Proceedings of International Joint Conference on Neural Networks, Atlanta, Georgia, USA, June 14-19, 2009

Optimal Connectivity In Hardware-Targetted MLP Networks

A. D. Rast, S. Welbourne, X. Jin and S.B. Furber

***Abstract*—**

In large neural networks, partial connectivity is both biologically plausible and a matter of necessity when targetting a hardware implementation. We are using the SpiNNaker neural chip multiprocessor to model such networks as a drop-in replacement for the Lens network simulator. For the popular MLP network, a theoretical model of the relation between connectivity, network size and gain in the activation function provides a method to set these parameters to near-optimal values. Using the model, we run a series of network simulations in Lens, permuting the parameters to explore the effects in 2 networks of different size and application. Initial test results show a clear connectivity-gain relation and a benefit to partial connectivity in both networks, with optimal hidden-output connectivity values ranging from ∼10%-∼30% depending on the network type. We show that optimal connectivity-gain settings reduce training time, minimising error oscillations during learning. Preliminary analysis also suggests that while very low connectivities may improve error they may also result in decreased adaptivity to new inputs or component failure. These results in combination with the theoretical relation give a method for determining reasonable initial connectivity and gain values *at design time* for an MLP network, allowing more efficient use of hardware resources such as SpiNNaker and faster simulations in any software environment. They also suggest a different way of considering the problem of MLP network design: rather than specify a fixed number of neurons, specify a fixed number of connections and vary the number of neurons to reach optimal connectivity.

I. Introduction

LARGE neural networks have obvious reasons to have partial rather than full connectivity. Observations on the brain demonstrate certain structural patterns that suggest other than random or full connectivity [1]. Nonetheless, most neural network models use full or random connectivity as an implementation convenience, partly to avoid obscuring the significance of other model parameters behind effects of the connectivity statistics [2]. This method is adequate for small networks, containing up to ∼1000 neurons, but computationally unmanageable at larger scales. It is unreasonable to assume that random partial connectivity will lead to an optimal structure in a large network performing a specific task; we might expect rather that the task itself would suggest a structured pattern of connectivity matching in some way the characteristics of the task. However, usually the defining characteristics are not known *a priori*, leading to a combination of intelligent guesswork and pure trial-and-error to find a good structure. Better methods for defining the connectivity, or at least for setting a reasonable starting point to optimise via some adaptive learning rule, are clearly essential as neural network models scale to very large sizes.

The authors are with the School of Computer Science, The University of Manchester, Manchester, UK (email: {rasta}@cs.man.ac.uk).

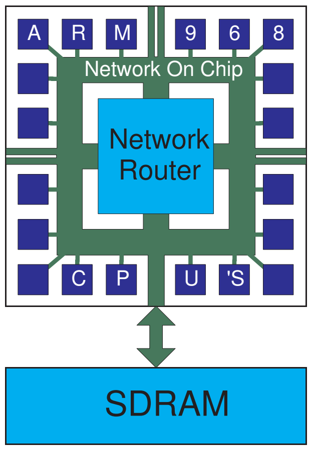


Fig. 1. SpiNNaker Chip Architecture.

The general problem of optimal routing is NP-complete, therefore we cannot expect and do not need to find the *global optimum* connectivity; it suffices to find reasonably good solutions. Since the determining characteristics are unknown a priori, these solutions must furthermore only use properties of the data that can be identified at the inputs and outputs of the neural network. We focus here on a specific neural architecture: the classical multilayer perceptron (MLP), since in the first place we expect that the optimal connectivity is architecture-specific, and secondly, the MLP, as the most common neural network architecture in actual use, is both a good exemplar for a general optimisation technique and the one in which a specific solution would be most valuable at present. The task, then, is this: given a set of inputs and outputs with known characteristics, along with a network of known size, what is the form of the optimal connectivity structure? We introduce here a series of rules based on connectivity per neuron and transfer-function sensitivity that make it possible to optimise an MLP neural network for a particular application.

II. SpiNNaker Hardware Mapping

Efficient network connectivity becomes particularly significant in the context of hardware systems. In such devices, available circuit densities constrain the number of connections available at any given size. Various hardware neural network implementations have used signal multiplexing to mitigate this problem, and yet inevitably there is an absolute limit where the number of active connections in the model would exceed the available hardware bandwidth. We are targetting the MLP implementations under examination for SpiNNaker (fig. 1), a general-purpose neural chip multiprocessor capable in principle of supporting virtually any network model. Each SpiNNaker chip contains up to 20 general-purpose ARM968 processors to implement the neurons' functionality. The ARM968 is a power-and-area optimised core that achieves its design goals through several important tradeoffs. The processor has no floating-point support; all neural functions must be mappable to a fixed-point representation, and in particular, transcendental functions such as the sigmoid threshold function typical of the MLP must be implemented as lookup tables. Local memory is small, each processor having 32K instruction memory and 64K data memory. By using a large, external 1 Gbit SDRAM in combination with an application-specific DMA controller we have developed a method [3] to make a larger memory area for synaptic data appear virtually local to the processor, provided that the processor does not need to provide immediate response to an input packet. Nonetheless, for continually-in-use data such as neural parameters, some data and instructions must remain permanently resident in local memory, affecting the implementation of the model.

SpiNNaker's connectivity fabric has the following features: 1) It is asychronous, in other words, each signal is self-timed and not referenced to a global system clock. This permits true concurrency in the system, each neural output signal being timing-independent from others and meaning that each of the processors on the chip runs independently without regard for coherency with its peers. 2) It is packet-switched by source processor, meaning that the system uses only the originating processor's address in order to route its outputs to any other processors that may use these outputs. 3) The router itself, one per chip, is a programmable multicast associative router with 1024 entries. While this architecture permits a fully configurable architecture and the ability to group both sources and local routing destinations into associated "bundles", it likewise sets finite constraints on what topologies are routable within the number of entries. 4) Each chip operates within a multi-chip environment connected to 6 neighbouring chips in a hexagonal mesh topology. The *physical* topology of the system is completely decoupled from the *virtual* topology of the modelled network, provided that the network topology is routable.

In previous tests we found that the 1024 entries in the local routing table tend to be the limiting factor establishing routability of the network [4]. Further analysis of the MLP network revealed an additional constraint. The backpropagation learning rule typical for the MLP requires that information be passed backwards through the synapses. Since SpiNNaker's communications fabric is source-routed, if the mapping of an MLP onto SpiNNaker followed a naïve one-processor-per-neuron model, it would require 2 router entries per connection: one for the forward link and one for the reverse link. Instead, we have mapped the MLP by dividing input processing into subunits representing square submatrices of the overall connectivity matrix. (fig. 2) This distributes the routing problem evenly between forward and backward signal propagation. Nonetheless, the routing

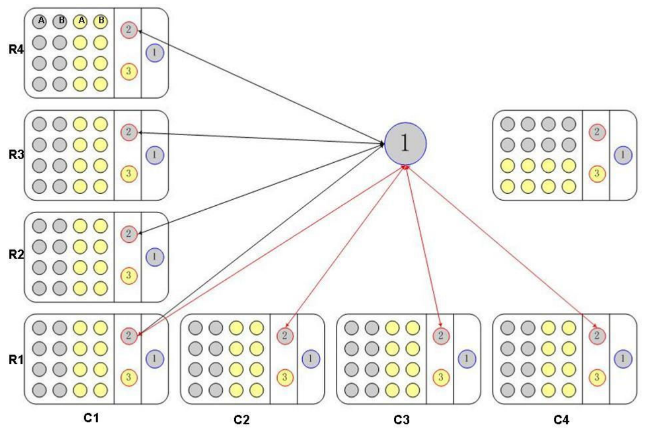


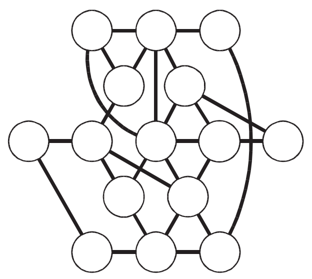
Fig. 2. SpiNNaker mapping of the MLP neural network. Each small circle is a processor within the box representing a SpiNNaker chip. The grey and yellow colours represent 2 separate submatrices of the connectivity matrix (arranged in source $\times$ target row-by-column format). Processors such as A and B on the left side of the line within each processor box perform the initial element calculations on each submatrix element representing a particular connection. Processors such as 2 and 3 on the right side of the line perform aggregation of the submatrix elements into local row (forward) and column (reverse) partial sums. Processors 1 perform the final summation and thresholding at the neurons' output.

overhead is significant, and furthermore, since the final sum-and-threshold processors "1" can only update after receiving all their partial sums, it is important to minimise both the number of partial sums to compute and their mean path length to processor "1". The simplest way to achieve this is by reducing the total connectivity of the network.

If, however, by reducing the number of connections important performance measures like training time or classification speed deteriorate, then the network optimisation is of little value. Therefore the goal is to find connectivity patterns that maximise the speed and learning rate per connection. In addition, it is essential to balance the communications load across the network. This is firstly because SpiNNaker's routing model assumes average-case traffic conditions, and secondly, since the overall sum-and-threshold processors "1" require all partial sums for any result, significant delays in receiving the partial sum from any processor will greatly slow down the overall update rate per neuron. Significant delays would result if there were local congestion: we have previously performed simulations to show that local congestion potential increases (as expected) as the network connectivity increases. Note however that the pattern of connectivity in the neural network has no direct impact on local congestion because of the complete decoupling of the physical SpiNNaker topology from the virtual network topology. The connectivity analysis that follows considers optimal patterns with respect to a SpiNNaker hardware implementation, where available processing resource is large relative to available connection resource.

III. Theoretical Connectivity Models

Exploration of the tradeoffs between connectivity and performance in neural networks has been somewhat specialised.



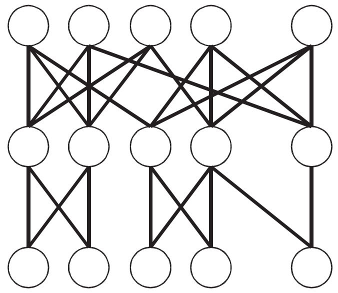


Fig. 3. Partial connectivity networks. At top is a "small-world" network. Note the absence of distinct layers. At bottom is a partially-connected MLP. Here neurons are clearly arranged in layers, the connectivity between each layer being less than full.

One prominent line of research involves examination of the benefits of "scale-free" and "small-world" networks. "Small-world" networks (fig. 3, top) possess simply the property that local connections are more probable than long-range ones [5]. Such networks are both biologically plausible [6] and computationally efficient in terms of storage capacity of an autoassociator per connection [7]. Scale-free topologies, e.g. [8], give a probability for a given neuron to have a given number of connections N that scales as an inverse power law: $\frac{1}{N^c}$. Such topologies appear to have good performance in autoassociator-like tasks but are of questionable biological plausibility [1]. Calcroft, et al. [9] observed near-optimal performance when connection lengths follow a Gaussian distribution centred on 0. None of these studies, however, considered learning time, and all of them use the same ring-shaped (1-layer) topology, distinctly different from the partial-connectivity MLP (fig. 3, bottom) KrishnaKumar [2] examines the learning time in an MLP topology, using a learning algorithm to optimise the connectivity. In [10] the authors examine the learning time within a feedforward MLP-type network. Their findings indicate an improvement in learning error and time with intermediate values of connectivity, using random rewiring across all layers. All of the above studies, however, consider a network of fixed size with respect to number of neurons, which may not be the most efficient technique given that in practice *connections* dominate the area in networks of reasonable size [11]. Furthermore, the results are purely empirical findings not linked to a theoretical model. A model relating connectivity and network size would therefore be extremely useful to generalise connectivity determination for arbitrary large-scale networks, particularly for targetting to hardware implementations.

The local connectivity of a given neuron affects both its output error per time step and its contribution to error in any subsequent layers. Assume a neuron j connected to n neurons i in a previous layer and to l neurons k in the next layer. Neuron j has error for input S δ_{Sj} from the standard backpropagation formula:

$$\frac{dB_j}{dO_j}\sum_k \delta_{Sk} w_{jk}$$

where B_j is the input aggregate before thresholding and O_j the output of neuron j. We can break this into an input-error term in the derivative and an output-error term in the backpropagated sum. On the input side of the neuron, the total error is

$$\sum_i \delta_i.$$

Let this be the sum of significant and insignificant inputs:

$$\sum_x \delta_{ix} + \sum_{n-x} \delta_{i0}.$$

Inputs i contribute to term B_j and may cause the derivative to deviate far from its correct value, resulting in a large weight correction. If a weight should actually be near-zero, indicating that neuron i does not contribute significantly to the correct output for neuron j, error terms involving these elements will dominate the total error. When $\frac{dB_j}{dO_j}$ is large, changes in the δ_{i0} non-contributing inputs would cause large error updates for neuron j. We therefore expect 2 results. Sparse inputs will cause smaller errors by reduction in the number of non-contributing inputs:

$$\sum_x \delta_{ix} + \sum_{n-x-y} \delta_{i0} \le \sum_x \delta_{ix} + \sum_{n-x} \delta_{i0}.$$

Lower gain values will minimise the impact of such non-contributing inputs by mapping them onto more nearby points in the output: if

$$\frac{dB_j}{dO_j} \le \frac{d'B_j}{d'O_j}, \frac{\sum \delta_i}{dO_j} \le \frac{\sum \delta_i}{d'O_j}.$$

On the output side of neuron j, its error propagates through successive layers. Following the backpropagation rule, the error propagated is

$$\sum_k \delta_{Sk} w_{jk}.$$

If a connection has a nonzero weight it will contribute to the total error through its output neuron. For the same reasons as the input side, total error propagated will be greater than the optimal error propagated, when there are inputs to neurons k that should have a near-zero weight, by

$$\sum_k w_{j0k}\delta_{Sk},$$

using the same notation as the input side to represent non-contributing inputs. Assuming the weights w_{j0k} and errors δ_{Sk} are randomly distributed, this "overpropagated" error is

$$(l-x)\bar{w_{j0k}}\bar{\delta_{Sk}},$$

where $l-x$ is the number of non-contributing inputs to neurons k. Each neuron k would ideally have

$$N_{wk} = l - x_k$$

initial connections, x_k being the number of non-contributing inputs for the particular neuron k. On the output we therefore expect that the optimal connectivity between j and k will be when the mean number of actual connections equals the mean number of required connections:

$$C_{jk} = \frac{\bar{N_w}k}{m},$$

where m is the number of neurons j.

For output neurons, we define the stopping criterion V_{ok} indicating how accurately outputs must match the target for the network response to be considered correct. Specified on a per-neuron basis,

$$(T_{Sk} - O_{Sk}) \leq V_{ok},$$

where T_{Sk} is the correct output. Maximum output coding efficiency occurs when the total number of states to be represented in neuron k is

$$\frac{O_{kmax} - O_{kmin}}{V_{ok}} = \frac{1}{V_{ok}}$$

for normalised outputs. The number of required significant inputs J' should be at minimum enough to code all the possible output states:

$$\sum_{r=1}^{J'} (r^{\frac{1}{V_{pj}}-1})_{J'} C_r = 1/V_{ok},$$

where V_{pj} is the internal resolution of the previous layer: how many valid states are coded. (For typical sigmoidal networks the value is usually 1: a hidden neuron registers the relative presence or absence of a template pattern) It is now possible in principle to work backwards through the error propagation, substituting the values for the minimum number of required inputs for each output neuron to determine the mean and thereby a reasonable approximation to the optimal connectivity. When connectivity is heavily optimised, however, the network may be vulnerable to overtraining. In practice it is best to start with a fairly aggressive criterion and a somewhat higher than optimal connectivity since the method assumes optimal coding.

IV. Critical Network Parameters

Previous connectivity examinations have primarily focussed on evaluating the effects of varying connectivities in a network of fixed number of neurons, generally with a "small-world" characteristic. In a hardware system such as SpiNNaker, however, it is likely that large neural networks will be more connection-limited than they are neuron-limited, given the finite routing resources. Simply varying connectivity with a fixed number of neurons may not lead to an optimal hardware-implementable network. Congestion issues also have a significant role. Since the synchronous nature of the update in SpiNNaker's MLP implementation requires the arrival of all contributing inputs before the processor computes an update, large numbers of inputs to a given output processor would lead to long delays before an update, as the system waited for distant inputs to arrive. In a congested situation, there would be additional delays, and since the SpiNNaker processing model drops packets after a certain time, a hub node could deadlock awaiting arrival of an input packet that had been dropped due to transient congestion. It is therefore more realistic to consider *uniform* partial connectivity on a per-layer basis.

MLP's have a distinct layered topology, making it useful to investigate the effects of differing inter-layer connectivities. For example, in a simple feedforward network with input, hidden and output layers, the input-hidden connectivity and hidden-output connectivity could be different. A high input-hidden connectivity and a low hidden-output connectivity would produce a network with rich feature representation but sparse feature selection in identification of an output class. The converse would instead limit feature complexity in an attempt to achieve high recognition accuracy through the integration of multiple, simple, separable features. The experiments therefore consider connectivity as an independent parameter on a per-layer basis, varying the degree uniformly throughout a given layer.

With a connection-limited system, it is practical to vary the number of neurons since they do not add substantially to the resource requirements. In the MLP model, however, both the number of input neurons and the number of output neurons remain fixed, because of the need to present the network with external input and training class identifications. Both of these originate in external sources that typically use a fixed format, thus only the hidden neurons may vary in number. Noting this constraint, however, the number of neurons within any given hidden layer may vary independently for similar reasons as the number of connections.

The third varying parameter in the model is gain (steepness of the threshold function). Within the SpiNNaker hardware, gain is a parameter loaded into instruction TCM. Memory use is a concern; with only 32K it is impractical to have different gain values across different (output) neurons mapped to the same processor. Furthermore, the lookup table to implement the threshold remains resident in the TCM at all times, therefore the implementation embeds most of the important parameters of the neurons' transfer function into the LUTs. Thus any given processor has the same parameters for all the neurons it implements. It is most practical to do this by mapping neurons in a given layer to the same processor or group of processors. Mapped in this way, gain may again vary on a per-layer basis, but not dynamically, thus Lens'

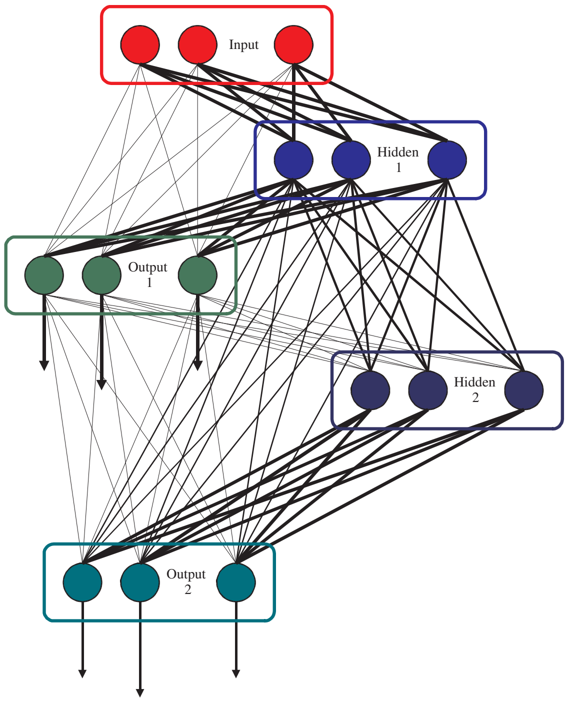


Fig. 4. Large phonetic network used in tests

ADAPTIVE_GAIN neuron type is not available. Experiments treat gain as a static parameter varied by run.

V. Modelling Setup

We have used the Lens [12] neural network simulator to run neural network simulations. The simulator ran 2 different neural networks: a relatively small network based on the Lens example hand-digits designed to recognise handwritten numbers. The network contains 64 input, up to 40 hidden, and 10 output neurons in a simple feedfoward configuration. The second network we ran was a larger one designed to recognise spoken words from a small fixed vocabulary (a development of the model in [13]) with further assistance in the form of visual input of the characters. This network contains a single input layer and 2 distinct groups each of hidden and output layers representing the visual and sonic characteristics.The input layer contains 400 units. The 2 output groups contain 800 and 61 neurons respectively. The 2 hidden groups have variable numbers of neurons; in the first round of tests we used 50 neurons per group. Inputs connect only to hidden group 1 from which there are 2 main connectivity paths: to output group 1 and to hidden group 2. Hidden group 2 then connects to output group 2 to establish a forking pattern of connectivity. In addition there are further connections from the input directly through to output group 1, and from output group 1 itself into hidden group 2 and output group 2 (fig. 4).

We varied 3 different parameter classes: connectivity, number of neurons, and gain. To test the effect of various parameter combinations we permuted each parameter in a series of steps. The permutation function can independently set the values in each of these parameters on a per-group basis, so that, for example, a network with 2 hidden groups could have different values for the output connectivity in each of those groups while maintaining the same gain, and in a later permutation vary the gain with the same or different connectivity values. It is, however, necessary to use discretion in which values to vary or the number of permutations rapidly becomes very large: for n parameters in k groups with v_n steps per parameter, the number of permutations would be $\prod v_n^k$. For the case of the smaller network we used a parameter step size leading to 5 different values for connectivity and gain over each of the input, hidden, and output layers, a total of 15,625 possible permutations. For the larger network we preferentially varied the connectivities in the preference order hidden-output, hidden-hidden, output-hidden, input-hidden, input-output, and output-output over 3 different values, and hidden unit gain over 2 different values. Considering only the actual connection paths as per the architecture above this leads to $3^8 * 2^2$ - 26,244 permutations. We ran each permutation for 500 training epochs in the small network, 1000 in the large one and measured the final and mean errors so as to get information both on ultimate classification performance and learning rate (using the mean as a proxy for the rate).

VI. Test Results

Results from all tests (figs. 5, 6, 7) demonstrate an approximately linear relationship between connectivity and error performance down to a certain minimum value. For the simpler hand-digits networks performance clearly also separates according to gain. At higher connectivities, lower gains show improved performance by decreasing the slope of the error line, thus the higher the connectivity, the greater the impact of reduced gain. We can interpret these results in the following way. Reduction of gain works by minimising the error in early stages of the training. With large gains and therefore steep transfer functions small deviations in input lead to possibly large output deviations, and hence large errors if there is any error in the initial inputs. Unless the network happened to begin with perfect weight matrices, these initial inputs would indeed create large errors, causing output fluctuation until the values settle. Smaller gain values yield lower fluctuations and therefore the ability to converge along a more closely monotonic error path. Per the theory, we also observe that this effect will be more pronounced when the connectivity is high, in agreement with the data.

If, as per the theory, we consider each connection as a symbolic path that codes for a particular subfeature, the minimal connectivity represents the point where the neural network exactly maps the training set. At this point further reductions in connectivity imply data loss. The network could be considerered "hard-wired" for the problem, thus this learning optimum occurs at the price of zero plasticity (no ability to learn new inputs). The observation that at this point the learning performance would be gain-independent is likewise consistent with the theory: if the structure fully encodes the information content of the training set, no changes in

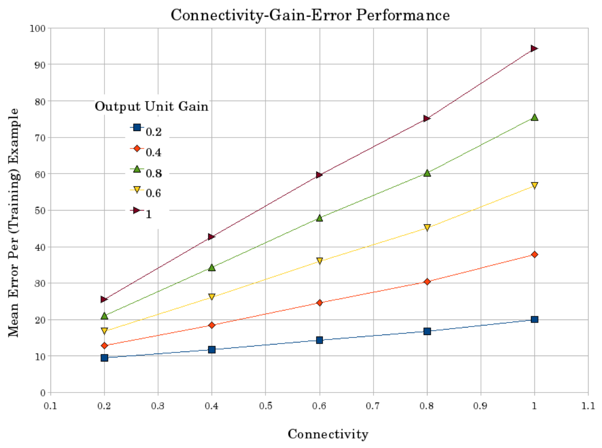


Fig. 5. Error-Connectivity results for hand-digits application. Data series are grouped by gain.

gain could encode more information, nor lead to better error performance (the structure perfectly routes the inputs into their associated subfeature neurons, thus steepness of the gain function serves no purpose in separating a correct input from an incorrect one). In the large neural networks, architectural complexity makes it difficult to separate the critical parameters, although the general pattern is clear. By grouping the series first by Hidden0-Output0 connectivity, then by Input-Output0 connectivity it is possible to identify groups. Each group contains 2 branches, identifiable as a high-gain (upper) branch and a low-gain (lower) branch. Overall, the 200-neuron-per-hidden-layer networks perform better than the 50-neuron-per-hidden layer networks, suggesting the improved computing power of additional feature representations per class. One interesting feature of the 50-neuron networks is the presence of some series (i.e. parameter combination sets) whose performance at all Hidden0-Output0 connectivities is comparable to the 200-neuron networks and considerably greater than the other series. These appear to occur with certain combinations of Input-Hidden0 and Output0-Output1 connectivities, although the precise nature of the relationships has not been determined.

Examination of the output performance with networks trained for the task reveals the effect of overtraining with lower connectivities. Fig. 8 gives the error with hand-digits networks that successfully learned the training set according to the maximum error criterion. These networks then ran a test set from a second, independent data source. With all connectivities the error is small, however, the mean error reaches its minimum at about 60% connectivity and increases (with greater variability) as the connectivity goes beneath this value.

VII. Network Design Implications

The experimental results show that partial connectivity can achieve up to 1.8 times faster training, in number of weight updates, even in neural networks of relatively small size. There are several possible contributors to this effect.

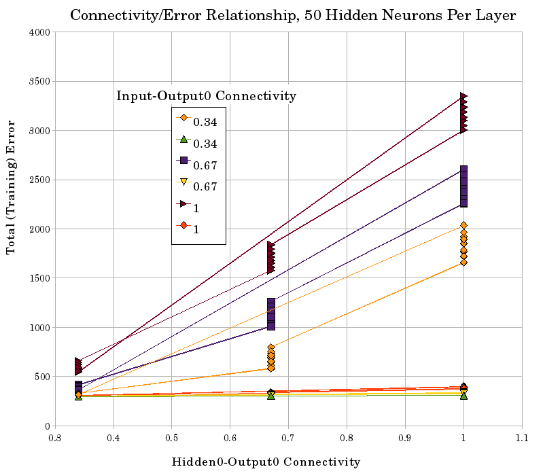


Fig. 6. Error-Connectivity results for the phonetic recognition network with 50 hidden units. Refer to the text for series groupings.

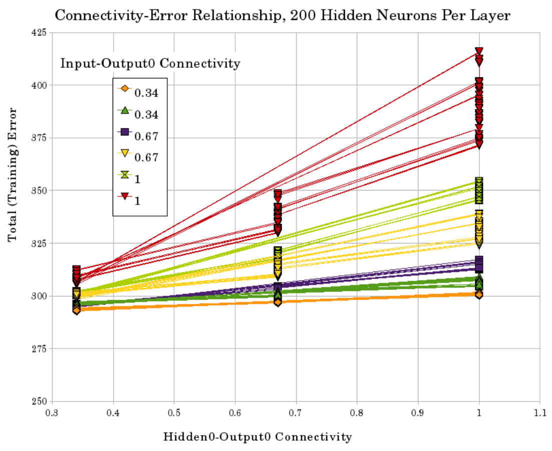


Fig. 7. Error-Connectivity results for the phonetic recognition network with 200 hidden units. Refer to the text for series groupings.

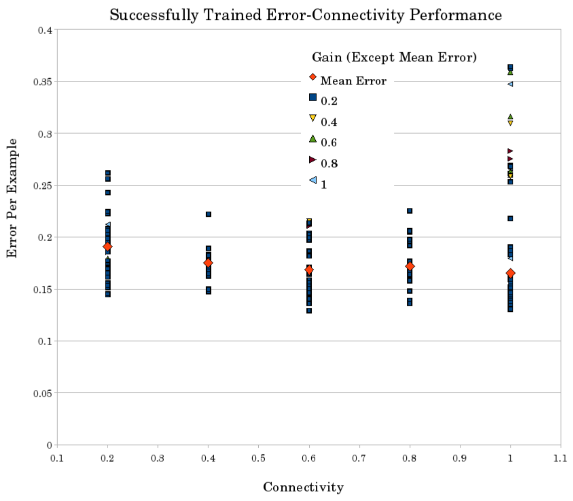


Fig. 8. Performance of the trained hand-digits network for configurations that successfully learned the task. Red diamonds are average values; blue squares are the individual results on each trial.

One possibility is symmetry breaking. In a large network under full connectivity, the mapping of a specific neuron in either the hidden or output layers to a specific combination of features or to an identified class is arbitrary. All the class and feature identification must lie in the weights. As a result, the network is initially class-undifferentiated and must spend a certain time during early learning to develop weights that bias individual neurons towards identification of a particular class or feature. The usual solution to this problem is to randomise the weight values. However, particularly with large networks, randomised weights at best give a weak asymmetry since a neuron receives inputs from many synapses. Simple statistical averaging will cause the net contribution to be near that of a uniformly weighted network. Unless the weight randomisation is heavily biassed towards zero the effect, therefore, will be the same as in the uniform full connectivity case. Thus strong symmetry breaking would only happen when the weight randomisation had a strong zero bias, equivalent to saying that the network had only partial connectivity. Experimental results nonetheless suggest that the symmetry breaking effect is relatively weak in networks of the size we examine. If the effect were strong, we would expect that reduced connectivity in *any* interlayer connection would produce improved error performance, when in fact the error improvement was most dramatic and obvious in the hidden-output connections. Such a result is more consistent with the second possibility: minimisation of error propagation.

The theoretical model predicts that reduction in the back-propagation of errors in delta-rule learning will reduce training time and give better error performance. If certain neurons in the output have large errors these errors can propagate throughout the network, resulting in large weight corrections in potentially irrelevant connections. Particularly early in training, the large error will tend to dominate the sum in neurons in the previous layer, propagating back through the network as a wave of large weight changes. Thus in a highly connected network, the overall error initially will fluctuate before the weights settle enough that inputs from irrelevant neurons have very small values. We observed this in actual training, where high-connectivity networks would tend to spend several epochs with oscillating or at best marginally decreasing error before starting to learn (fig. 9). By contrast a network with partial connectivity propagates fewer large errors throughout the network. Localising the backpropagation along the connected paths confines any weight adjustments to the neurons in the path, and since there are fewer, each one has a greater probability of contributing meaningfully to the total error. The effect is two-fold: first, it helps to suppress overadjustment of weights not involved in the error - "innocent bystanders" - as it were, and second it more rapidly adjusts the erroneous inputs towards a correct value, so that training is more effective in the early epochs. Again, we observed this in simulations, where sparsely connected networks very rapidly settled in error even in the initial epochs. (fig. 9). Both in the forward and backward passes it is reasonable to infer that the effect of partial connectivity

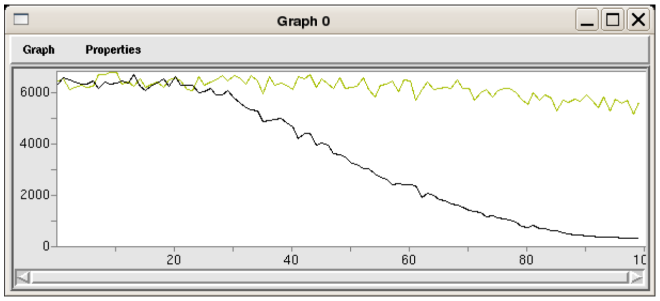


Fig. 9. Settling time during early training. X-axis is the weight update number, in 10's (thus the tick mark "20" is the 200th update) Y-axis is the global network error. The light-green trace is the fully connected network. The black trace is the best of the sparsely connected networks.

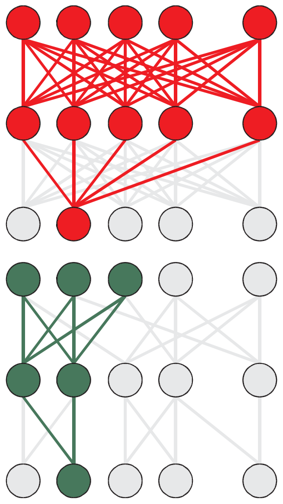


Fig. 10. Error propagation in fully- versus partially-connected networks. In the full case, the error propagates throughout the network from even a single output with inaccurate value. Errors are propagated under partial connectivity, but remain confined to specific paths where there exist connections, leaving large sections on the network unaffected.

is to localise signal propagation into class-specific groupings (fig. 10).

The optimum connectivity tends to be lower with smaller network size, and likewise beyond a critical point the performance of large networks with very sparse connectivity drops dramatically. These are expected results with a simple interpretation: at some point the number of connections and/or neurons is too small to represent the data accurately. Thus as the model removes more connections, the network must eliminate classes or features. If we adopt the model where the number of connections is fixed, and the number of neurons (and hence the mean connectivity per neuron) varies, it might be equally possible to represent the same data to the same degree of accuracy with 2 networks of different number of neurons but identical number of connections. Comparison of best-case performance for the 50-hidden and 200-hidden neuron large phonetic recognition networks supports this hypothesis. However, the smaller network will have a significant disadvantage: diminished fault tolerance and ability to represent new classes. In the limit, the smallest possible network will use every neuron and every synapse completely to represent the training data, thus if a single neuron or synapse fails there is some data loss. By contrast in the large-hidden-layers case, while neurons specialise due to their differing local connectivities, the loss of one does not imply permanent and total loss of a represented class or feature if another neuron otherwise uncommitted or only partially contributing to the class separation can specialise to replace the faulty one. One benefit of large networks therefore lies in robustness under component failure and ability to adapt to new, heretofore unknown inputs. It is clear from the data that the larger network performs better under a wide variety of parameter combinations and hence is the preferred choice in a system such as SpiNNaker where processing is cheap and connections expensive.

VIII. Conclusions

We have examined optimal patterns of connectivity in large MLP networks and seen that there is a clear relationship between network size and optimum connectivity measurable in terms of total number of connections. By examining simple networks it is also clear that the gain in the transfer function plays an important role, and where minimal error is desirable, should be relatively low. With decreasing connectivity, however, the impact of gain diminishes, an important property if strong class separation is desirable. While this is an important first step towards the development of objective formulæ for calculating connectivity and gain values in an MLP model, considerable work remains to quantify the relationship. The analysis of the data in this study is only preliminary; there is scope both for further analysis (particularly in the extraction of the most important parameters for the large phonetic network), and for richer exploration of the parameter space. An important future topic is the development of methods for automated parameter extraction from the raw input data: some means to normalise the data characteristics so that they can be used directly in the gain/connectivity calculations.

Using partial connectivity optimisations makes it possible to apply the MLP to dedicated hardware like SpiNNaker containing configurable but finite routing resources. The ability to reduce the connectivity significantly reduces overall network traffic, lessening potential congestion. That partial connectivity is optimum improves the hardware feasiblity of large neural networks, and suggests that the ideal architectural model for future neural hardware systems may be a configurable interconnect structure that assumes at most partial connectivity, such as SpiNNaker. The connectivity and gain values we have observed provide a useful point of departure for MLP implementation on SpiNNaker, or for that matter, other hardware systems. Ultimately it may be possible to provide a complete theoretical formula to compute required connectivity with a given hardware resource, network model, and application.

Little work has been done on formal methods to compute connectivity, gain, or for that matter any other parameter of neural networks, whose design remains partially empirical; it seems timely that this should change. The increasing interest in larger neural networks inevitably means that empirical methods cannot in any case continue: what is adequate for optimising small networks of tens or hundreds of neurons becomes completely unfeasible at the scale of tens of thousands or millions where fully automatic methods are essential. If nothing else, by viewing the problem from a hardware implementation perspective, this work might suggest a different way to conceptualise the design question: rather than specifying the number of *neurons*, specify the number of *connections* and fit the number of neurons to the optimal connectivity point. In the limit, interactions between parameters such as connectivity and gain may be basic to the neural model of computation, and if so the research could reveal fundamental properties of neural systems both biological and artificial.

Acknowledgements

The SpiNNaker project is supported by the Engineering and Physical Sciences Research Council, partly through the Advanced Processor Technologies Platform Partnership at the University of Manchester, and also by ARM and Silistix. Steve Furber holds a Royal Society-Wolfson Research Merit Award.

References

[1] J. C. Reijneveld, S. Ponten, H. Berendse, and C. J. Stam, “The application of graph theoretical analysis to complex networks in the brain,” *Clinical Neurophysiology*, vol. 118, no. 11, Nov. 2007.

[2] J. KrishnaKumar, “Optimization of the neural net connectivity pattern using a backpropagation algorithm,” *Neurocomputing*, vol. 5, no. 6, Nov. 1993.

[3] A. Rast, S. Yang, M. M. Khan, and S. Furber, “Virtual synaptic interconnect using an asynchronous network-on-chip,” in *Proc. 2008 Int’l Joint Conf. on Neural Networks (IJCNN2008)*, 2008.

[4] M. M. Khan, D. Lester, L. Plana, A. Rast, X. Jin, E. Painkras, and S. Furber, “SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor,” in *Proc. 2008 Int’l Joint Conf. on Neural Networks (IJCNN2008)*, 2008.

[5] D. J. Watts and S. Strogatz, “Collective dynamics of ‘small-world’ networks,” *Nature*, vol. 393, no. 6684, Jun. 1998.

[6] O. Shefi, I. Golding, R. Segev, E. Ben-Jacob, and A. Ayali, “Morphological characterization of in vitro neuronal networks,” *Phys. Review E*, vol. 66, no. 2, Aug. 2002.

[7] N. Davey, L. Calcroft, and R. Adams, “High capacity, small world associative memory models,” *Connection Science*, vol. 18, no. 3, Sep. 2006.

[8] J. J. Torres, M. A. Muñoz, J. Marro, and P. L. Garrido, “Influence of topology on the performance of a neural network,” *Neurocomputing*, vol. 58-60, no. 6, Nov. 2004.

[9] L. Calcroft, R. Adams, and N. Davey, “Efficient architectures for sparsely-connected high capacity associative memory models,” *Connection Science*, vol. 19, no. 2, Jun. 2007.

[10] D. Simard, L. Nadeau, and H. Kröger, “Fastest learning in small-world neural networks,” *Phys. Letters A*, vol. 336, no. 1, Jan. 2005.

[11] L. Maguire, T. M. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, “Challenges for large-scale implementations of spiking neural networks on FPGAs,” *Neurocomputing*, vol. 71, no. 1-3, Dec. 2007.

[12] D. L. T. Rohde, “LENS: The light, efficient network simulator,” *From http://tedlab.mit.edu/˜dr/Lens*, 1999.

[13] S. Welbourne and M. A. L. Ralph, “Using parallel distributed processing models to simulate phonological dyslexia: The key role of plasticity related recovery,” *J. Cognitive Neuroscience*, vol. 19, no. 7, Jul. 2007.