Practical Network Design and Implementation

Tony Robinson
Cambridge University Engineering Department,
Trumpington Street, Cambridge, England.
ajr@eng.cam.ac.uk

Presented at the Cambridge Neural Network Summer School
7 September 1992

1 Introduction

The aim of this paper is to put Artificial Neural Network techniques in perspective, discussing the practical issues of network design and implementation. Firstly, a comparison is made with other pattern matching techniques with an overview of the capabilities and complexity of these techniques. Some basic ground in multi-layer perceptrons is then covered so that a link may be made between these and Gaussian classifiers. In the process, some of the practical difficulties associated with gradient descent based training are covered, along with the popular remedies. Finally, the paper is concluded with a summary of the advantages of recurrent networks and a review of the architectures available.

2 A few rivals to artificial neural networks

If the problem is simple enough, there is no need to invoke a complex tool to solve it. For instance, if a linear function can perform the required mapping, then it is likely that standard linear techniques are more appropriate [1]. Similarly, if there are sufficient training examples to populate the input space to the resolution required, then averaging over radius of resolution will give an acceptable solution. If the input space can be partitioned with sufficient samples falling into each partition, then the output for each partition can be precompiled.

In many interesting real-world tasks, the form of the solution is not known, and there are insufficient examples to use simple partition and average techniques. Thus it is necessary to use a more complex form of smoothing in the input space. The technique of k nearest-neighbours does this very neatly by averaging over the k training patterns that are closest to the probe input. This has the advantage of fine resolution in the densely populated areas, whilst maintaining a sufficient number of data points to maintain the accuracy of the result. The search for the nearest neighbours can be made fast using kD trees, and it is possible to hard limit the size of the tree for the continuous learning case [2]. There can be a large difference in the variance of the input dimensions, in which case a weighted Euclidean distance metric is more appropriate for finding the nearest neighbours.

A related parametric technique models the probability density function of the data in the input space as a single or a set of Gaussians. The mean and covariance matrix for a single Gaussian is easy to compute, and for the case of multiple Gaussians there exists an efficient Estimation Maximisation algorithm for computing the set of mean and covariance matrices.

However, it can be argued that if high order correlations exist in the input data, then any technique that uses a simple distance metric in the input space will not be able to exploit this information. A sequence of transformations may be necessary, as can be provided by multi-layer perceptrons. However, as is shown in figure 1, this is only achieved at an increased cost in model generation. Multi-layer perceptrons are very flexible function approximators [3], but if the relevant simplifying assumptions hold, then there are a variety of faster and perhaps more accurate techniques available.

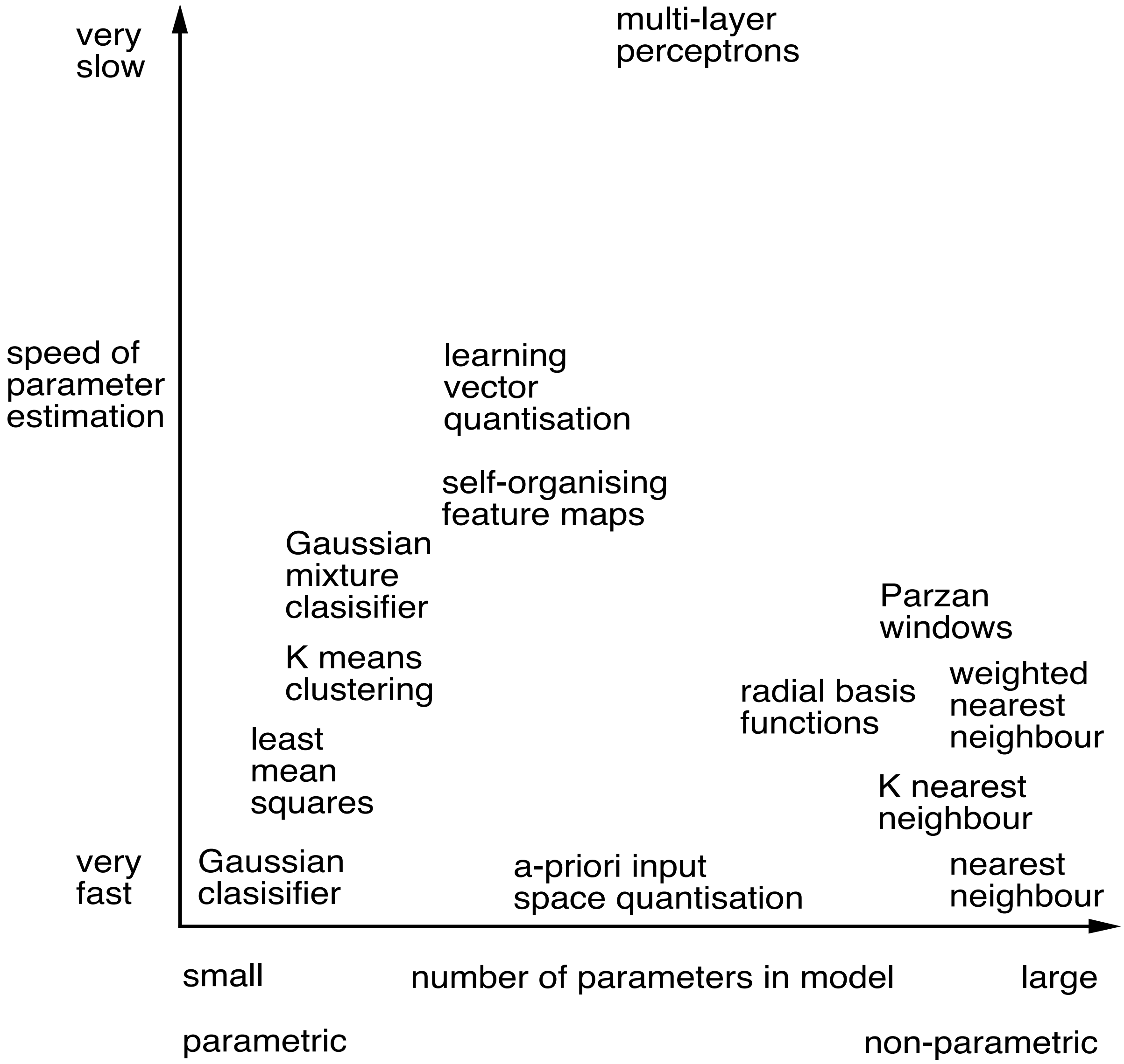

Figure 1: Pattern classification techniques

3 Connectionist Architectures

The field of artificial neural networks, or *connectionism*, covers a wide variety of techniques and models [4, 5, 6, 7, 8, 1, 9]. In general, artificial neural networks are used to model a task by adapting internal parameters. Due to the non-linear nature of many artificial neural networks, iterative techniques are often employed to perform the parameter adaptation, which can then be reasonably viewed as a "learning" process. Tasks are usually specified in terms of modelling a data set.

The models consist of a large number of simple *processing elements* (*computational neurons*, *nodes* or *units*) that in number can be used to generate non-linear mapping functions to do useful tasks. Each processing element computes a simple function of the other processing elements to which it is connected, and this result (activation) is made available to other units. The pattern of interconnection of the processing elements leads to a range of architectures:

- Strictly *feed-forward networks* are those where an ordering of the processing elements exists such that the output of one processing element is only dependent on the input of elements with a lower index. Thus the state of the network can be determined by computing the activations of the processing elements in order.

- If feedback is allowed, then the network becomes a dynamical system. Such *recurrent networks* can be formulated in terms of discrete time [10, 11] or continuous time [12]. The feedback may be either used to settle to a fixpoint for a given input, or many change as a function of the changing input.

These two architectures are shown in figure 2. Units are shown as circles, and connections between units as arrows. The architecture of the feed-forward network is one layer of *hidden units* and two layers of *weights*.

4 Training Paradigms

In addition to the use of feedback discussed earlier, classification of artificial neural network techniques can be done in terms of the types of task and the training procedures used. The task of composing a general mapping can be treated as non-linear function approximation, and it has been shown that a standard non-linear unit (weighted sum and sigmoid) can be used to construct any desired mapping [3]. However, there are number of interesting special cases:

4.1 Tasks:

- The *Associative memory* task is to recall the closest training pattern given a partial or corrupted input. As pointed out in [4], some care must be taken when using this style of computation, since if the distance measure is well defined, then the network can perform no better than simply searching the training set.

- The task of *Prediction* is to form an internal model of the process sufficient to generate the next point in a series. The fields of system identification and control have long been applied to linear models and are now being applied to non-linear systems [13, 14].

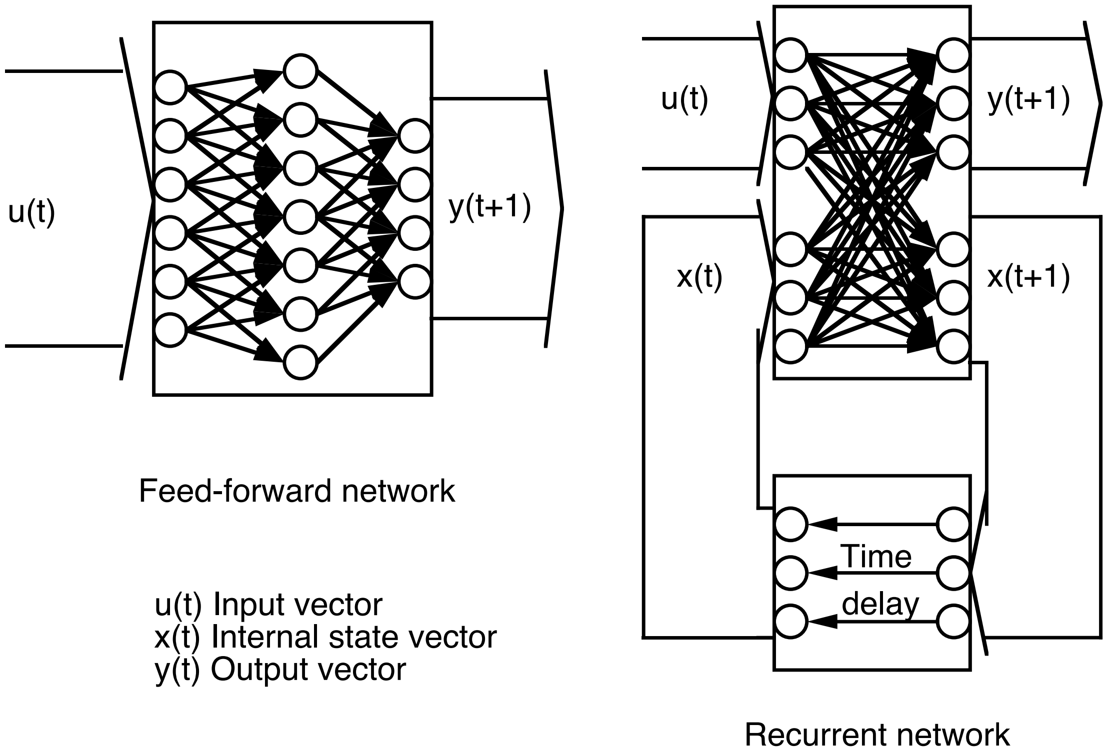

Figure 2: The Feedforward and Recurrent Error Propagation Networks

- Perhaps the most common task that artificial neural network models are applied to is that of *classification*. This is because many classification tasks require the construction of complex decision surfaces. Whilst classification merely requires a decision as to which class a given input most probably belongs, a powerful extension is to estimate the probability density function over all classes. Simple classification can be performed by picking the class with the maximum a-posteriori probability, but in addition the network may be used as a component in a larger classifier that incorporates other probabilistic evidence, such as a Markov model [15].

4.2 Training procedures:

- In the *supervised learning* task, the data set consists of typical input/output pairs. The aim here is to adapt the internal parameters to minimise some cost function dependent on the distance between the output of the network and the target given by the data.

- The *reinforcement learning* task is similar to that of supervised learning, but in this case only the distance between the actual and ideal outputs is available as a training signal. As there is much less information available, the task is correspondingly harder.

- The third type of task, *unsupervised learning*, is when there is no information at all about the ideal output for a given input. Instead, learning is guided by a general principle or format for the outputs, such as minimising the information loss through the model.

5 Back-propagation for multi-layer perceptrons

Back propagation [16, 17] is a gradient descent technique for cascades of non-linear but differentiable functions. It can be applied to learn the set of weights associated with an artificial neural network consisting of multiple layers of processing elements, as illustrated in figure 2. Such models are known as *Multi-Layer Perceptrons* (MLPs) and normally consist of weighted-sum nodes with sigmoidal activation functions. For a given set of inputs, x_j, and a set of weights, w_{ij}, the outputs of a standard feed-forward network are given by:

$$\mathrm{net}_i = \sum_{j=0}^{i-1} w_{ij} x_j \tag{1}$$

$$x_i = \frac{1}{1 + e^{-\mathrm{net}_i}} \tag{2}$$

The Euclidean distance measure is often used on the outputs (although see section 6.4):

$$E = \frac{1}{2} \sum_{\substack{\text{all}\\ \text{patterns}}} \sum_i (o_i - t_i)^2 \tag{3}$$

Differentiating this error measure with respect to the output of all units gives the *error signal*. This may be done directly for units in the output layer, and indirectly using the chain rule for all other units.

Differentiating the error measure with respect to the weights and summing over all patterns gives the direction of steepest descent.

Changing the weights by a small amount in the direction of steepest descent minimises the error measure and so adjusts the internal parameters so as to better model the target input/output pairs.

6 Practical Issues

In practice, several modifications have to be made to get reasonable learning performance from the back-propagation algorithm.

6.1 Gradient descent?

Perhaps the most fundamental problem is that strict gradient descent is not a practical learning algorithm in that it assumes infinitesimal changes in weight space on each iteration. More desirable is to make as large a change as possible on every iteration. A simple method, known as adding *momentum*, applies a first order filter to the gradient signal [17]. Thus changes in dimensions with oscillatory trajectories are damped in relation to changes in non-oscillatory dimensions. Under suitable conditions it is possible to optimise this weighting of past gradients so the next step is conjugate to the last [18]. Alternatively, a line search can be performed along the direction of steepest descent until a minima is found. Other methods iteratively adapt the weighting of past gradients so the next step is liley to be conjugate to the last [19]. However, perhaps the most sucessful methods keep a separate step size for every dimension in the weight space [20, 21]. The sign of succesive changes to a weight is used to estimate whether the change made is too great or too small, and so optimise the descent speed. An analysis of this technique is prestented in [22, 23].

6.2 Weight update policy

There are two extremes to the frequency at which the gradient information is used to update the weights. One end of this is that of true gradient descent, which requires that the partial gradients are summed over all pattern classes. The other end is to update the weights on the basis of one pattern alone. If the number of patterns is small and covers the whole of the pattern space, such the "exclusive or" toy problem, then summing over all patterns is a good technique. However, real-world tasks require a large amount of training data to specify a model accurately, but a much smaller amount to make a poorer approximation. Iterative learning techniques have to refine poor approximations, and during the initial training it is sufficient to only see part of the training set. This is taken to the extreme in *stochastic gradient descent* which maintains a per pattern update throughout the training. For linear systems there are schedules for decreasing the gradient weighting in order to ensure convergence. In the middle ground, it is possible to either gradually increase the number of patterns used to estimate the gradient over, or maintain per-pattern updating but increase the momentum smoothing factor to eventually average over the whole training set. Finally, it should be noted that strict gradient descent will find the first local minima, which may be not be a good solution. The use of stochastic gradient methods may add sufficient noise to escape shallow minima.

6.3 Number of parameters

The architecture and number of connections in a network are a critical part of the model design. Too many or too few layers and the required mapping may take too long to train, if at all. Most researchers using the basic architecture have settled on one hidden layer and two layers of adjustable weights. This leaves the size of the hidden layers, and hence the number of free parameters in the model. Too few, and the model won't even fit the training data, too many and overfitting will occur so that poor generalisation is seen. One easy way to limit the effects of overfitting is to keep back a proportion of the training data, and test on this data periodically. Training stops when performance ceases to increase on the *cross-validation set*. Some progress is being made by applying Bayesian methodology to model selection [24], and weight constraints [25].

6.4 Cost functions

It is very important to use the right cost function, or error measure when parameter fitting any model. In the case of artificial neural networks, this is closely related to a sensible choice of the activation function on the output units. Three simple cases may be identified:

- In the case where the model is known to have a strong linear component, then it is important to be able to model that. This means having linear output units and direct connections from the input units to the output units. Under the assumption of a Gaussian distribution of the predictor error, then minimising the mean squared error maximises the mutual information between the target values and the predicted values.

- If the outputs are to be interpreted as the estimation of independent probabilities, then the information difference between the target and actual output distributions is given by the cross entropy distance measure:

$$E = -\sum_i (t_i \log(x_i) + (1 - t_i)\log(1 - x_i)) \tag{4}$$

- Similarly, if outputs are to be taken as a probability density function over N classes, then the use of the *softmax* activation function [26] on the outputs ensures that the total sums to unity. A sensible distance measure is then to maximise the log likelihood of the occurrence of the target class

$$x_i = e^{\mathrm{net}_i} / \sum_j e^{\mathrm{net}_i} \tag{5}$$

$$E = -\sum_i t_i \log(t_i / x_i) \tag{6}$$

These are only examples, the general principle is to pick a network architecture and cost function that best fits the task to hand.

7 Sigmoidal Units and Gaussian Classifiers

There is obviously a great deal of similarity in the pattern classification methods presented in figure 1. For instance, the Kohonen self organising feature maps can be shown to give the same clustering as the K-means (or LBG) algorithm under certain conditions [27]. The use of Gaussians for data smoothing or data modelling is also present in many techniques. However, there is a fundamental split that needs to be addressed at this point. Dealing only with classification for simplicity, it can be seen that some techniques model the likelihood of an observation, x, given the class, c_i, i.e. $\Pr(x|c_i)$, whilst others model the posterior classification probability, $\Pr(c_i|x)$. The two are of course related by Bayes rule:

$$\Pr(c_i|x)\Pr(x) = \Pr(x|c_i)\Pr(c_i) \tag{7}$$

It is interesting to compute the posterior probabilities when the likelihoods of the data from each class can be modelled by a Gaussian with mean, μ_i, and class independent covariance matrix, Σ:

$$\Pr(x|c_i) = \frac{1}{(2\pi)^{n/2}\,|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_i)^T \Sigma^{-1} (x-\mu_i)\right) \tag{8}$$

given $\sum_i \Pr(c_i|x) = 1$, we have:

$$\begin{aligned}
\Pr(c_i|x) &= \frac{\Pr(x|c_i)\Pr(c_i)}{\sum_j \Pr(x|c_j)\Pr(c_i)} && (9)\\
&= \frac{\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_i)^T \Sigma^{-1} (x-\mu_i)\right)\Pr(c_i)}{\sum_j \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_j)^T \Sigma^{-1} (x-\mu_j)\right)\Pr(c_j)} && (10)
\end{aligned}$$

The computation of the Mahalanobis distance may be rewritten using the fact that Σ^{-1} is symmetric so $\mu_i^T\Sigma^{-1}x = x^T\Sigma^{-1}\mu_i$,

$$\begin{aligned}
(x-\mu_i)^T \Sigma^{-1} (x-\mu_i) &= x\Sigma^{-1}x - \mu_i^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_i + \mu_i^T\Sigma^{-1}\mu_i && (11)\\
&= x\Sigma^{-1}x - 2\mu_i^T\Sigma^{-1}x + \mu_i^T\Sigma^{-1}\mu_i && (12)
\end{aligned}$$

Now dropping the constant multiplier to the exponential, factoring out $\exp(x\Sigma^{-1}x)$ we have,

$$\begin{aligned}
\Pr(c_i|x) &= \frac{\exp\left(\mu_i^T\Sigma^{-1}x - \frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i\right)\Pr(c_i)}{\sum_j \exp\left(\mu_j^T\Sigma^{-1}x - \frac{1}{2}\mu_j^T\Sigma^{-1}\mu_j\right)\Pr(c_j)} && (13)\\
&= \frac{\exp\left(\mu_i^T\Sigma^{-1}x - \frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i + \log\left(\Pr(c_i)\right)\right)}{\sum_j \exp\left(\mu_j^T\Sigma^{-1}x - \frac{1}{2}\mu_j^T\Sigma^{-1}\mu_j + \log\left(\Pr(c_j)\right)\right)} && (14)
\end{aligned}$$

Finally we may define

$$\begin{aligned}
W_i &= \mu_i^T\Sigma^{-1} && (15)\\
\theta_i &= -\frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i + \log\left(\Pr(c_i)\right) && (16)\\
\mathrm{net}_i &= W_i x + \theta_i && (17)
\end{aligned}$$

to get the familiar softmax activation function:

$$\Pr(c_i|x) = \frac{\exp\left(\mathrm{net}_i\right)}{\sum_j \exp\left(\mathrm{net}_j\right)} \qquad (18)$$

or for the two class problem

$$\begin{aligned}
W_i' &= W_i - \sum_{i\neq j} W_j && (19)\\
\theta_i' &= \theta_i - \sum_{i\neq j} \theta_j && (20)\\
\mathrm{net}_i' &= W_i' x + \theta_i' && (21)
\end{aligned}$$

yielding the standard sigmoid activation function:

$$\Pr(c_i|x) = \frac{1}{1+\exp\left(-\mathrm{net'}_i\right)} \qquad (22)$$

From this we can conclude that a single layer perceptron with sigmoidal/softmax output units can correctly estimate posterior probabilities of classes with Gaussian distributions and identical covariance matrices. Furthermore, provided that the samples close to the class boundary are Gaussian, the estimate will be good. Sufficient units in the lower layers of a multi-layer perceptron could rotate the input dimensions such that the restriction of identical covariance matrices is valid, and perhaps massage the data to a Gaussian distribution.

8 Recurrent Networks

For problems with sequential input, it is important to exploit the known structure in order to make a good model of the system. The simplest method is to truncate the series, giving the *sliding window* approach. This may be sufficient if the information needed is known to decay rapidly outside the input window. However, this technique can only deal with finite context and equal weighting is given over the whole window. Increasing the context available rapidly increases the number of parameters to be estimated, which decreases the accuracy of the model.

For example, in the application of speech recognition, context information is important in speech recognition at all levels of representation. On short time-scales, co-articulation changes the pronunciation of phonemes due to physiological constraints on the rate of change of the articulators. On a longer time scale, we can expect the voice characteristics of the speaker to be constant, and so recognition should occur in the context of the estimated speaker characteristics. Hidden Markov models attempt to incorporate these context effects by considering first and second order differences of the acoustic vector. The same effect may be achieved by using many consecutive frames as input to a sliding-window multi-layer perceptron [28, 15], or the use of recurrent networks [11].

However, common computations need to be shared for accuracy of parameter estimation and for run-time efficiency. This may be achieved within the sliding window framework by tying the weights from every frame of input to the hidden layer. The same thing can be done for a range of hidden layer units, and this architecture is known as a time delay neural network [28]. The use of shared weights can be seen as incorporation of prior knowledge that subsections of the complete network should perform the same computations. A related way of incorporating prior knowledge into network design is to specify a range of values that the weights may adopt, or to specify the form of the probability density function [25].

A more efficient way of sharing computations is to employ feedback. This may be achieved in a number of ways, which can be divided into supervised and unsupervised methods for estimating the internal state. Historically, most connectionist research has concentrated on the supervised methods:

- Conceptually the simplest method is *Back-propagation through time* [17]. Here the network is expanded in time, and at the end of the sequence the target values are presented and the gradient signal calculated by back-propagating the error signal in time. This has the disadvantage that no learning takes place until the end of the sequence, and therefore requires some approximations to deal with non-terminating sequences. Nevertheless, this is still a very useful technique.

- To avoid the unrolling of the network in time, it is possible to limit the recurrence to *Output Feedback* [29]. However, this method can't deal with feeding back information that is not present in the output.

- An approximation to back-propagation through time can be made by truncating the error signal after one time step. If the hidden units of one time frame are presented to the network input at the next time frame, then some context information is fed forward in time. This network architecture is known as the *Simple Recurrent Network* [30]. However, the truncation of the gradient signal can lead to problems in convergence, and there is no gain in reduced computational requirements over back-propagation through time.

- A neat way of achieving local learning in time is to limit the feedback to a fraction of the past value of the unit. This scheme is known as *Focussed backpropagation* [31].
- Another approach is to place the burden of the feedback with the link connecting the units. This is normally taken to be a time-independent multiplicative function, but generalising this to a linear IIR filter gives recurrent behaviour. Such *Globally-Feedforward Locally-Recurrent Networks* [14] have the advantage that the dynamics can be analysed by linear systems theory.
- Yet another approach is to carry a set of partial derivatives forward in time from which the error signal can be calculated. This *Real-Time Recurrent Learning* [11, 32] has the appeal of a simpler structure than maintaining past inputs, although in practice the required computation is much larger.
- Finally, the *Recurrent Cascade-Correlation* [33] achieves fast sequential learning by detecting correlations in the input and state units which are likely to be exploitable in forming the output.

Unsupervised recurrent networks deserve a mention, for their potential if not for their current degree of development. Here the aim is to form a dynamical system such that the internal state retains the maximum amount of information from past inputs.

- In the linear case, this may be achieved with *Anti-Hebbian learning* [34, 35] which is an iterative technique to perform principle component analysis on the input. Under the appropriate assumptions, the information loss is minimised when the output spans the subspace of maximum variance.
- One approach to the non-linear case is to try to unpack the current state into the last input and last state. Such a *State Compression Network* [11] has been found useful for simple problems.
- An alternative to the non-linear case is to try use all but one state output to predict the remaining value. Non-linear decorrelation can be driven by the difference in these values, which forms the basis of *History Compression* [36].

Feedback in recurrent networks adds stability problems during training. Many of the approximations used to speed up the training of non-recurrent networks fail when the parameters to be adapted occur to high order powers in the gradient descent procedure. However, when used with care, these networks have the power to represent complex non-linear dynamical systems.

9 Other connectionist advantages

In a number of other conditions it is advantageous to use connectionist approaches:

- For a linear problem, if the number of input dimensions is large then it may no be possible to invert the covariance matrix due to numerical stablility problems. Iterative, gradient based techniques are more robust and may still converge.

- If an inverse model is required, then propagating derivatives though to the input can be used to track changes in the input [37, 38]
- Finally, there is the broad spectrum of non-engineering applications from the psyiological modelling of collections of neurons to the psychological modelling of learning in animals and humans.

10 Conclusion

Although artificial neural net techniques are obviously not the solution to every problem, the examples given show that they can be usefully applied to a range of speech processing problems. The technology is just a way of performing parameter estimation for non-linear models, and whilst blind application may give a low-manpower solution to a problem, forming a good model is normally more important. The iterative techniques involved mean that the models normally take longer to train, although there are potential benefits of large-scale parallelism given suitable hardware. In problems where the exact specification of a good solution is not known and a sensible neural net architecture can be found, these techniques can provide the best approach to obtaining a good solution.

References

[1] Teuvo Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, New York, second edition, 1988.

[2] Andrew W. Moore. *Efficient Memory-Based Learning for Robot Control.* PhD thesis, Cambridge University Computer Laboratory, 1990. Available as TR 209.

[3] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[4] Richard P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4(2):4–22, April 1987.

[5] Geoffrey E. Hinton. Connectionist learning procedures. Technical Report CMU-CS-87-115, Computer Science Department, Carnegie-Mellon University, June 1987.

[6] John Makhoul. Pattern recognition properties of neural networks. In *Neural Networks for Signal Processing - Proceedings of the IEEE workshop*, pages 173–187. IEEE, Piscataway, NJ, 1991.

[7] Y. H. Pao. *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley, 1989.

[8] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

[9] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. I: Foundations.* MIT Press, Cambridge, MA, 1986.

[10] Luis B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proceedings of the IEEE First Annual International Conference on Neural Networks*, pages II:609–618, San Diego, June 1987.

[11] A. J. Robinson. *Dynamic Error Propagation Networks*. PhD thesis, Cambridge University Engineering Department, February 1989.

[12] Barak A. Pearlmutter. Dynamic recurrent neural networks. Technical Report CMU-CS-90-196, School of Computer Science, Carnegie-Mellon University, December 1990.

[13] M. Niranjan and V. Kadirkamanathan. A nonlinear model for time series prediction and signal interpolation. In *Proc. ICASSP*, pages 1713–1716, 1991.

[14] Andrew D. Back. *New Techniques for Nonlinear System Identification: A Rapprochemenst between Neural Networks and Linear Systems*. PhD thesis, Department of Electrical Engineering, University of Queensland, 1992.

[15] Steve Renals, Nelson Morgan, and Hervé Bourlard. Probability estimation by feed-forward netowrks in continuous speech recognition. In *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing*, Princeton, NJ, October 1991.

[16] Paul J. Werbos. *Beyond Regression: New Tools for Predicition and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. I: Foundations.*, chapter 8. Bradford Books/MIT Press, Cambridge, MA, 1986.

[18] Willaim H. Press, Brian P. Flannery, Saul A. Teukolsky, and Willaim T. Vetterling. *Numerical Recipies in C: The art of Scientific Programming*. Cambridge University Press, Cambridge, England, 1988.

[19] L. W. Chan and F. Fallside. An adaptive training algorithm for back propagation networks. *Computer Speech and Language*, 2(3/4):205–218, 1987.

[20] Robert A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[21] Scott E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In *Proceedings of the 1988 Connectionist Models Summer School*. Morgan Kaufmann, 1988.

[22] Richard S. Sutton. Gain adaptation beats least squares? In *Proceedings of the* 7^{th} *Yale Workshop on Adaptive and Learning Systems*, pages 161–166, 1992.

[23] Richard S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *Proceedings of AAAI*, 1992.

[24] David J. C. MacKay. A practical bayesian framework for backprop networks. *Neural Computation*, 1991.

[25] Steven J. Nowlan and Geoffrey E. Hinton. Soft weight-sharing. In *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, 1991.

[26] John S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fougelman-Soulie and J. Hérault, editors, *Neuro-computing: Algorithms, Architectures and Applicatations*, pages 227–236. Springer-Verlag, 1989.

[27] Lizhong Wu and Frank Fallside. Fully vector quantized neural network-based code-excited non-linear predictive speech coding. Technical report, Cambridge University Engineering Department, March 1992. Submitted to IEEE Transactions on Signal Processing.

[28] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339, March 1989.

[29] M. I. Jordan. Serial order: A parallel distributed processing approach. ICS Report 8604, Institute for Cognitive Science, University of California, San Diego, May 1986.

[30] Jeffrey L. Elman. Finding structure in time. Technical Report CRL-8801, Center for Research in Language, UCSD, April 1988.

[31] M. C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, 3:349–381, 1989.

[32] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. ICS Report 8805, Institute for Cognitive Science, University of California, San Diego, October 1988.

[33] Scott E. Fahlman. The recurrent cascade-correlation architecture. Technical Report CMU-CS-91-100, School of Computer Science, Carnegie Mellon University, 1991.

[34] Peter Földiák. Adaptive network for optimal linear feature extraction. In *International Joint Conference on Neural Networks*, volume I, pages 401–405, Washington, D.C., 1989.

[35] Mark D. Plumbley. *On Information Theory and Unsupervised Neural Networks.* PhD thesis, Cambridge University Engineering Department, 1991. Available as CUED/F-INFENG/TR78.

[36] J. H. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4:234–242, 1992.

[37] A. Linden and J Kindermann. Inversion of multilayer nets. In *International Joint Conference on Neural Networks*, volume II, pages 425–430, Washington, 1989.

[38] Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. Occasional paper #40, Center for Cognitive Science, Massachusetts Institute of Technology, 1990.