike most reasoning systems, Magnus is not hardwired for first-order logic. This is a very appealing idea for a number of reasons.

Firstly, it breaks a hard-wired mechanism that was restricting the potential richness in the relationships between concepts. Secondly, concepts which resemble ‘all’ but which do not exist in first-order logic (such as ‘a few’, or ‘most’) could be constructed in a similar way to ‘all’.

In the earlier discussion of rules-based logic, the generation of unknowns was mentioned as a potential source of difficulty. Such unknowns are normally handled in first-order logic by the existential operator: ‘*there exists y such that...*’. So in this network, we also need to represent the existence operator but in a richer manner. It must be richer since we require the reasoning process to encode the fact that a new variable has appeared, that it has no assigned value but is not unconstrained. We might include a mechanism to identify potential values for *y* from several independent pieces of evidence, perhaps implemented as a recognition process. For example, there could be multiple activated rules in working memory of the type:

grandfatherOf(John, David) => fatherOf(John, Y) AND fatherOf(Y, David)

and it remains for the system to track down values for *Y* which satisfy all conditions in parallel either by retrieval from memory or by seeking out the information in the outside world.

11.3.4 Analogical and Case-Based Reasoning

An area of fruitful research in the 1990s has been the case-based or analogical approach to reasoning in which the intelligent agent explicitly draws upon knowledge of previous experiences it has had and situations it has “encountered” to provide support for its decision making in novel situations (Leake, 1996, Kolodner, 1993). Newell’s Soar architecture is similar in this respect (Newell, 1990; and this thesis, section 4.1.1, page 86).

From the point of view of this work, the appeal of an analogical reasoning system is that it is based around a recognition task and neural networks are good at such tasks. In case based reasoning, for example, the first step of solving a new problem is to extract key features such as goal, context and constraints and try to match them to known cases. Having identified candidate cases, the previous solution can be retrieved and (if it was successful^{iv}) applied. Of course, modifications are often needed since a new situation is not guaranteed to conform to the old one in every respect.

Not only is the search by pattern matching suited to neural networks in general but we can also find parallel between the identification of differences between the old and new situations and the symbolic subtraction discussed at length in section 11.1.3, page 298. Such a mechanism is required not only to establish that the two cases (old and new) are not the same, but also to pinpoint the main areas of difference.

iv. The solution might be applied even if it wasn’t successful last time, if the system can identify what went wrong before and can either correct for it or decide that the previous impediment does not exist in the new situation.

11.3.5 More Flexible Recall

As has been described earlier, most rule-based systems follow the path from cause to effect. A generic rule defines how a given set of input variables that conform to certain types can be transformed into another set of variables that conform to certain types.

It is worth pointing out that the architecture put forward in chapter five treats the input, output and transforming symbols as the same from the point of view of the association process. Three symbols are brought together and any two can be used as key to retrieve the third.

In the simplest case of an instantiated rule, a set of input variables (with values bound to them) are transformed by a generic rule to produce a set of output variables (with new values or values related to those bound at the input). This is the case in production systems and the reasoning architectures based on marker passing, detailed in chapter four. We can generalise the approach so that, in addition to the production system like mechanism of:

input variables and bindings + rule => output variables and bindings

we could also reason from effect to cause using:

output variables and bindings + rule => input variables and bindings

or perform a sort of explanation based reasoning using:

output variables and bindings + input variables and bindings => rule

It is left for the control network to correctly interpret the output. During ‘rules-based’ processing this task is made easier by the fact that it is the control network itself that has set up the retrieval in the first place and is expecting output of a certain form, as discussed in section 5.4.6, page 126.

However, as will be discussed next, trying to execute intuitive processing will render the results more difficult to interpret since there is less control over their generation. The onus is then placed on the result to be self-explicit. That is to say that while the resulting symbol structure may be grounded (so that they can only be interpreted in terms of other symbols) there is no need for state in the control network to help interpret the operation for which this is the result. Aspects of intuitive reasoning will be treated next.

11.3.6 Bringing Together Rules-based and Intuitive Processing

As noted near the beginning of this chapter, one of the key drivers in attempting to use neural networks as the substrate for symbolic processes is the belief that such an approach will lead to a better explanation of both rules-based and so-called “intuitive” processing. While it might be possible to construct a system in which each of the two types of processing was served by a uniquely crafted network so that there was little similarity in the way in which rule-based and intuitive process-

ing take place, we take the alternative approach here and assume that a single framework could be constructed that could exhibit both types of processing and a continuum of intermediate types. This is desirable both because it is likely to be more efficient and because it is probably more flexible than an architecture that separates the two processes.

What would such a framework look like? To understand this we need to know what behaviour we require from each end of the continuum and to generate a single network which embodies both. Rule-following behaviour is the easier of the two to define, not only because of the depth of scrutiny rule-based systems have undergone over the past forty years, but, more fundamentally, because rules are built on language and language is a medium in which we can readily express ourselves precisely.

A rules are traditionally made up of a number of conditions and a conclusion (or list of actions). If the conditions are satisfied, the conclusion is justified (or the actions are executed). By way of contrast, attempting to formalise the “intuitive” level is much more difficult. Even though one could argue that, by definition, the intuitive level cannot be described in any formal manner, here we take the physicalist view that intuitive processing is still a mechanism (albeit one that operates at a lower level than that of symbols). Thus, we should in principle be able to find an algorithm to describe it.

Intuitive (or lateral) thinking is often seen as not conforming to the laws of logic. It might be generally described as a process in which links are made between ideas in a manner that are not logical, but that the results obtained from such a process are relevant in the real world, and may even be logically coherent in spite of the process used to produce them.

Note that by demanding that the results of intuitive reasoning are relevant to the real world we allow conclusions that cannot be logically derived from the given facts (even by working backwards logically after the solution has been obtained intuitively) but exclude nonsense results that are the product of insanity (such as intuitively deciding that the way to cross a river using only a length of rope is to throw the whole rope in the river and watch as it floats away). Thus, while intuition is allowed to be illogical it must at least have the potential to generate valid results.

Note that by this definition we have excluded such non-verifiable types of intuition as artistic temperament. While valid in their own right, our inability to logically validate any conclusions resulting from such ‘computation’ puts this kind of intuition outside the scope of engineering discussion at this time.

The mechanisms of human intuition are not known today (any more than the mechanisms underlying rule-based processing) and will no doubt be difficult to establish. The connectionist approach to modelling the phenomenon often uses constraint satisfaction as a means of performing intuitive inference. In this approach, the process of solving the problem consists of finding the most mutually satisfying (in terms of energy) values for the free variables by defining an energy function which embodies the constraints. The Hopfield network as applied to the

solution of the travelling salesman problem and (more generally) applications of simulated annealing are examples of this style of processing (Hopfield & Tank, 1985). Such methods are a form of search through the space of possible solutions. The use of an energy function and an appropriate update rule for the neurons is to try to ensure that resources are concentrated in more plausible parts of solution space than would be the case if an exhaustive search were being executed.

But such examples seem hardly to describe the ways in which everyday problems seem to be solved intuitively. The main restriction of such methods is that the set of constraints must be known in advance. A vital aspect of intuition that is, arguably, missing from such a formalisation is it is often the case that even the method of finding the solution is missing and that constraints are added to, and removed from, the set of working constraints in a dynamic fashion as one works through a particular problem. Thus the fluid nature of the developing solution must surely be taken into account in any plausible explanation of intuition.

A additional problem with putting all of the constraints together in one energy function and searching for the global minima is that a real-world problem often has so many degrees of freedom that finding a solution is often computationally intractable. We can see parallels with the comment made above about the fluid nature of our own intuitive processes. Perhaps as intuitive thinkers we are capable of breaking down a large number of variables into groups, fixing the values of some and exploring the energy function while varying the others. The results of such steps could lead to changes in which variables have their values fixed and which are allowed to be free floating. Such changes would need to be orchestrated at the sub-symbolic level to continue to qualify the process as intuitive (although, as noted below, we would want to allow both symbolic and sub-symbolic to interact at appropriate times).

How could a system exhibit both types of behaviour in the same piece of hardware? Let us again consider both extremes of the continuum, using the production system as a framework for this discussion. First, we can examine rule-based behaviour in its strictest form. In the course of processing the system admits only true facts into its working memory. If the conditions are not fulfilled exactly, the rule does not fire and its conclusions are not accepted as new facts. In the network, new associations are made strictly when it is decided that it is valid to do so. Results do not fade as they must remain in working memory until the current task is complete or until explicitly removed.

In situations where the method of solution of a problem is known and can be executed at the rules level, the network can easily apply its rules in serial order, guided by the control network which decides which rule to apply. When such an algorithm does not exist, the control network must be less rigid with the choice of avenues that it explores. We assume here that the same knowledge base acts as the source of raw information for both intuitive and rules-based processing, but the former case resources are being used much less rigidly. Thus, rules whose conditions are not completely filled could still fire, only with less than 100% strength.

The overall effect is to allow the system to explore avenues not justified on strictly logical grounds but still more likely to succeed than a purely random search through all possible combinations.

As a final note, in domains where a logical double-check is possible, we could conceive of a system which uses intuitive processes to generate possible solutions and then use more reality-based logic to verify the conclusions reached. There might also be much benefit in allowing rule-based and intuitive processes to interact in the course of generating a solution.

Even if the intuitive process was simply one of constraint satisfaction, it would be useful to allow a rules-based process to add and remove constraints, or fix variables to restrict the search space. Such decisions could, themselves, arise from conclusions reached during intuitive processing.

To summarise and conclude this sub-section, it is postulated that it should be possible to represent both rules-based and intuitive processing in the same neural substrate. The way in which the network was used would determine the type of processing exhibited. The demands of intuitive processing are probably as evident in the control network as in the data network and an important part of realising intuitive processing will be in understanding the way in which constraints and possible solutions are explored and discarded dynamically. This is nominally a control issue.

11.3.7 Conclusions and Discussion

This section has presented and discussed a number of existing alternatives to the forward-chaining mechanism that currently dominates rule-based processing both in AI and (with less justification) neural reasoning systems. Despite a current lack of focused attention, these other mechanisms not only bode well for more flexible reasoning, but also seem (to a greater or lesser extent) well suited to implementation within the neural network developed in this work.

More fundamentally, the introduction of the idea of releasing the system from the constraints of first-order logic, although not strongly represented in this thesis, has always been a fundamental tenet of the system and will probably play a larger role in future work than it has done so far. The implication of such an idea is that logical behaviour is something that the system must learn like any other activity. Researchers are, arguably, too entrenched in first-order logic as the way to model the world. This is understandable since every system needs a core of hard-wired functions from which to build more flexible functionality by composition. First-order logic is intuitive and parallels a way of thinking that has existed since antiquity. The objection to it here is as the source of the core set of operations. In this work, what will replace logical reasoning as the means of finding ordering in the world is the extraction of similarity between objects and (just as importantly) events. For events, we assume a modelling of consistent cause-and-effect as the main mechanism. More work is required to justify this view.

11.4 On Parallel Processing at the Symbolic Level

One of the much vaunted properties of neural networks is the potential ability to represent and operate upon many pieces of information in parallel. However, in all practical networks the encoding methods employed, together with the overall system architecture, impose limits on the true degree of parallelism obtained. It is almost expected of a neural network to perform operations in parallel. Parallel search is at the heart of simulated annealing, for example, which is itself the core of the Boltzmann machine.

However, when we consider how neural networks have been applied to rules-based architectures, the general rule is that a parallel search of the knowledge base is followed by a serial selection of a winning rule to use (for example, Hinton and Touretzky's distributed connectionist production system and Austin's AURA architecture).

The serial bottleneck is something that we would like to avoid to improve system efficiency. Is this possible? This section examines some of the issues in trying to operate on multiple symbolic structures in parallel and considers ways in which the symbolic level might be parallelised without merely replicating the network to mirror the parallelism of the symbols. Thus, the issue is one of improving the efficiency of a single network.

This section acknowledges the fact that attaining the required behaviour requires us to consider issues at both the symbol level and at the neural level. We begin with the symbol level to establish the types of operations we would like to perform in parallel then consider the lower level problems this creates later on.

11.4.1 Rules as Transformations

As far as implementation is concerned, we can think of a rule as merely a transformation of one framework into another, with a corresponding transfer of variables and their bindings from one to the other. For example consider the rather contrived and specific rule:

$$\text{missed}(X, Y) \text{ OR } \text{broke_down}(Z, Y) \Rightarrow \text{late}(Y)$$

Assuming a production system structure of rules memory and working memory, then given that either the true predicate *missed(plane, John)* or the alternative *broke_down(car, John)* are present in the working memory the following instantiation of the rule will occur:

$$\text{missed}(\text{plane}, \text{John}) \text{ OR } \text{broke_down}(\text{car}, \text{John}) \Rightarrow \text{late}(\text{John})$$

allowing the new predicate *late(John)* to be placed in the working memory. In the implementation, a match is performed between the conditions required to fire each rule and the contents of working memory. When the rule fires for a given set of variable bindings, a new predicate (*late*) is created (set to true) with a variable binding ($Y=John$) propagated forward from the conditions.

The bottleneck to processing is that (usually) on a single rule is allowed to update the contents of the working memory at a time. This decision making phase consists of a number of parts, each requiring resources. The winning rule must be selected, its consequents (*late(Y)* in this case) must be retrieved the value of its variables (if any) must be attached to the new predicate and the whole must be written to the working memory. This could be done for multiple rules in parallel but each such rule would normally require parallel hardware (extra rule-firing decision logic, extra variable binding propagation paths, extra ports into the working memory, etc.).

It is also clear that it would be difficult to allow multiple rule to fire if one is dependent on the outcome of another. In other words, forward chaining in which the consequent of one rule forms the antecedent of the next would still be executed in serial order on the type of production system we have described. Later sub-sections consider briefly how the need to reason about changing the state of the world impacts the achievable parallelism.

The same arguments that have just described rule-based behaviour can also be used for the more complex *schemas* of Arbib (1994), and to the yet more complex *cases* of case-based reasoning (Kolodner, 1993). We note that as the rules become more complex, the frequency with which all of the conditions are met for a given ‘rule’ is reduced. We are therefore obliged to reduce the level of strictness applied in rule matching.

In schema theory, each individual schema can be activated with a certain confidence value based on the degree of match of its arguments. This value can, in turn, propagate forward to other schemas. Note that many schemas, representing many potential interpretations of the same set of facts, can be active at once with decisions being made based on the schemas with the highest degree of confidence within a competing set of interpretations. This allows chains of dependencies, quantified by the parallel chain of confidence values to be established. Decisions can be made only at the end of chains so that evidence from many possible worlds (interpretations of the facts) can be considered in parallel. There is much in schema theory that is appealing but it will be argued a little later in this chapter that it is a simplistic view of processing which is unworkable as a solution for general purpose intelligence because the lack of decisions leads directly to a lack of control over scarce resources.

For case-based reasoning, partial matches on stored cases are allowed, but this demands the introduction of an accommodation phase to map the stored actions of the winning case to the current situation to take account of the points of difference between the old and new situations. An advantage of the case-based approach is that the “unit-step” in forward chaining can be very much larger than for a simple rule. What this means is as follows: first, we recognise that a case could have been constructed at some earlier time by a long series of logical steps and then stored as a single unit. During recall the mapping of one case to the current problem eliminates the need for the sequential series of steps. It is an example of *chunking*, a term used in psychology and discussed earlier in relation to the Soar architecture (section 4.1.1, page 86).

The use of cases is one way in which parallelism (on this occasion, a distributed case-searching algorithm) can replace a section of sequential logic, but it is also quite clearly a process that is open to error due to non-exact matches. Without actually executing the sequential operations in the current context it might not be possible to identify certain crucial points of mismatch between the current problem and the stored case. A single such point of difference might render inappropriate the choice of that particular case for the current problem.

Overall we note that chunking on its own is a technique that, for all of its advantages of efficiency, must be backed up by more fine grained analysis. Due to its efficiency, we postulate that chunking (in one form or another) is a fundamental tool in the architectural arsenal and its use should at least give the impression of executing multiple rules in parallel even if the true mechanism being applied is somewhat different.

11.4.2 Parallel Paths, Many Worlds and Search

It is well known in AI research that, while handling the conjunctive case is relatively straightforward, handling the disjunctive case is often an intractable problem for situations of real-world interest. The conjunctive case corresponds to a single thread of reasoning while the disjunctive case (common for planning in which one of a number of possible actions must be chosen at each timestep, the classic example being the game of chess) leads to a tree-like structure, with multiple branches possible at each decision point.

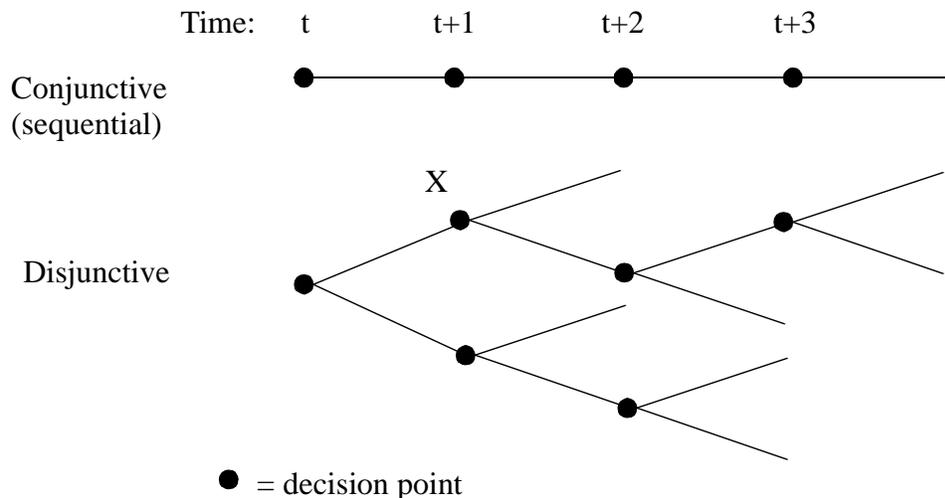


Fig. 11-5 Conjunctive and disjunctive reasoning

Consider the disjunctive case in the figure. If the system is considering a particular state of the world, say X, it knows that there are multiple possible states to which it could transition. The choice of new world might depend upon the choice made by the system (as in chess) or may involve other outside factors (perhaps the actions of the other player in chess or, more generally, the random response of a probabilistic environment).

The problem with this type of problem is that each state represents a different possible world. The same action taken from two different states might produce two entirely different outcomes and so the decision making process must take the state of the node into account for the evaluation of the utility of each action.

An extra complication can arise if the choice of action itself is determined by the state. Now, not only is the actual state information different from one world to the next, but each state could demand a different set of inputs from the control network to generate the next states.

Thirty years ago, using search algorithms was the only real choice for such situations and a significant investment in AI research in the sixties and seventies was into algorithms to search the nodes of the tree, state by state, to find the best solution. The development of heuristic algorithms sometimes led to a speed-up in finding the solution, but the approach was still a sequential process from state to state. The only difference was in the choice of which states to explore and which to ignore (see Russell & Norvig(1995) for a review of a range of search algorithms). We want to avoid the sequential nature of the computation by devising a system capable of evaluating multiple solutions in parallel, but does such an architecture exist?

11.4.3 Parallel Computation Using Extra Resources

The trivial solution is to duplicate resources for each fork. Such an approach has been applied in superscalar microprocessor design, where the decision of whether or not to take a conditional branch (a similar disjunctive problem) must often be made before the branch condition can be evaluated. When resource costs are not a problem, it is feasible to take both forks at once (using parallel hardware), discarding the results of the incorrect branch when the actual result is known (Johnson, 1991). Here, this approach will be ignored since it does not represent an efficiency improvement in resource usage. (Also, it requires an exponentially increasing number of resources in parallel as a function of the depth of search).

11.4.4 Parallel Computation by Superposing Solutions

The alternative solution is to use a fixed quantity of resources but to try to represent and operate on multiple solutions in parallel. If this is possible, it would increase the efficiency of the resources and therefore represents the ideal solution. There are eight scenarios within this solution which are the result of making three independent decisions. The first is the independence of the parallel solutions, the second is the way in which the evolving state and the short-term memory interact and the third is the nature of the trace that is assembled during processing.

We consider all three questions, beginning with the independence of the solutions. For this question we consider two extremes, one in which each solution is truly independent of all others and the opposite extreme in which there is only a single evolving solution with a list of variations.

For independent solutions, we allow the state of the system to evolve at each timestep in such a way that the set of states that are currently represented at time t are replaced with their multiple successor states at time $t+1$. The algorithm is allowed to prune a branch at any time. It might do this because it concludes that the state represents an invalid or costly solution. So the state vector represents all valid solutions at the current timestep and, provided the number of states does not exceed a limit for the network, it can maintain them all in parallel.

Part of the design process for such a system is to ensure that all solutions remain orthogonal (in some sense akin to that of vectors) so that no interference occurs between them and each solution evolves as if it were alone in the network. The onus is placed on the network to correctly interpret the superposition of state vectors and apply the correct control patterns to each component. The individual control vectors would also be subjected to some kind of superposition.

The likely problem with the orthogonal solution is that either the maximum number of vectors that can be handled simultaneously is low or the quantity of resources required would be high. Such resources, if provided, might remain idle for much of the time. Issues in representing and operating on multiple entities in parallel are discussed later (section 11.4.5, page 325).

The opposite end of the spectrum in answering the question of solution independence is to create a single evolving state in which solutions are not represented explicitly. Instead, the differences and (most importantly) the dependencies of possible states are represented explicitly. At each time step, a new operation is applied to the state but instead of generating new states, the single structure is modified to take account of the changes to some of the possible worlds it represents.

For example, many future worlds might depend upon the assumption that a certain belief is true. Learning that the belief was false would prune off all of these possibilities. This does not involve finding and removing explicit possible states in a superposed vector (as was the case before) but merely removing the component of the vector that corresponds to that dependency. A potential benefit of this approach is that by storing solutions as differences and dependencies, it might be possible to use resources more efficiently, especially when the many solutions are quite similar.

The drawback with this second option is that only a single control operation can be applied to the state at any time. Thus we have a kind of sequential processing, but acting on many possible solutions. (In computer architecture terms, this is a Single-Instruction, Multiple-Data, or SIMD architecture). From the point of view of control network design this is a beneficial simplification, however. The differences between the two extremes are subtle and much further work is required to clarify the issues of this method of representation.

The second question concerns the way in which the evolving vector and the short term memory interact. At one extreme, the whole detailed content of each solution could be represented in the single vector of evolving state, accessing a fixed long-term memory and guided by an external controller.

At the other end of the spectrum, however, we could allow parts of the content of the state vector to be written to short-term memory as and when required. Remember that in the architecture presentation of chapter five, the state vector is itself merely a symbol that acts both as a carrier of immediately relevant information and as a pointer to potentially larger structures in the higher levels of memory.

Up until now in the discussion of parallel processing at the rules level, we have considered the state vector as storing the whole symbol structure itself, but this is outside of the original architectural philosophy. How could we bring the two into line? The state vector is meant to point to a symbol structure. That structure could itself hold either the multiple solutions or the single set of differences and dependencies (as decided by our answer to the first question).

As it evolves, new facts and possibilities are added to the structure either based on or related to existing elements. Such elements could be made ‘permanent’ by rapidly registering them in the lowest level of memory (the term ‘level *a* synapses’ was used in the discussion on learning hierarchies). At that point they move from being explicitly represented in the symbol to being merely pointed to by it.

There are a number of points to note here. First, note that each timestep as it has been described, is represented by one recall step. Such a step would allow enough time to fully access the memory of the data network, drawing on the full range of its knowledge base. Therefore, the fact that information has been moved out of the active symbol and into memory (although there will surely be implications on how it can be used) does not render it inaccessible to the on-going computation.

Next, note that the movement of such information must be a reflex action, that is not requiring a decision on the part of some other entity. To be reflexive it must be based on decisions made at the neural level. If this were not so, the benefits of parallelism in the data network would be lost as some other network worked out what should be transferred to memory. In moving to memory, the information must also leave the symbol encoding at the same time. For this to occur seamlessly the symbol encoding will presumably be even further constrained.

For future work, this latter solution seems the most appropriate one since it fits in with the rest of the architecture. Indeed, it calls upon us to address some of the issues left unresolved after the architecture development of chapter five (such as how information is split between direct encoding in the symbol itself and the symbolic structure for which the symbol is a pointer).

The third and final question in the discussion of options for parallel processing of symbol structures concerns the nature of the trace that is assembled during processing. We could devise a system in which multiple solutions could evolve in parallel without a method of tracing the line of reasoning which led from the starting state to the end result. Alternatively, we could include sufficient extra information in the evolving state to allow the sequence of logical steps to be reconstructed at some subsequent time.

To illustrate this, consider the time worn example of the chess computer. Critics of such machines as candidate ‘intelligent entities’ argue that even if a machine can beat a grandmaster at chess, it cannot explain what thought process led it to make a critical move nor describe what alternatives it rejected and why. We know that chess playing machines, such as IBM’s Deep Blue work by searching the tree of possible moves and retaining more than the score of the fitness of each state visited is not practical. Such an algorithm is no use for reconstructing the trace of the reasoning process.

For people, things seem to be different. We do retain some sort of trace. We can reconstruct in our own minds the critical paths that we considered and the thoughts that we had at certain instants of our deliberations. Therefore, some mechanism, either within the same network as the reasoning process itself or in another network, is able to record a trace of events.

A trace such as this is more useful than a means of providing a post-mortem long after the decision making process is over. At any time new information could invalidate old conclusions requiring a re-examination of the possibilities. Trace information allows the system to narrow down its search and avoid retreading the same fruitless ground. Indeed, we would probably be loathed to attribute intelligence to any entity which was forced to re-evaluate a set of states because it had no trace of what it did five minutes ago.

One final note is that since the trace information is not critical to the evolving state it could be easily transferred to short term memory and not clutter the current computation. Therefore, although the use of tracing could be decided independent of the decision on the integration of state evolution with short-term memory usage, it probably makes sense for tracing to only exist if the transfer to memory is also employed. The trace function will not be considered further here, but will be mentioned in the future work section of the last chapter (section 12.3.2, page 342).

11.4.5 Limitations of Distributed Representation for Parallel Processing

In this sub-section, we consider the implications at the neural net level of trying to represent multiple symbol structures in parallel. The vector coding proposed in chapter six is a K-from-N code, a member of the class of distributed representations, a technique that is also called coarse coding. For all of its improved efficiency of bit usage, coarse coding suffers relative to a more localist representation in the number of separate items it can represent ambiguously in parallel on a given set of units.

For example an N-bit vector using 1-from-N coding could, in principle, represent N different things and potentially all of them in parallel. The same N bits used in a single K-from-N code can represent C_K^N things, but only one at a time. An obvious way to try to improve the representational power of such a vector is to divide it into fields and allocate a subset of the bits to each field.

This has two disadvantages, however. First, each field contains only a fraction of the total number of bits, N. (Let’s say f fields of n bits each). So the number

of valid vectors that can be represented is much reduced (a topic that was discussed at length in section 6.3, page 157 and section 6.4.2, page 164).

The second problem is that unless the vector can be shifted and processed one field at a time, the network which processes each field must be duplicated (as was the case in Hinton's *within-level timeshare* network described in section 3.3.3, page 71). This is undesirable since we want to avoid high-level operations like vector shifting. Also, the aim of this chapter is to try to implement parallelism at the symbol processing level, a result that is not achieved using a serial, timeshared machine.

Let us consider the issues in overlapping valid vectors, beginning with the static case as it is simplest to visualise. We could allow several valid K-from-N vectors to co-exist in a single string of bits as shown below. In the figure below, three separate, valid patterns are superimposed creating a vector which is potentially 3K-from-N. (Note that non-zero bits in the combined pattern are set equal to their pattern number merely for sake of comprehension. In reality all firing bits are set to '1'). The question we need to ask is how does the network make sense of the overlapping vectors?

Pattern 1:	0 0 0 1 1 1 0 0 0 0
Pattern 2:	1 0 1 0 0 0 0 0 1 0
Pattern 3:	0 0 0 0 0 1 0 1 0 1
<hr/>	
Combined:	2 0 2 1 1 ? 0 3 2 3

Fig. 11-6 Combining multiple K-from-N vectors

From the analysis provided in chapters six and seven on the simple network it is clear that the sudden increase in firing neurons will cause a problem. For a given K-from-N pattern, any extra firing neurons are a source of noise. In the recurrent network of chapter seven, that noise can destabilise an otherwise stable pattern.

Theoretical analysis and subsequent simulation showed that the basin of attraction of a given pattern is difficult to extend beyond half the number of firing neurons in the pattern unless unacceptably low loading of the memory is employed. Here we expect to merge multiple patterns, creating a noise component at least as large as that of the signal (and greater still for three or more simultaneously active patterns). Without modification, the simple network would immediately fail in such circumstances.

Even if it were possible to modify thresholds to maintain a stable, multi-vector state, the thresholds would need to be modified again if any more vectors were added to the representation. This problem is similar to that faced by Austin in his AURA network, where cues to the memory could have a variable arity (section 4.4.1, page 108). For AURA, the problem was solved by using multiple parallel

memories each looking at a different arity. This is unacceptable here, as we are looking for a way to represent multiple things in the same network.

More fundamentally, a key driver in selecting a fixed K-from-N encoding was that for low K/N the number of vectors that could be stored was high relative to a K/N of ~ 0.5 . A fixed number of firing neurons is also much more easily enforced at the network level than a value that must change as the number of parallel solutions builds up. To vary this value presumably requires external signals derived from some other agent: there is a danger of summoning up the homonculus that we are trying hard to banish.

In summary, we see that the memory storage and recall properties as well as the mechanisms that keeps the network in the normal working regime are critically dependent on the number of firing neurons. If we are to permit multiple parallel solutions, the mechanism to do so should not impact the lowest level of functionality of the network. What alternative exists?

If a single valid K-from-N vector is all that we are permitted to represent simultaneously, and such a vector can represent only one symbol (and thus one constituent structure) then one possible answer is to hold multiple solutions in multiple branches at the top of the syntactic tree which the symbol represents. So the first branch point delineates the multiple solutions (with branch labels to distinguish between the different choices that were made for each solution) and the tree beneath each solution constitute the data for that solution.

To pursue this interpretation, we need to fill in the details of how the network could process these multiple solutions in parallel. (It is assumed that serially processing the single tree is trivial, but does not provide the parallel symbolic processing that we are searching for). This mechanism would be the same as that required for a single symbolic tree, since in both cases the tree is of arbitrary complexity. It would be expected that in both the parallel and singular cases, larger trees will lead to less exact solutions since the network has only finite resources (a issue we turn to next).

Remember that one of the key properties of a symbol is to bring to the fore important parts of the underlying structure, which implies that other parts are less accessible. As the number of solutions builds up, the attention to the details of each will be necessarily reduced. It will then be up to the control network to decide to break up the solutions and consider them in sub-groups or singly, as required. It is expected that to guide the control network in making this decision, the subtraction mechanism described earlier will play an important part.

So in summary, it is proposed that multiple solutions might be handled by creating a single super-structure containing many possible solutions in parallel. The normal mechanisms that must be created to handle any symbolic structure would process this super-structure. It remains to define the normal mechanisms for handling symbolic trees, a task that is not complete today. Later, we consider a sub-symbolic mechanism for generating possible symbol structures, more akin to intuitive than strictly rule-based processing (see section 11.4.7, page 329).

11.4.6 On the Importance of Managing Finite Resources

In this sub-section, we argue that a central problem in network design should be the management of computing “resources”, being the networks ability to store and manipulate complex data structures. The reason for this is simple. For the kinds of system we are describing, operating in the real world, not only is the total quantity of resources for the system likely to be fixed at its inception but it is also likely that the system will be capable of actively representing only a fraction of the total number of relevant pieces of information at a time for a given problem.

A common criticism of much of the existing work presented and discussed throughout this thesis has been the failure either to manage these finite resources or, in most cases, even to acknowledge that the problem exists.

For example, the marker passing networks of Lange & Dyer, Sun and Shastri assumed fully parallel instantiations of their units of computation (be they frames, schemas, predicates, or whatever). Shastri was keen to stress that the inference time of his network was roughly proportional to the length of the longest single chain of inference in the knowledge base, but he downplayed the fact that the parallel resources that are assumed in this result scaled with the size of the knowledge base and were thus without an upper bound. Similarly, the schema theory of Arbib, although appealing as a flexible way of modelling multiple possible interpretations or actions in parallel would also be crippled by its equal liberality with resources.

For practical systems, the management of resources must be a prime focus of attention. Whenever the reasoning process encounters a set of parallel, but mutually exclusive, options and insufficient information exists to make a single choice we would like to keep all of our options open and wait for further data. But as the number of branch points increases and the number of possible solutions grows exponentially there must come a point at which we are forced to make tentative decisions to close off some of the open paths. Couched in the terminology of the schema theory world, this would correspond to choosing one from the many competing interpretations of a set of facts and assuming in further processing that the one selected schema is correct. All other competing schemas are removed from the active list and are either erased or moved to a lower cost (inactive) store such as hard disc (for a computer simulation) or long-term store (for a brain-like network).

Subsequent processing should allow any taken decision to be checked. Future information may reveal that the wrong choice was made. This scenario is common in our own reasoning process as we explore various options. Our only recourse in such situations is to backtrack, removing the erroneous conclusions and re-instantiating the multiple schema options that existed at the point where we made the wrong choice. A new choice can then be made and reasoning can proceed from there. In all of this deliberation the system should try to keep as much information around as possible to facilitate good decision making, but the key point being made here is that the system must be prepared to commit to a decision when required to avoid running out of resources. It also needs to remember what decision it made and where, both for explanatory purposes and to facilitate back-tracking in case of errors.

In passing, a note is probably in order to describe the kinds of resources that the system we are describing will have. It is not envisaged that there will be whole functional units for the system to allocate and de-allocate, like blocks of computer memory. Instead, it is probably more accurate to think of resources like a single slab of memory and the total quantity of resources is measured in terms of the separation of patterns in that memory. When the memory is almost empty the vectorial distance between patterns stored there is large. As the memory fills up there is increasing cross-talk between parallel patterns. The noise margins are eroded until finally the point is reached where the interference between memories generates too many errors and inhibits performance.

11.4.7 Exploring Possibilities Without Guidance

The restrictions on representation using distributed coding have been shown to limit the number of items that can be simultaneously and unambiguously represented in parallel in a single N-bit vector using K-from-N coding. It was concluded that trying to represent multiple solutions was best done at the symbolic level rather than by modification of the fundamental coding strategy of the network (discussed in section 11.4.5, page 325). It was assumed that for such processing, specific, known transforms are applied to the growing set of solutions, in a manner akin to exploration of a state space.

An alternative way of processing information in parallel, more akin to intuitive than rule-based thinking, is to remove the driving control network altogether and allow the data network to build-up solutions based on a sub-symbolic exploration of its representation space.

This method draws on three main areas of previous work. The first is the discussion of the use of finite resources just presented. The second is the exploration of the issues in intuitive reasoning from earlier in the chapter. The third is the whole concept of learning hierarchies, as developed in the previous chapter.

To proceed we ask: what is the processing we want to achieve? If a group of N neurons can represent a single K-from-N vector, but that vector can itself represent a whole symbol structure then the act of processing, being to generate new knowledge is to create a new K-from-N code that defines a new relationship between existing stored entities, encoded as a symbol structure.

If the new symbol structure is arrived at by logical means and driven by a control network, we look upon the style of processing as rule-based. But what if we don't know how to proceed in generating the new structure? What if no algorithm exists and all that we have are a large number of facts and a goal? One answer is to explore possibilities by making free associations and seeing if the resulting structure solves the problem. To do this, we think of the network as a hierarchy in which all of the knowledge and constraints have been coded. We could apply any control patterns to the network as inputs, if we wish, and generate output patterns corresponding to new derived facts, logically arrived at. What we can't do is activate all of the stored facts simultaneously (in an attempt to generate all logical conclusions derived from all of the stored facts) because this would merely saturate the repre-

sentational power of the layers. So the use of free association is to allow individual facts (or sub-facts as shall be described next) to become active and associated with others in a random manner in the hope of serendipitously stumbling upon new and interesting results.

The Boltzmann machine (Hinton & Sejnowski, 1986) and Harmony theory (Smolensky, 1986) are two examples of networks that use noise and a simulated annealing algorithm to search the space of possible solutions subject to a set of constraints. In this proposal, we apply a similar approach of using noise to illicit activity in the network and forging new, but temporary connections. Since the individual facts or associations are stored as distributed representations, the random activation of a set of units corresponds to sub-symbols. The benefit of noise is that it allows random combinations of sub-symbolic to become active in parallel. If they are well matched (in some sense) a temporary link is made, indicating some sort of feature has been detected.

Three important questions are in order. First, how does using noise in this way represent an efficient method of processing? Second, how do we ensure that the addition of noise produces valid and useful results? Third, how does the hierarchy map onto the flat group of N neurons in the standard building block put forward in this thesis? The first question is an interesting one, since one of the major criticisms of earlier work on simulated annealing was that it is highly compute intensive and convergence to a good solution is not guaranteed. Perhaps one problem with the Boltzmann machine was that the search space was rather flat, meaning that one units activation was much like another. To try to avoid this, the approach taken here is a more hierarchical one.

Imagine the random activation of units at the bottom of the hierarchy. Each such unit detects the occurrence of a feature stimulate by noise. But it means that for the time the unit is active, the network is ‘thinking’ about that feature, feeding information about it to the neurons at the level above. In an instant, the unit deactivates and other units fire to take its place. Now imagine a unit on a higher level. If it gets to fire it is due to the random occurrence of active features on lower levels that it is already tuned to recognise. By firing, it is focusing the networks attention on a higher level feature, one that occurs less frequently than the occurrence of any of the lower level features that comprise it. Because of this, we force the higher level feature to remain active for longer, signifying that some thing important has been detected.

We can continue up the hierarchy in similar fashion, with each level detecting more and more complex (and hence rarer and rarer) features and hence being active for longer and longer once activated. (To compensate for longer activation, we suppose that the higher levels must be slower to activate also or else saturation at the higher levels would soon occur). By temporarily strengthening the connections between firing neurons in the hierarchy, we make it more likely that the noise will re-establish the firing of a higher level neuron than the previous time. The idea is to try to balance two competing requirements, trying to explore as many states as possible but trying to focus resources (in this case the resource in question is time) on combinations that might lead to useful results.

Links made in free association should be made to fade quickly to prevent saturation of the network and to prevent it becoming entrenched in a small fraction of the search space. We should allow for the possibility that links at different levels fade at different rates and that in spite of the fading, entrenchment might occur from time to time (to make the human analogy, the standard solution would be to take a break and ‘clear the mind’, allowing time for links to fade before returning to the process).

Combinations that represent valid structures could trigger a response from the control network and should lead to registration in memory in a form that is more durable than the temporary links made during the course of free-association. This requires an explicit decision to store something in memory. How this should be done without recourse to the homonculus is not yet clear.

We return to the second question that was posed earlier: how do we ensure that the addition of noise produces valid and useful results? The simple answer is that we cannot guarantee it. The use of random associations between sub-symbolic vector components means that the state of the data network need not correspond to a valid fact, nor even a valid symbolic structure, at any given time. The control network must not ‘crash’ (in the modern computer science meaning of the term) as a result of trying to interpret a nonsense vector from the data vector, a point that was made in the description of the subtraction network in section 11.1.3, page 298. Requiring that a valid symbol carries within its encoding a tag (such as the chapter five example of ‘Paris_city’) is one means to control this problem, while the demand that symbols be hierarchically organised is another. (Such an organisation is to try to ensure that even if the symbol is not exactly correct, some valid information remains within to aid the control network in tracking down any problems. For this to work, the higher levels of the symbol must form first and lower levels must depend upon those above^v).

Finally, we consider the third question posed earlier, how does the hierarchy of neurons described for this style of processing map onto the flat group of N neurons in the standard building block put forward in this thesis? The answer to this question has already been considered in this chapter, in the early section on the ideal and practical module (section 11.2.4, page 307). By trying to remain general purpose and yet represent an arbitrary hierarchy, the network uses fully interconnected neurons but relies on the multiple parallel synaptic units of the learning hierarchy to act as if it were a hierarchically organised network. Thus the proposal put forward for a hierarchy as the framework for free-association should be amenable to mapping directly onto the N neuron networks already proposed. How to do this is left for future work.

v. In this context, the level of organisation of the symbol are no the same as the levels in learning hierarchies. They refer to the hierarchy of category membership for an individual symbol, as explained in section 5.8, page 138.

11.4.8 Conclusions and Discussion

This section has tried to outline some of the issues and possible options for taking better advantage of the parallelism in neural networks and applying it to more rule-based reasoning. No matter how desirable, it is not expected that the network presented in this thesis will supply a solution that will allow perfect superposition of network states each representing an independent symbol structure. However, a number of other possibilities have been put forward that, although vaguely defined at this time, probably represent better avenues for research for the medium term. These other possibilities include the evolution of a single vector representing multiple solutions in terms of their dependencies and differences, as well as the proposed interaction between state and short term memory as a means of off-loading non-critical data and hence improving the efficiency of the state vector.

It is difficult to escape the feeling that somewhere beneath the surface that we can understand and explain today, there exists a unification between the mechanism of intuitive reasoning, the development of multiple parallel solutions to a problem and the efficient use of computing resources. It is hoped that a better understanding of more complex memory systems will lead to a more tangible model of the feedback between the network state and the lowest levels of memory, as defined in the chapter of learning hierarchies. This, in turn, should provide a platform to investigate, clarify and quantify the options presented in this section on parallel rules processing.

11.5 Towards a Unified Neuro-Symbolic Theory

This chapter has attempted to draw in the threads of many different colours with the intent of weaving a unified tapestry which could form the basis of a general purpose intelligent system. In doing so, it has drawn from the work carried out for this thesis as well as other pieces of relevant work from the field.

While the definition of such a system is still beyond the reach of the today's research community, it is clear that in the last decade the focus on hybrid neuro-symbolic architectures has brought the possibility of achieving that goal much closer to hand.

This chapter has explored many of the issues that must be resolved to realise such a system and, it is hoped, has concentrated on the most important among these. The development of ideas presented in this thesis, although made up of two parallel branches (symbol architecture and neural building block) are intended to have an underlying unity, albeit one which is not formally defined at this stage of the work. This unity will ultimately flow from the fact that a single set of goals are being addressed and work in both areas is required to achieve those goals. At this point the list of topics left to explore is long and it seems that every step forward uncovers several new paths to explore. The expectation is that as we cover more and more ground, the paths will start to converge more than they diverge.

To realise a practical neuro-symbolic system, there appear to be many major problems to overcome. At the neural level the capacity and reliability of currently proposed associative memory is simply not good enough. There is a tendency for researchers to focus only on storing a set of unrelated patterns while a more powerful (and potentially more reliable) approach is to look at patterns as part of a hierarchy and to exploit this idea by representing new patterns in terms of existing ones. Merely scaling today's associative memory architectures to create larger memories is not the answer. This work has tried to find more powerful alternative memory architectures and has proposed a number of schemes (centred around the learning hierarchies principle) which might bare fruit with more investment of research resources.

Turning now to the symbolic level, there is a gap which few seem to be trying to fill at this time. This gap is a definition of more flexible reasoning architectures and a means of integrating the results of each processing step with the choice of the next step and with modifications of the structure of the knowledge base itself. Existing work in this area has been criticised as avoiding the real issues. This chapter has elaborated on what some of these issues might be and to suggest other areas of reasoning that have been neglected but could be integrated into the architecture with a little effort.

At an intermediate level, there is no good solution on the table to bring together the neural and symbol-structure levels. In this thesis, we have again been critical of existing work (largely based around back-propagation), but in spite of the advocacy of the learning hierarchies approach as a better basis for extracting symbol structure, no alternative methodology has been put forward in a concrete form to date. This area of research might be a fruitful one to pursue in the short-term.

The final chapter presents both a recap of the work presented in the thesis and a list of more areas to explore that have received no coverage at all in this work.

Conclusions & Future Work

12.0 Introduction

The current wave of interest in the development of hybrid symbolic/neural architectures is, in large part, fuelled by the belief that a system which embodies both disciplines can overcome the disadvantages of both in the realm of intelligent system design. This work has outlined an architecture which could meet the requirements for such a system and has proposed a symbol encoding scheme and developed a neural building block which is tuned to the representation of symbols using that encoding.

This chapter will review and summarise the new ideas and development work presented in this thesis, before outlining possible future areas of research which are intended to address the shortcomings of the proposed solution and to extend the scope of the work to consider more complex applications and structures.

12.1 Comparison of Requirements and Results

This section will review the results of the work. Specifically, what positive results were obtained, in what way the implementation fails to implement the proposed architecture and to what extent the architecture-implementation pair fulfils the goals first laid out in chapter one.

12.1.1 Results Obtained

A review of many key developments in the fields of Artificial Intelligence and Neural Network theory has been presented and the major differences between them (both real and apparent) have been discussed. The rationale behind a hybrid approach was described along with work in that direction. This review led to the conclusion that in the design of large scale intelligent systems both AI and ANN theories have vital elements to contribute. The rest of the work was devoted to an investigation of the necessary properties of both the symbolic and the neural levels of such a system.

An architecture has been outlined which addresses many of the requirements for a robust symbol system imbued with sufficient flexibility to handle a complex and evolving database of relationships between objects. This architecture is aimed at the implementation of the knowledge base of an intelligent agent which must develop and optimise its own representations, including developing categories and generalising. A set of constraints was enumerated which a symbol encoding must meet in order for the system to exhibit the required properties and a symbol encoding was described which would meet those constraints.

A neural network has been proposed and developed to fill the role of the building block for the architecture. The initial network was a combination of a Hopfield network and the non-holographic associative memory using a K-from-N encoding scheme which was itself consistent with the symbol encoding properties of the architecture. This network displays a number of useful properties including one-shot learning and reliable recall up to a certain limit which was shown both by analysis and subsequent simulation to give better performance than the basic Hopfield network.

Further network development added a time-sequential element to the pattern encodings, leading to patterns of fixed period L . Networks storing these patterns were shown to have a higher storage capacity than in the simple static case, although the signal to noise ratio was reduced and each pattern was now expressed over L cycles, increasing the bandwidth necessary to transmit it.

The addition of a control structure, which facilitated the association of two patterns using a control pattern to perform the mapping, showed that the simple learning algorithm developed so far was insufficient to handle a large number of patterns and associations.

Next, the added complexity of learning hierarchies was introduced in which each synapse possesses significant internal structure, facilitating the development of feature detectors in the neural population. This scheme led to a trade-off between the storage capacity of the network and the degree of fidelity of recall of the pattern. Over time, pattern encodings would experience 'drift' but it was proposed that the creation of correlations between pattern bits could be used as a vehicle for inheritance and generalisation. This remains to be proved, however.

Finally, several key issues in the implementation of a general purpose neuro-symbolic machine were presented and discussed, both in the context of the literature and of the ideas developed in this thesis. It was shown how key problems such as sub-symbolic reasoning should be well suited to implementation in the learning hierarchies network.

In addition, some key areas that are not well represented in the literature, such as the handling of meta-knowledge, were discussed. The requirement for and putative properties of a subtraction network were put forward to handle some of the functions that are currently handled outside of the representational space of the most existing reasoning networks (if they are handled at all).

Several approaches to handling parallel processing at the rule level were put forward. It was argued that the K-from-N coding would be difficult, if not impossible, to modify in order to represent multiple developing solutions in parallel. Alternative choices put forward for this level of parallelisation were to represent multiple solutions as parallel branches in a syntactic tree, or to attempt an intuitive (non-rule) based scheme using noise to drive activation of neurons and the subsequent exploration of the solution space.

12.1.2 Implementation of the Architecture

A network building block has been proposed which is intended to implement the architecture. It has displayed many of the properties that were deemed necessary for such a unit in order for the system to develop and learn in an unknown environment. Such properties include one-shot learning and the consolidation of any given pattern or association of patterns.

Key to the neural building block development is the K-from-N code, which has been extended into a D-from-K-from-N code with period L, increasing the maximum number of stored patterns in the network at a cost of lower signal to noise ratio. This architecture gives the designer extra degrees of freedom in network design.

The coding scheme was shown to grant powers of stability, noise immunity and controllability to the network, each property being crucial in a large system which might consist of very many such networks, heavily interconnected and operating in a noisy environment.

However, at this time, several architectural features have not yet been implemented and the computational performance of the network has only been partially assessed. As a result, some of the assumptions made in the architecture development are not yet justified. For example, the control network which guides the data network from state to state is not fully specified. This is a necessary step in 'closing the loop' to produce an autonomous system. In addition, certain features of the network operation, such as the mechanism which decides which network (data or control) is updating at any given time, are not fully described.

At each step of network development, the memory capacity and dynamic behaviour of the model were analysed. Additionally, the performance of the network was assessed by simulation and compared to earlier steps in the development. Simulations backed up the analysis to a reasonably high degree and showed that the memory capacity was better than that of an equivalent Hopfield network and that the convergence of the network in the case of noisy input was more predictable, based on a basin of attraction around each stored memory.

12.1.3 Fulfilling the Goals

The goals of the work are recapitulated below, with comments on the extent to which each has been addressed.

- *To Provide a Robust Substrate for Symbolic Computation.* The proposed architecture should provide such a substrate. The symbol encoding developed in chapter five is the basis for robust and reliable computation and data storage, when used in conjunction with the concepts of knowledge hierarchies and learning hierarchies that were also developed as part of the architecture. Simulations of the developing network show that many of the required properties have been achieved, although it is clear that much work remains.

- *To Provide a Means of Storing Large Amounts of Data Efficiently.* It was shown that the network storage capacity was higher than that of conventional network architectures. This was due to a combination of the sparse coding approach and the use of dynamic patterns expressed over several cycles. Storage efficiency has not yet reached adequate levels for the proposed architecture, however. The recasting of patterns using learning hierarchies should be the vehicle to improve the efficiency to the needed levels.

- *The Efficiency of the System Should Increase as the Network Size Increases.* The capacity of the network was increased as the number of neurons increased, as one would expect. For the simple and dynamic networks the efficiency of storage was significantly improved with increasing network size. The use of feature extraction, through the learning hierarchies approach, is intended to address this goal but while the approach seems valid in theory, it has not yet been shown to be true by simulation.

- *To Make Data Available for Computation as Appropriate.* The symbol approach outlined in the architecture fulfils this goal, since the symbol output of the network acts as the primer for the next recall. For example, we see that for the association network of chapter nine, each control pattern acted to restrict the set of possible output vectors to a fraction of the whole pattern set, suppressing the influence of unrelated output vectors. In addition, the use of a subject pattern, as outlined in the exposition of learning hierarchies, will break up the monolithic memory into sub-memories, so that patterns from one such memory do not interfere with the recall of patterns in another.

- *To Provide a Learning Mechanism which Facilitates the Efficient Acquisition of New Information.* The concept of learning hierarchies, presented and developed in this thesis, is centred around the ability to store new data in terms of previously learned relationships. The rationale behind this development is to improve the learning efficiency of the network and, hence, the storage capacity. At the same time, learning hierarchies is a mechanism which permits the seamless integration of memories which are intended for short-, medium- and long- term storage and thus has the potential to be a powerful design approach in addressing the stability-plasticity dilemma which has been a problem in adaptive filter and neural network design since their inception.

- *The System Should Be Realisable*. In the development of the neural building block, implementability was a key concern; the output of each neuron is binary and all learning algorithms use only information which is local to the neuron undergoing synaptic modification. Each such decision was taken to simplify the task of implementing the system. A network of a few hundred neurons using the proposed architecture would be realisable even with the silicon technology available today.

12.2 Future Work

The possibilities for future work are considerable. They can be divided grossly into four sections. First, into elements which make up the complete implementation of the architecture proposed; second, into investigations of many of the options where several possibilities were proposed but only one was chosen for further study at that time; third, into extensions of the network implementation and fourth, into future enhancements to the architecture and their subsequent realisation. This section considers some of these possibilities.

12.2.1 Completing the Architecture & Implementation

As outlined in the last section, there is a number of tasks remaining to complete the basic architecture and implementation. In the architectural exposition, the mechanism of generalisation via inheritance was outlined using a hierarchical pointer structure. This structure must be formalised in order to proceed to the second step, implementing the full control network and 'closing the loop'.

It was felt that to adequately formalise the architecture and symbol encoding would require detailed knowledge of the network implementation, an assumption that lies at the heart of the connectionist views of neural networks as more than a mere medium of implementation for practical symbol systems. Thus, the work done on the implementation and its implications for the symbol encoding should prove invaluable in this regard, acting as background for the formalisation process.

It was noted in chapter five that while the concept of learning hierarchies (which was implemented in chapter ten with accompanying discussion) appears to furnish properties similar to those required by symbol encoding in the inheritance model, it does not follow that any scheme of learning hierarchies will necessarily represent an environment for valid inheritance. Further work is required, firstly to justify that there are sufficient conditions in which learning hierarchies can robustly implement inheritance and secondly, to identify the necessary structural and learning constraints for this to occur.

Chapter eleven discussed many of the remaining issues that must be overcome to construct a true neuro-symbolic intelligent system. The most significant among these are the symbol-structure subtraction mechanism, the handling of multiple evolving solutions and the realisation of non-rule-based, intuitive processing. It is asserted that the neural substrate and architecture proposed in this thesis form a useful framework within which to address these key topics.

12.2.2 Investigating Other Options

In chapter eight, the development of the dynamic pattern network described a number of options for the calculation of neuron potential. The chosen option was the digital signal processing (DSP) approach where the firing of a neuron affects others for a fixed number of cycles after it has fired. This simplifies analysis but places a burden on the implementation in the storage of multiple rounds of firing neurons. A more natural approach is to model the neural potential as a differential equation so that the change in the potential at each timestep is made up of an increment from firing neurons and a leakage term. The more complex analysis involved may be compensated for both by easing the implementation and, perhaps, by an increase in storage capacity.

Next, in chapter ten, the development of learning hierarchies proposed two simple means of combining the output of the parallel synaptic units. The linear combiner was selected for analysis, again based on the simplification of the analysis. The *max* function was also presented but not analysed. There may be other options which possess different properties.

Also in chapter ten, the learning algorithm used in consolidating learned patterns allowed numerous combinations of parameters to be considered. Only a few options were given in depth treatment, while many potentially useful combinations were ignored. One outstanding example of this occurred in the choice of the *a* unit synapses as those with longest duration and lowest weight vector length (and consequently the *g* unit synapses as shortest duration but longest weight vector length). The alternative choice was to reverse these trends, which might lead to better performance of the learning hierarchies approach.

One other interesting area of investigation would be to quantify the consolidation process. The process itself involves the trade-off between well defined potentials for firing neurons after multiple rounds of consolidation on the one hand and disruption to the slow moving unit *g* synapses on the other.

12.2.3 Unconsidered Implementation Possibilities

A major area of implementation which was not considered in depth is the breakdown of the fully inter-connected neural region into overlapping sub-regions. Such a notion is inspired by our knowledge of the structure of the mammalian neo-cortex, in which the axons of individual neurons have a limited span, making connections with near neighbours with very high probability but having little or no direct intra-cortical connectivity with neurons beyond a certain distance. Consider the figure overleaf, which shows a data network divided into a number of regions.

Each hexagon in the figure represents a region such as has appeared throughout this work. Each region consists of *N* neurons and is constrained to reproduce patterns using the D-from-K-from-N rule. Now, we expand the sources of two types of input, the external and the internal, inter-neural inputs.

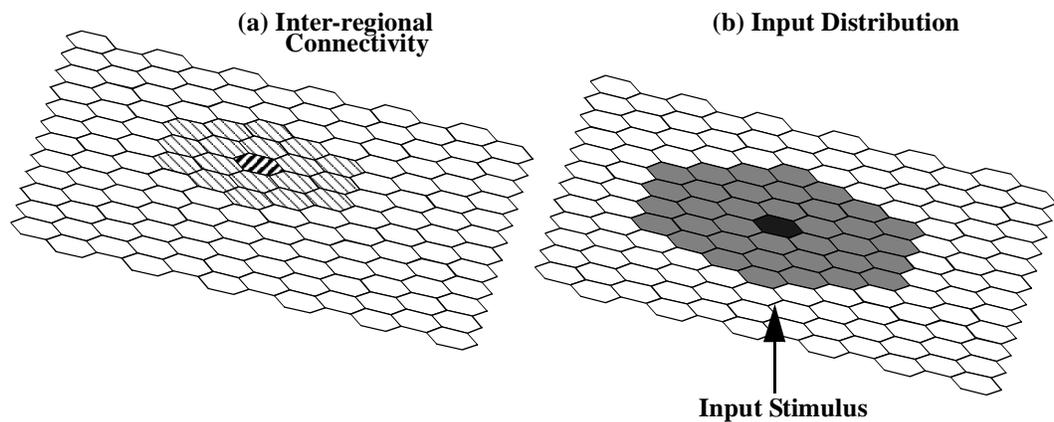


Fig. 12-0 Dividing the network into multiple regions.

In the inter-region case, each neuron in addition to input from neurons in its own region, also receives input from neurons in regions a distance r away (in part (a) of the figure, $r = 2$). Each such region uses D-from-K-from-N coding, so that for $r=2$, the total internal input to a neuron comes from 19 regions or 19D firing neurons.

For the external input, we define a radius, x , over which an input signal is distributed. In the figure $x = 3$, so that an external input arriving at the dark shaded region is connected directly to a total of 37 regions. Each neuron receives activity from 37D firing neurons at any time. This clearly increases the level of redundancy in the representation, since regions less than a distance x from each other will receive overlapping inputs.

The objectives of this modularity are threefold. First, to increase the redundancy in the system by distributing input across multiple regions. No two regions will receive exactly the same input vector: neighbouring regions will receive almost identical input but the overlap will decrease with distance. The increase in redundancy is intended to improve the accuracy of recall and to permit graceful degradation in performance in the face of damage to some neurons or regions.

The second objective is to increase storage capacity by reducing the inter-connectivity between the firing subsets while still having a large network. Reducing the overall connectivity between firing sets has been shown to increase the total number of patterns that can be stored in the network before saturation at a cost of lower signal to noise ratio for any given pattern.

The shift from static to dynamic pattern encoding was one step to reducing the number of firing neurons which connect with each other while learning a single pattern. Modularising the network so that the firing subsets do not have to be fully connected is based on the same idea. As was shown for the dynamic case, we would expect the reduction in connectivity to also lead to a reduction in signal for

any given firing neuron which would presumably lead to an increased sensitivity to noise. Analysis is needed to quantify the trade-offs involved.

The third reason for modularisation is to render the system more implementable. By modularising, we are breaking the system down into many identical sub-units whose connectivity is restricted, spatially. For very large networks (consisting of many hundreds or even thousands of neurons) such an approach should certainly lead to a more practical implementation in silicon.

In the future, alternative media of implementation may become available. Even then, it is unlikely that the premise that modularity will lead to a more practical realisation will not be equally valid. Certainly the mammalian brain itself, implemented in a biochemical substrate, appears to be bound by similar constraints.

12.3 Future Architectural Expansion

The presentation of further work in the architecture is a section unto itself since it consists of two sub-topics. The first is in the use of inductive learning to extend the knowledge base to cover relationships not explicitly given to the network by extrapolating from known data. The second is related to the issues of induction and concerns an extension of the memory system to aid in restructuring after inductive learning. Here, we add a parallel memory dedicated to recording the events as perceived by the system. The goal with such a scheme is to allow the system to restructure its knowledge tree based on past trials and has application in inductive learning and case-based reasoning.

12.3.1 Inductive Learning

While deductive inference has been intensely studied as a tool in AI for the whole of its existence (Russell & Norvig, 1995), inductive inference has been largely neglected and it is only in recent years that it has come in for serious scrutiny (see Wolpert, 1995 for examples). It is not difficult to understand why: deductive inference is inherently safe, taking true propositions as input always results in true logical inferences as output. Inductive inference, on the other hand, is tentative, non-truth preserving and usually demands a more complex encoding scheme to represent the degrees of belief that the system possesses with respect to given propositions.

As was discussed in chapter two, in the comparison of symbolic and neural architectures the symbolic approach has often been criticised for its inflexibility in the face of novel stimuli: current symbolic architectures are not efficient at generalisation. Neural networks have an inbuilt generalisation capability, since novel stimuli can be mapped to a novel output based on interpolation between known examples, which is a simple form of generalisation. However, the ANN paradigm is not yet well enough advanced to offer a complete solution to the induction problem since current neural networks are not well suited to representing structured data and it is in this area that induction is both desirable and currently unobtained.

In this chapter, the idea of inheritance as a shared, structured pointer was proposed. It was envisioned that learning events could occur both at the level of specific instances of a concept and at the higher, category level and that the consequences of that learning could be propagated in both directions. Downwards, to permit instances of a category to inherit the relationship learned at the level above, as a process of deductive inference. Upwards, to permit the higher level category to share the newly learned mapping, this time as a more tentative process of inductive inference.

There are many issues associated with such a scheme. The rate at which associations at one level are absorbed into the level above needs careful consideration. Too fast and the system will be inclined to over-generalise, drawing conclusions based on joint membership of a category when in fact such conclusions are unjustified. Conversely, too slow a rate leads to a system which is over cautious in its application of knowledge obtained in one situation to a related one. Ideally, the 'rate of induction' should be constant for the system, but should be related to the content of the knowledge base itself.

How the generalisation of one relationship to a whole category affects the membership of other members of the category is also a topic for future exploration. Consider a generalisation of a property at level n is made at level $n+1$. Thus all members of that category at level $n+1$ now inherit the property. If this is a valid generalisation then no problems occur. But if it is not, how does the system detect the problem and subsequently recover from it? Detection would involve either introspection (however that might be achieved) or a counter example arriving from the external world. In either case, the system is faced with an example mapping which does not match its expected result. It could divide the category in two, creating two separate categories which share all previous features and mappings except the one which caused the problem.

But how does the system assign the existing members to one of the two new categories? The safest scheme is to allocate the new counter-example to one category and leave all other members in the other. But there are two problems with this approach. First, this choice of partition may well be an oversimplification of an underlying schism in the category membership, which will only be revealed slowly over time as more and more counter-examples appear and are individually transferred from one category to the other. Second, it is possible that the real problem was the over-generalisation of the property in the first place. In this case, mechanisms should exist to withdraw the generalised property from the category at level $n+1$ and leave it as an association specific to certain members at level n .

These are all issues in the extension of the basic architecture to inductive learning and should provide a source of much future work.

12.3.2 Episodic & Semantic Memories

A century of psychological testing has convinced researchers that the mammalian brain does not contain a single memory store but many, each responsible for the storage and retrieval of a distinct type of information (see chapter two for references).

Leaving aside the distinction between the storage of memories of, say, audio or visual information, one division of memory type is particularly relevant to this work: the distinction between episodic memory and semantic memory. The functional differences between these types of memory were highlighted by Tulving, and his work was, in turn, referenced by Klein (1991). In the context of a learning symbol system, these differences are important, for reasons which will soon become clear. First, what are these two types of memory?

Semantic memory is used to store universal relationships (objective knowledge about the world), whereas episodic memory is more a chronological record of events from a personal perspective. Thus, when one learns at school that the capital of France is Paris, there are initially two learning processes going on in parallel: one is to semantic memory where it is recorded that the capital of France is Paris. The second learning process is a personal record of the actual event of learning this fact: one might remember that the learning took place in the dark classroom at the end of the hall, that it was a Friday, etc. There might be connections to other events which occurred before, during or after, such as the fire alarm sounding in the middle of the lesson.

The relevance of these differences to a memory system stems from the way in which stored knowledge in the semantic memory is developed and extrapolated from individual learning events. Ones semantic memory develops with the acquisition of new knowledge; simple categorisations become richer and more complex as ones knowledge of a subject deepens. This process must result in changes in the relationships between items and categories. In contrast, the individual events which led to the acquisition of each new relationship are themselves unchanging.

Consider a system which makes a decision at time t based on information in its semantic memory. Its action results in new information which causes it to update semantic memory. A record of the event held in episodic memory makes it possible to reconstruct the state of semantic memory as it was before the learning event took place which is useful for introspection and reasoning.

Consider an example. An agent believes that the capital of France is Berlin, perhaps due to a misunderstanding or misinformation at an earlier time. Upon receiving information that the correct answer is Paris, how does the system respond? Should it reject the new information because it conflicts with the old? Should newer information always be considered more reliable than old? By retaining a record of the event of learning that the answer was Berlin, the agent can re-examine the evidence which led to the changes in semantic memory, perhaps with the benefit of new processes and insight not possessed in that earlier time, to help it decide which answer is the more reliable.

Another good example is the level of generalisation. Given an assignment of object x to category y , the agent might assume that all properties of objects in y apply to x . Every experience it has had might confirm this assumption until one event occurs which appears to violate the rule. Access to knowledge about the previous trials might permit the agent to discern which aspects of the new situation are different from the previous ones and hence to split category y into y_1 and y_2 which are distinguished by the differing contexts.

The episodic memory essentially keeps track of the results of trials and the context in which they occurred (even context which appears to be irrelevant may be retained, since it might become significant later on during category refinement).

How all of this might be achieved in the context of this architecture is left for future work since the ideas presented here are far too vague at this point to be incorporated directly. However, it is envisaged that such a memory would be a step towards applying the architecture to case-based reasoning or in a more general sense to an architecture which is able to apply meta-reasoning: to re-examine the assumptions it is making about its own knowledge base and make adjustments. As such, it is worthy of further consideration.

12.4 Conclusions

This thesis has presented the results of many years of study into the issues involved in applying neural networks to a symbol-based computing system. A neurally-implemented symbol architecture has been outlined which is intended to address many of the short-comings of, on the one hand, the Classical symbolic architecture as it has been traditionally implemented on conventional computers and, on the other, the simple neural network models which are often overly concerned with the meticulous extraction of optimal features while ignoring the (potentially simpler but sufficient) challenge of quickly learning patterns and the relationships between them.

A number of well known ideas from past neural network research have been revived, adapted and synthesised into a network structure which has been demonstrated as offering superior levels of performance in some categories than other auto-associative networks as well as improved levels of control in the trade-off between key network properties.

While many issues remain to be resolved, it is the belief of this author that the fundamental direction in which this work is heading is the right one in the search for a robust and flexible architecture for the design of intelligent systems. This belief is supported by two notions. First, by many (intuitively necessary) properties of symbol systems which are not fully supported by traditional symbol architectures but which are facilitated by a foundation in a neural substrate. Second, by the generality, flexibility and representational power inherent in symbol systems which current neural networks lack.

The biggest remaining puzzle is, perhaps, why it has taken until the nineties for those active in the field of AI and neural networks to realise the potential that has been there since the beginning.

List of References

A

- Adamson, M.J. & Damper, R.I., 1999, B-RAAM: A Connectionist Model which Develops Holistic Internal Representations of Symbolic Structures, *Con. Sci.*, Vol. 11, No. 1, pp 41-71 81
- Ajjanagadde, V., 1994, Unclear Distinctions lead to Unnecessary shortcomings: Examining the rule vs fact, role vs filler, and type vs predicate distinctions from a connectionist representation and reasoning perspective. *Proc. of AAAI-94* 99
- Al-Asady, R., 1995, *Inheritance Theory: An Artificial Intelligence Approach*. Ablex Publishing 92, 94
- Aleksander, I. & Morton, H.B., 1993, *Neurons and Symbols*. Chapman & Hall 114
- Aleksander, I., 1996, *Impossible Minds: My Neurons, my consciousness*. Imperial College Press 113, 309
- Aleksander, I., Brown, C., Evans, R.G. & Sales, N.J., 1995, MAGNUS: A review of current work in learning cognitive skills. *WnnW 95*, University of Kent at Canterbury 113
- Amit, D.J., 1989, *Modelling Brain Function*. Cambridge University Press 17, 37, 60, 162, 247, 288, 298
- Anderson, J.A., 1972, A simple neural network generating an interactive memory, *Math. Bio.* 14, pp197-220 49
- Arbib, M., 1994, Schema Theory: Cooperative computation for brain theory and distributed AI. In Honovar, V. & Uhr, L. (eds), *Artificial Intelligence and Neural Networks: Steps towards principled integration*. Academic Press 320
- Austin, J. & Filer, R., 1995, *Using Correlation Matrix Memories for Inferencing in Expert Systems*. Advanced Decision Technologies, Brunel University 108
- Austin, J. & Stonham, T., 1987, Distributed associative memory for use in scene analysis. *Image and Vision Computing*, 5, 251-260 53, 56
- Austin, J., 1997, A distributed associative memory for symbolic reasoning. In Sun, R. & Alexandre, F. (eds.), *Connectionist-Symbolic Integration*, Lawrence-Erlbaum 108
- Austin, J., Jackson, T. & Wood, A., 1991, Efficient Implementations of Massive Neural Networks, in Delgado-Frias, J.G. & Moore, W.R. (eds), *VLSI for Artificial Intelligence and Neural Networks*, Plenum Pub. Corp. 53

B

- Barron, A.R., 1991, Complexity regularization with application to artificial neural networks. In (Roussas, G., ed.), *Non-parametric functional estimation and related*

topics34

Baum, E.B., Moody, J. and Wilczek, F., 1988, Internal representations for associative memory, *Biol. Cyber.*, 59, pp217-22850

Bishop, C.M., 1995, *Neural Networks for Pattern Recognition*. Oxford Press. 26, 137, 255

C

Calvin, W.H., 1995, Cortical columns, Modules and Hebbian Cell Assemblies. In Arbib, M.A.(ed), *The Handbook of Brain Theory and Neural Networks*, Bradford. 16, 303

Calvin, W.H., 1998, *The Cerebral Code: thinking a thought in the mosaics of the mind*. MIT Press 16, 308

Carpenter, G.A. & Grossberg, S., 1994, Integrating Symbolic and Neural Processing in a Self-Organizing Architecture for Pattern Recognition and Prediction. in Honavar, V. & Uhr, L. (eds.), *Artificial Intelligence and Neural Networks*. Academic Press 143

Casasent, D. & Telfer, B., 1992, High Capacity Pattern Recognition Associative Processors. *Neural Networks*, Vol. 5, pp687-698 53

Chalmers, D. J., 1990, Syntactic Transformations on Distributed Representations, *Con. Sci.*, Vol. 2, Nos 1 & 2, pp 53-62 79

Chomsky, N., 1965, *Aspects of the Theory of Syntax*. MIT Press 41

Chrisman, L., 1991, Learning Recursive Distributed Representations for Holistic Computation, Internal memo CMU-CS-91-154, Carnegie Mellon University, Pittsburgh 80

Cover, T.M. & Thomas, J.A., 1991, *Elements of Information Theory*, Wiley-Interscience 159

Crick, F. & Asanuma, C., 1986, Certain Aspects of the Anatomy and Physiology of the Cerebral Cortex. In *Parallel Distributed Processing*, vol. 2. Cambridge: MIT Press 16

Crick, F., 1982, Do dendritic spines twitch? *Trends in Neurosciences*, 5, 44-46 18

Crick, F., 1984, Function of the thalamic reticular complex: the searchlight hypothesis. *Proc. of Nat. Acad. of Sci.*, 81, pp4586-4590 289

D

Desieno, D., 1988, Adding a Conscience to Competitive Learning, *IEEE Int. Conf. on Neural Networks*, vol.1. 32

Dolan, C.P. & Smolensky, P., 1989, Tensor Product Production System: a Modular Architecture and Representation, *Con. Sci.*, Vol. 1, No. 1, pp 53-68 73

E

Ernst, G.W. & Newell, A., 1969, *GPS: A Case Study in Generality and Problem Solving*. New York Academic Press 19

F

Fahlman, S., 1979, *NETL: A System for Representing and Using Real World Knowledge*. MIT Press 92

Fodor, J., 1975, *The Language of Thought*. Harvard University Press 39

Fodor, J.A. & Pylyshyn, Z.A., 1988, Connectionism and Cognitive Architecture: A Critical Analysis. *Cognition* 28: 3-72 39, 71, 128

G

Gardner, E., 1988, The phase space interactions in neural network models. *Journal of Physics*, 21A, p25 73

Gardner-Medwin, A.R., 1976, The recall of events through the learning of associations between their parts, *Proc. R. Soc. Lond. B.* 194, pp375-402 63

Graham, B. & Willshaw, D.J., 1995, Capacity and information efficiency of a brain-like associative net, *NIPS* 7:513-520 57

Graham, B. & Willshaw, D.J., Improving recall for an associative memory. *Biol. Cyber.* 72:337-346 57

Gross, R.D., 1996, *Psychology. The Science of Mind and Behaviour*. 3rd ed., Hodder & Sloughton 70

Grossberg, S., 1976a, Adaptive pattern classification and universal recording: I. Parallel development and coding of neural detectors. *Bio. Cyb.* 23, 121-134 157, 307

Grossberg, S., 1988, Competitive learning: From interactive activation to adaptive resonance, in S. Grossberg, (ed.), *Neural Networks and Natural Intelligence*, Cambridge 143, 256

H

Hall, Z.W., 1992, *An Introduction to Molecular Neurobiology*. Sinauer 18, 222

Haykin, S., 1994, *Neural Networks, A Comprehensive Foundation*. MacMillan 255, 270

Hebb, D.O., 1949, *The organisation of behaviour*. Wiley 48

Hely, T.A., Willshaw, D.J. & Hayes, G.M., 1999, A new approach to Kanerva's sparse distributed memory, To appear in *IEEE trans. on neural networks* 58

- Hertz, J., Krogh, A. & Palmer, R.G., 1991, Introduction to the theory of neural computation. Addison-Wesley 60, 206
- Hinton, G.E. & Sejnowski, T.J., 1986, Learning and Relearning in Boltzmann Machines, in Rumelhart, D.E. & McClelland, J.L., Parallel Distributed Processing, MIT Press 66, 71, 330
- Hinton, G.E., 1990, Mapping Part-Whole Hierarchies into Connectionist Networks, Art. Int. 46(1-2):47-75 71
- Hinton, G.E., McClelland, J.L. & Rumelhart, D.E., 1986, Distributed Representations. In Parallel Distributed Processing, vol. 1. Bradford 71
- Hirahara, M., Oka, N & Kindo, T., 1997, Associative memory with a sparse encoding mechanism for storing correlated patterns. Neural Networks, Vol. 10, No. 9, pp1627-1636 61
- Honovar, V. & Uhr, L., eds., 1994, Artificial Intelligence and Neural Networks. Steps Towards Principled Integration. Academic Press 43
- Hopfield, J.J. & Tank, T.W., 1985, 'Neural' computation of decisions in optimization problems. Bio. Cyb. 52, 141-152 317
- Hopfield, J.J., 1982, Neural Networks and Physical Systems with Emergent Collective Computational Abilities, Proc. Nat. Acad. Sci. of USA, 79 17, 37, 60
- Hopfield, J.J., 1984a, Neurons with Graded Response have Collective Computational Properties like those of Two-State Neurons, Proc. Nat. Acad. Sci. of USA, 81 37
- Hubel, D.H. & Wiesel, T.N., 1977, Functional architecture of macaque visual cortex. Proc. of Roy. Soc. of London, Series B 198, 1-59 309
- J**
- Jacobs, R.A., 1988, Increased rates of convergence through learning rate adaption. Neural Networks 1, 295-307 33
- Johnson, M., 1991, Superscalar Microprocessor Design, Prentice Hall 322
- Jolliffe, I.T., 1986, Principle Component Analysis. New York: Springer-Verlag 9, 30
- K**
- Kanerva, P., 1988, Sparse Distributed Memory, MIT Press 57
- Kelso, J.A.S., 1997, Dynamic Patterns. The Self-Organisation of Brain and Behaviour. MIT Press 3, 247
- Klein, S.B., 1991, Learning: Principles & Applications. 2nd Edition. McGraw-Hill 17, 343

- Kohonen, T., 1972, Correlation matrix memories, *IEEE Trans. on Computers* C-21, pp353-359 49
- Kohonen, T., 1982a, Self-organised formation of topologically correct feature maps, *Biological Cybernetics* 43 31
- Kohonen, T., 1982a. Self-organisation of topologically correct feature maps. *Biological Cybernetics* 43, 59-69 30
- Kolb, B & Wishaw, I.Q., 1996, *Fundamentals of Human Neuropsychology*, 4th ed., Freeman 16, 121
- Kolodner, J., 1993, *Case Based Reasoning*. Morgan Kaufman 314, 320
- Kosko, B., 1988, Bidirectional associative memories. *IEEE Trans. on Sys., Man & Cyber.*, 18: 49-60 64
- Kothari, R., Lotikar, R. & Cahay, M., 1998, State-dependent weights for neural associative memories, *Neural Computation* 10, pp59-71 67
- Kurzweil, R., 1990, *The Age of Intelligent Machines*. MIT Press 2

L

- Lange, T.E. & Dyer, M.G., 1989, Frame Selection in a Connectionist Model of High-Level Inferencing. *Proc. of the Ann. Conf. of Cog. Sci. Soc.*, pp 706-713 97
- Lange, T.E. & Dyer, M.G., 1989, High-level Inferencing in a Connectionist Network, *Con. Sci.*, Vol.1, No. 2, pp 181-217 97
- Lapin, L.L., 1990. *Probability and Statistics for Modern Engineering*. 2nd Edition. PWS-Kent 192
- Leake, D.B., 1996, *Case-Based Reasoning. Experiences, Lessons & Future Directions*. MIT Press 26, 314
- Leung, C.S., 1993, Encoding method for bidirectional associative memory using projection on convex sets. *IEEE Trans. on Neural Networks*, vol. 4, no. 5, pp879-881 65
- Leung, C.-S., 1994, Optimum learning for bidirectional associative memory in the sense of capacity. *IEEE Trans. on Sys., Man & Cyb.*, vol. 24, no. 5, pp791-796 66
- Leung, C.S., Chan, L.-W. & Lai, E., 1995, Stability, capacity and statistical dynamics of second-order bidirectional associative memory, *IEEE Trans. on Sys., Man and Cyb.*, vol. 25, no. 10, pp1414-1424 66
- Linsker, R., 1986, *From Basic Network Principles to Neural Architecture*. *Proc. of the Nat. Acad. of Sciences of the USA*, 83 30

Little, W.A. & Shaw, G.L., 1975, A statistical theory of short and long term memory, *Bahav. Bio.*, vol. 14, pp115-133 50

Lynch, G. & Baudry, M., 1984a, The biochemistry of memory: A new and specific hypothesis. *Science*, 224, 1057-1063 18

M

Malek, M. & Amy, B., 1997, A preprocessing model for integrating case-based reasoning and prototype-based neural network. In Sun. R. & Alexandre, F. (eds), *Connectionist-Symbolic Integration*. Lawrence-Erlbaum Assoc 113

Martin, J.H., 1996, *Neuroanatomy: text and atlas*, 2nd Edition. Appleton & Lange 16, 308

Marx, K. & Engels, F., 1848, *Manifesto of The Communist Party* 128

McClelland, J.L. & Rumelhart, D.E., 1981, An interactive activation model of context effects in letter perception, part 1. *Psychol. Rev.* 88, pp375-407 72

McCulloch, W.S. & Pitts, W.A., 1943, A logical calculus of the ideas immanent in neural nets. *Bull. Math. Biophys.* 52

McDermott, D., 1976, Artificial Intelligence meets natural stupidity. Reprinted in Haugeland (ed.), 1981, *Mind Design*. MIT Press 106

McDermott, J., R1: A Rule-Based Configurer of Computer Systems. *Artificial Intelligence*, 19(1):39-88 21

McEliece, R.J., Posner, E.C., Rodemich, E.R. & Venkatesh, S.S., 1987, The capacity of Hopfield associative memory. *IEEE Trans. on Inf. Theory*, vol. IT-33, No. 4, pp461-68 160

Miikkulainen, R., 1993, *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicons, and Memory*. MIT Press 112

Miikkulainen, R., 1994, *Integrated Connectionist Models: Building AI systems on Subsymbolic Foundations*. In Honovar, V. & Uhr, L.(eds). *Artificial Intelligence and Neural Networks. Steps Towards Principled Integration*. Academic Press 112

Minsky, M., 1988, *The Society of Mind*. Touchstone, Simon & Shuster 88

Minsky, M.L & Papert, S.A., 1969, *Perceptrons*. MIT Press 33, 36, 40

Minsky, M.L., 1961, Steps towards Artificial Intelligence, *Procs. of the Institute of Radio Engineers* 49, 8-30 33

N

Nadal, J-P. & Toulouse, G., 1990, Information storage in sparsely coded memory nets, *Networks*, vol. 1, pp61-74 51

Narendra, K. & Thathachar, M.A.L., 1989, Learning Automata, an Introduction. Prentice-Hall 222

Newell, A. & Simon, H.A., 1963, GPS, a program that simulates human thought. In E.A. Feighbaum & J. Feldman (eds.), Computers & Thought. McGraw-Hill 86, 87

Newell, A., 1990, Unified Theories of Cognition. Harvard 86, 124, 314

P

Palm, G., 1980, On associative memory, Bio. Cybernetics, 36, pp19-31 57

Patterson, D.W., 1990, Introduction to Artificial Intelligence and Expert Systems. Prentice-Hall 26

Plate, T., 1996, Holographic reduced representations: convolution algebra for compositional distributed representations. Proc. of the 12th IJCAI 82

Plate, T., 1997, Structure matching and transformation using distributed representations. In Sun, R.& Alexandre, F., (eds), Connectionist-Symbolic Integration, Lawrence-Erlbaum 82

Pollack, J.B., 1990, Recursive Distributed Representations, Art. Int., 46(1-2): 77-105 76

Q

Quillian, M.R., 1968, Semantic memory. In Minsky, M., (ed.), Semantic Information Processing, MIT Press 23

R

Rosenblatt, F., 1958, Principles of Neurodynamics. Spartan 66

Rumelhart, D.E. & McClelland, J.L. (eds), 1986, Parallel distributed processing, vols. 1 & 2. Bradford 49, 293

Rumelhart, D.E., Hinton, G.E. & Williams, R.J, 1986b, Learning internal representations by error propagation. In Parallel Distributed Processing: Explorations in the Microstructure of Cognition., Vol. 1, MIT Press 28, 50, 143

Russell, S. & Norvig, P., 1995, Artificial Intelligence: A Modern Approach. Prentice-Hall 18, 23, 87, 134, 312, 322

S

Sandewall, E., 1986, Non-monotonic inference rules for multiple inheritance with exceptions. Proc. of IEEE-86, vol. 74, pp1345-1353 94

Searle, J.R., 1992, The Rediscovery of the Mind. MIT Press 114

Selfridge, O.G., 1958, Pandemonium: a paradigm for learning. In Anderson, R.A., (ed), 1989, Neurcomputing: Foundations of Research 88

- Shafer, G.A., 1979, *Mathematical Theory of Evidence*. Princeton 26
- Shastri, L. & Ajjanagadde, V., 1993, From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. In *Brain and Behav. Sci.*, 16, pp417-494 99
- Shastri, L., 1988, *Semantic Networks: An Evidential Formalization and its Connectionist Realization*. Pitman, London 92, 95, 96
- Shastri, L., 1990, Connectionism and the Computational Effectiveness of Reasoning. *Theoretical Linguistics*, vol. 16, no. 1, pp65-87 100
- Shastri, L., 1996, Temporal Synchrony, Dynamic Bindings, and SHRUTI: a representational but non-classical model of reflexive reasoning. *Behav. and Brain Sci.*, 19 (2), pp 331-337 99
- Shastri, L., 1999, Advances in SHRUTI - A neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. To appear in *Applied Intelligence* 99
- Shepherd, G.M. & Koch, C., 1990, Introduction to Synaptic Circuits, in *The Synaptic Organisation of the Brain*, 3rd Ed. OUP 29
- Shepherd, G.M. (ed), 1990, *The Synaptic Organisation of the Brain* (3rd ed). New York: OUP 15, 16
- Shoham, Y., 1990, Non-monotonic reasoning and causation. *Cog. Sci.*, pp913-923 112
- Simpson, P.K., 1990, Higher-order and intraconnected bidirectional associative memories. *IEEE Trans. on Sys., Man & Cyber.*, vol. 20, no. 3, pp637-653 65
- Smolensky, P., 1986, Harmony Theory, in Rumelhart, D.E. & McClelland, J.L., *Parallel Distributed Processing*, vol.1, MIT Press 330
- Smolensky, P., 1988, On the Proper Treatment of Connectionism. *Behavioural & Brain Sciences* 11, 1-74 38, 71
- Smolensky, P., 1991, Connectionism, Constituency, and the Language of Thought. In Loewer, B. & Rey, G. (eds), *Meaning in Mind: Fodor and his Critics*. Blackwell 73
- Steinbuch, K., 1963, *Automat und Mensch*. Springer 34, 48
- Stork, D.G., 1997, Scientist on the Set: An interview with Marvin Minsky. In *HAL's Legacy: 2001's Computer as Dream and Reality*. MIT Press 1, 4
- Stroustrup, B., 1991, *The C++ Programming Language*. 2nd Edition. Addison Wesley

Sun, R., 1992, On Variable Binding in Connectionist Networks. *Con. Sci.*, vol. 4, No. 2, pp93-124 103

Sun, R., 1994, A Neural Network Model of Causality. *IEEE Trans. on Neural Networks*, vol. 5, No. 4, pp604-611 103

Sun, R., 1994, Logics and Variables in Connectionist Models: A Brief Overview, in Honovar, V. & Uhr, L. (eds.), *Artificial Intelligence and Neural Networks. Steps Towards Principled Integration*, Academic Press 42

Sun, R., 1994, Robust Reasoning: Integrating Rule-based and Similarity-Based Reasoning. Internal Memo, Dept. of Comp. Sci., Univ. of Alabama 103, 111

T

Touretzky, D.S. & Hinton, G.E., 1988, A Distributed Connectionist Production System, *Cog. Sci.* 12, pp 423-466 89

Touretzky, D.S., 1990, BoltzCONS: Dynamic Symbol Structures in a Connectionist Network, *Art. Int.* 46, pp5-46 89

Touretzky, D.S., 1984, *The Mathematics of Inheritance Systems*. Pitman, London 92

Touretzky, D.S., Horty, J. & Thomason, R., 1987, A clash of intuitions: the current state of non-monotonic inheritance systems. *Proc. of IJCAI-87*, pp476-482 93

Turner, M. & Austin, J., 1997, Matching performance of binary correlation matrix memories. *Neural Networks*, Vol. 10, No. 9, pp1637-1648 57

V

Van Gelder, T., 1990, Compositionality: a connectionist variant on a Classic theme. *Cog. Sci.*, 14 79

Vidyasagar, M., 1993, Location and stability of the high-gain equilibria of nonlinear neural networks, *IEEE Trans. on Neural Networks*, Vol. 4, No. 4, pp660-671 61

von der Malsburg, C., 1981, Internal Report 81-2, Dept. of Neurobiology, Max-Planck Institute for Biophysical Chemistry 289

Von Neumann, J., 1958, *The Computer and the Brain*. Yale University Press 1

W

Werbos, P.J., 1974, Beyond regression: New tools for prediction and analysis in the behavioural sciences. PhD thesis. Harvard University 33, 49

Weste, N.H.E. & Eshraghian, K., 1992, *Principles of CMOS VLSI Design: A System Perspective*. 2nd Edition. Addison-Wesley 158, 305

Willshaw, D.J. & von der Malsberg, C., 1976, How patterned neural connections can be set up by self-organisation, Proc. of the Royal Soc. of London, B 194:431-445
31

Willshaw, D.J., Buneman, O.P. & Longuet-Higgins, H.C, 1969. Non-holographic associative memory. Nature (London) 34, 49, 109

Wolpert, D.H. (ed.), 1995, The Mathematics of Generalisation. Proceedings of the SFI/CNLS Workshop on Formal Approaches to Supervised Learning. Addison-Wesley 20

Z

Zadeh, L.A., 1965, Fuzzy Sets. Information & Control, 8:338-353 5

Zhang, B.-L., Xu, B.-Z. & Kwong, C.-P., 1993, Performance analysis of the bidirectional associative memory and an improved model from the matched-filter viewpoint. IEEE Trans. on Neural Networks, vol. 4, no. 5, pp864-872 65