Applying the Backpropogated Delta Rule (BP) to problems.

1: Nature of the problems.

BP has been applied to problems in many different domains.

For examples of some of these, see

http://www.emsl.pnl.gov:2080/docs/cie/neural/products/

We can consider two (related) types of problem:

Classification and Interpolation.

Classification.

Example:

Say there are many varieties of some plant, all similar, but non-identical. We have an expert's opinion on some set of these, and we have made a number of measurements of some parameters of each plant, for example, height, leaf length, stem diameter, leaf width, leaf colour.

We aim to produce a network which will classify new examples of these plants from these measurements.

This is an example of a classic type of classification problem.

Notes:
(i) The available "correct" data is limited.
(ii) We do not know if the measurements we have taken are relevant or not. (However, we can use *domain-specific knowledge* to select which measurements to take.)
(iii) We do not know how these measurements should be combined to produce a correct classification.

There are non-neural network techniques, particularly statistical techniques which can be used in these problems as well.

Interpolation.

Example:

In some industrial process, we aim to achieve the highest yield, whilst minimising the amount of noxious pollutants produced. The actual process has many parameters, for example temperature, pressure, concentration of each of the chemicals used in the process, air inflow settings....

Measurements of process yield and pollutants produced have been taken for a number of settings of the parameters. Taking measurements for new settings is costly - and often impossible without stopping production.

This is an example of a classic interpolation problem: we have measurements of results at a number of points in the input space, and we would like to form some sort of model of the process so that we can predict results from new parameter settings.

Notes:
(i) The training data is again limited. Perhaps even more so than last time.
(ii) We do not know much about the precise functional relation between the output and the input.

Terms: training sets, test sets, validation, generalisation, and related issues.

Before we go on it may be as well to consider exactly what we are trying to achieve:

in the classification problem?

* "appropriate" behaviour on new data.
* apparently correct classification of new data.

in the interpolation problem?

* "appropriate" values for new parameter settings

It is important to realise that producing correct values for the supplied data, but inappropriate values for any new data is completely useless. What is required is appropriate generalisation performance. We need to use these networks in such a way as to provide good generalisation performance.

In addition, we need to be able to *assess* the generalisation performance of the network before releasing it.

Training sets and test sets.

In both the classification network and the interpolation network, we were supplied with a finite set of (<input value>,<output value>) pairs.

In addition, we generally know something about the possible (that is, valid) range of input values for each input. Given R input units, we have some set S of possible inputs

$$S \subset \mathcal{R}^R$$

where $\mathcal{R}$ is the real numbers. Additionally, S is a closed bounded set, since the range of input values for each input will necessarily be bounded.

In addition, we have the supplied data set, I = {input values}.

Clearly,

$$I \subset S$$

and generally, I is only a very small part of S.

Our aim is for the network to behave appropriately over the whole of S: how can this best be achieved?

At first sight, it might seem best to simply train the network on the whole of the training set.

This may even be true:

But if we do this, we have no way of telling whether the network generalises appropriately or not. This is a serious problem: it is easy to produce a network which can produce the correct output for the supplied input, but be completely wrong for any other input.

This is because the problem is *ill-posed*: for a given set of input-output pairs, there are many possible generalisations. (This is less true when I is nearly all of S: but this is not normally the case in real-world problems.)

We need to try to

(i) attempt to make the network generalise appropriately
(ii) be able to assess whether the network *does* generalise appropriately.

If we use all the training information at our disposal in training the network, we cannot assess the network at all.

We therefore divide the training data into two parts, and use one part (the *training set*) for training the network, and the other part (the *test set*) for testing the generalisation performance of the network.

In this way, we lose some of the potential training information, but gain the possibility of knowing that we have some appriopriate generalisation performance. An additional possibility is to

(i) train up many networks on the training subset
(ii) assess the models produced by seeing how well they generalise
(iii) choose the best model.

Sometimes, some of the training data (a *validation set*) is held back, and used only in final model selection.