Slide 0

Lecture 2.3

MLP Design: Tricks of the Trade

Jonathan Shapiro

Department of Computer Science, University of Manchester

February 2, 2003

Slide 1

Development Process

1. Get data.
2. Choose input and output representation.
3. Partition data into training, validation, test sets.
4. Pre-process data (?).
5. Select network architecture or set of architectures. “How many hidden units do I use?”
6. Choose learning method and regularization method.
7. Train the network.
8. Validate the network.
9. Evaluate — Performance not good enough $\rightarrow$ return to step 3 or 4 or 5 or 1. Performance good enough $\rightarrow$ test on test set and use.

Slide 2

Here we will look at some of these steps in more detail.

Slide 3

Input and output representation

Help the network

- Make similarities in input encodings represent real similarities – no spurious similarities.
- Make the relationships between input and output as linear and monotonic as possible.

Examples:

Nettalk: 29 input characters encoded with 29 input nodes.
$a = 1000000\ldots, b = 0100000\ldots$, etc. *one-of-n* encoding. Makes the letters as dissimilar as possible. Outputs represented in terms of articulatory features. Made similar outputs encode similar sounding words.

Slide 4

Wind direction: (for weather prediction). If encoded as an angle, it is non-monotonically related to output, because 359 degrees seems very dissimilar from 1 degree, but both are essentially north. Encode as x-y coordinate, say.

Make the problem as linearly related to the input attibutes as possible.

Slide 5

Softmax — an output transfer function for 1-of-n encodings

- For an n class classification problem, 1-of-n encoding is often used for the output.
- An MLP with n sigmoidal outputs can produce outputs which are in more than one class, or are in none.
- Here is an alternative:
  - Let h^k be the weighted sum of input to the kth output node.
  - The output of node k, y_k is given by

$$y_k = \frac{\exp(h^k)}{\sum_j \exp(h^j)} \tag{1}$$

  - Note that all of the output node activities sum to 1. Thus, y_k can be interpreted as *the probability of the data being in class k*

Slide 6

Preprocessing the inputs

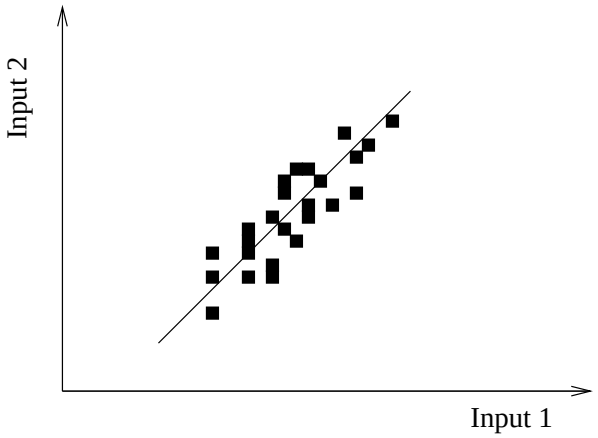
1. It usually improves learning speed to remove the mean of each input attribute.
2. Often useful to scale them to unit variance.
   - Remove arbitrary scale differences between input attributes (e.g. one is in millimeters, so has big numbers, another is in kilowatts, so has small numbers. But they are not comparable).
   - Don't remove scale differences if inputs are of same units and you know that one is more important than the other.
3. Consider removing linear correlations between inputs using Principal Component Analysis. Also used for reducing the dimension of the input space.

Slide 7

Principal Component Analysis — an aside

- Finds a linear combination of the inputs which contains most of the variation.
- First principal component is the direction containing most variation in the data.
- Second principal component is direction orthogonal to the first containing most of the variation in the data.
- Higher components defined similarly.
- The data can be transformed onto the space spanned by a small number of principal components, to have lower dimensional inputs.

Slide 8



Slide 9

Improving Learning

Batch learning versus incremental learning: Incremental learning is faster and less susceptible to local minima, but the most sophisticated search techniques require batch learning.

Symmetric inputs and transfer functions: Learning is usually faster if the inputs and transfer functions are negative as often as positive. Weight changes are proportional to the inputs to the weights. Best if that is not near 0 much of the time. So use symmetric sigmoid and remove mean of inputs.

Dealing with the stepsize problem: It can be impossible to find an appropriate stepsize or learning rate for simple gradient descent learning.

Momentum: Weight change = weight change due to gradient descent +

direction weights were already going.

$$\Delta w_i(t+1) = -s\text{Grad}E_i + \alpha\Delta w_i(t)$$

Introduces another parameter α which you have to set. ($0 < \alpha < 1$. 0.9 is a typical value.)

This is an old idea which is no longer used much.

Slide 10

Adaptive stepsize: Have the network tried to find appropriate stepsize in real time. Consider the error before and after the weight change.

- Weight change increases the error — reject the weight change and reduce the stepsize.
- Weight change decreases the error — accept the weight change and increase the stepsize.

Use better optimization algorithms

Many improvements over gradient descent have been developed.

Newton's method: If we know the curvature as well as the gradient, an appropriate size step can be taken. Curvature can be calculated, but this is usually too computationally costly in high dimension for practical use. Quasi-Newton methods update an approximation to this at each time.

Slide 11

Conjugate gradient methods: These do line searches in a series of conjugate directions. Very fast in many cases. Can only be used for batch learning.

Levenberg-Marquardt algorithm: A specialized algorithm for sum-squared errors optimization. Very fast. Has a stepsize parameter which is usually set adaptively.

Slide 12

Choosing the network architecture

1. Try to find the simplest network which stores the data.
2. Bracket the complexity: try a perceptron, try very simple MLPs. Try complex MLPs.
3. Without regularization, 4–10 training patterns per adjustable weight is often required to get good generalization.
4. Use a large network and regularize aggressively. (Regularization methods to be discussed shortly).

Slide 13

Model selection via search

The approach is to train a number of networks until the stopping criterion is met. Choose the network which is optimal on the validation set. Test that on the test set and use it.

1. Exhaustive search — Try a number of runs on a number of different number of hidden units. Select the best network on the validation set.

   e.g. 1 hidden unit on 10 runs, 2 hidden units on 10 runs, etc. Could use cross-validation on this.

Slide 14

2. Other search methods — Genetic algorithms and other non-exhaustive search methods have been used. (See Yao, 1995 –)
   (a) Generate a population of neural networks. Number of hidden units randomly chosen within prescribed ranges.
   (b) Partially train the networks using backpropagation until some stopping criterion met.
   (c) Rank the networks via performance on validation set.
   (d) Select the better networks.
   (e) Duplicate the better networks and mutate them by deleting a node, adding a node, etc.
   (f) Partially train the mutated networks.
   (g) Rank whole lot on validation set. Select the best to be the next population. Go back to step 2.

Slide 15

Improving Generalization

Early stopping: Highly recommended.

Weight-decay regularization: Suppose we start with a big network, but penalize the network for using complexity which is not required to store the training data.

Regularized Error: $E_r = E + \beta P$

where E is the usual sum-squared error, P penalizes complex networks, and β is an arbitrary parameter.

A common penalty term encourages unnecessary weights to be set to 0.

$P = \sum_i w_i^2$; Penalize large weights

Choosing an appropriate value for β can be difficult. There is a solution in the Bayesian framework.

Slide 16

Evaluation — What can go wrong

After training and validation, you find the network has inadequate performance on the validation set. What to do.

Two possible scenarios:

1. Network cannot train to desired performance level,
2. Network can train, but generalization performance is beneath desired level.

Slide 17

Network fails to train

The possible reasons and remedies are:

Cause — Network is unable to do the task.

Remedy — Try networks of greater complexity until you find some that do the task if possible.

Cause — Network can do the task; learning time was too short.

Remedy — Try training for longer. Try faster learning algorithms. More complex architectures may learn this task more quickly.

Cause — Data is too noisy or there is insufficient information in the input attributes.

Remedy — Noisy data may require more data, more informative input attributes, or a lowering of performance goals.

Slide 18

Diagnostics:

1. To determine whether or not the network can do the task, explore complex and simple architectures to bracket the complexity. Use validation to determine the best learning methods and stopping criteria.
2. To determine whether input noise is a problem requires some analysis of the input patterns,
   - How does a nearest neighbor classifier compare with a 3rd nearest neighbor classifier?
   - How does an linear interpolation perform as a function of the neighborhood size?

Slide 19

Network trains successfully, fails to generalize

Cause — Network can perform a task which does generalize well, but has not discovered that task.

Remedies — Use more agressive regularization methods, decrease network complexity, get more training examples.

Cause — Network cannot perform a generalizing task, but training set size was sufficiently small so that the apparent error is small (memorization regime).

Remedy — To recognize that this is the case, do as above. Then increase network complexity, while increasing amount of training data or regularization.

Slide 20

Diagnostics:

If you can get more training data, the two causes are distinguished by

1. in case 1, validation error will decrease,
2. in case 2, apparent error will increase.

It is hard to tell otherwise, but all you can do is increase the regularization which may cause the validation error to decrease in case 1 and increase in case 2.