Dynamic Error Propagation Networks

Anthony John Robinson
Trinity Hall
and
Cambridge University Engineering Department
ajr@dsl.eng.cam.ac.uk

Submitted February 1989
Examined June 1989

This dissertation was submitted for the degree of Doctor of Philosophy at Cambridge University.

Summary

This thesis extends the error propagation network to deal with time varying or dynamic patterns. Examples are given of supervised, reinforcement driven and unsupervised learning.

Chapter 1 presents an overview of connectionist models.

Chapter 2 introduces the error propagation algorithm for general node types.

Chapter 3 discusses the issue of data representation in connectionist models.

Chapter 4 describes the use of several types of networks applied to the problem of the recognition of steady state vowels from multiple speakers.

Chapter 5 extends the error propagation algorithm to deal with time varying input. Three possible architectures are explored which deal with learning sequences of known length and sequences of unknown and possibly indefinite length. Several simple examples are given.

Chapter 6 describes the use of two dynamic nets to form a speech coder. The popular method of Differential Pulse Code Modulation for speech coding employs two linear filters to encode and decode speech. By generalising these to non-linear filters, implemented as dynamic nets, a reduction in the noise imposed by a limited bandwidth channel is achieved.

Chapter 7 describes the application of a dynamic net to the recognition of a large subset of the phonemes of English from continuous speech. The dynamic net is found to give a higher recognition rate both in comparison with a fixed window net and with the established k nearest neighbour technique.

Chapter 8 describes a further development of dynamic nets which allows them to be trained by a reinforcement signal which expresses the correctness of the output of the net. Two possible architectures are given and an example of learning to play the game of noughts and crosses is presented.

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration.

Acknowledgements

The author would like to acknowledge financial support from the UK Science and Engineering Research Council, Cambridge University Engineering Department and Trinity Hall, Cambridge. Technical support was received from the UK Alvey Project MMI 069 which established the Hotel speech database and the ParSiFal project IKBS/146 which developed the transputer array.

The author would also like to thank:

- My supervisor, Professor Frank Fallside, for his unique blend of supervised, reinforcement driven and unsupervised teaching.
- Past and current members of the Cambridge University Engineering Department Speech and A.I. group, especially the connectionists: Ian Braithwaite, Lai-Wan Chan, Mark Plumbley, Mike Chong, Mahesan Niranjan, Patrick Gosling, Peter Todd, Phil Steen, Sean Kelly, Simon Duncan, Richard Prager, Tim Harrison, Tim Marsland, Tony Fotherby, Victor Abrash and Visakan Kadirkamanathen.
- The staff of the Photocopy Department for amazing service when deadlines have been tight and also the staff of the tea room for something like 10^3 toasted cheese buns.
- The following who all know why they are being thanked: Andrew E., Andrew M., Beryl, Chris, Don, Frances, George, Gordon, Matthew, Mary, Phil, Sue, Tamsin, Tim H., Tim N., Tim S., Tom, Wendy.

Contents

Chapter 0

Introduction

In the early 1960's the desire to model human intelligence was based on two opposite philosophies. The first was a 'holistic' approach which arose from a desire to model the computation strategy of the brain by using many simple processing elements. Each processing element uses information which is distributed over the whole of the store, and gives a result which is interpreted in the context of the results from all the other processing elements. The other was a 'reductionist' approach, using a single complex processing element which, acting on locally stored information, can be used to break down complex calculations into more tractable sub-problems. These simpler problems can in turn be reduced to problems which are simple enough to solve. Feldman (1986) provides a detailed discussion of these two approaches. As in modern science, medicine and economics, the reductionist approach has built on its own successes and now dominates the field.

Recently there has been a revival in the holistic approach in the form of 'connectionist models', 'neural nets' or 'parallel distributed processing', arising from the development of new learning algorithms and an increased speed of simulations on digital computers. This renewed interest has brought together neuroscientists, cognitive scientists, engineers, computer scientists, mathematicians and philosophers with considerable enthusiasm for a rapidly expanding subject.

0.1 Features of Connectionist Models

The connectionist approach to information processing is an alternative to the 'program and data' von Neumann model of computation which has found application from washing machines to launch-on-warning systems for nuclear weapons. This section makes a comparison of the two approaches. There are many good introductory texts in which which further information may be found, for example the two volumes of 'Parallel Distributed Processing', (Rumelhart and McClelland, 1986; McClelland and Rumelhart, 1986) and 'Self-Organization and Associative Memory' (Kohonen, 1988).

0.1.1 Complexity of Processing Elements

Perhaps the strongest difference between the conventional and the connectionist approach is in the representation of the processing and storage. Conventional computers store each piece of information, (for example an instruction, integer or 'symbol') in a store which is discrete and separate from the store of other information. A single processor with a rich instruction set accesses this data, and depending on its contents, it may access another location or store data in a location. In contrast a connectionist system has many simple processing elements which read the output of a large number of other processing elements, and performs some simple operation such as a weighted sum on these values, making the answer available to many other processing elements.

0.1.2 Distributed Representations

In contrast to the localised data storage of the von Neumann machines, connectionist models operate on a non-localised or distributed representation (Hinton, McClelland and Rumelhart, 1986). This applies both to the data from the input as it is transformed by the internal processing elements or 'hidden units', and also to the knowledge used to do this transformation. In practice, this has the advantage that the machines are robust to small variations in the parameters used to describe the machine, and also that faults in the processing elements result in a graceful degradation of performance.

0.1.3 Self Organisation

A common feature of all connectionist models is that they are defined using a set of examples, the training set, which makes some specification of the transformation to be performed. In many, but not all connectionist models, the parameters are determined by an iterative training procedure, starting from random values and converging onto a set which performs the mapping defined by the training set. This process of adaption is often referred to as 'learning'.

0.1.4 Inherent Parallelism

Many connectionist models have an inherently fine grained parallel structure such that the computations performed by the processing elements are independent and so may be computed in parallel. This is important when considering hardware implementations of connectionist models which can exploit this parallelism. However, all the connectionist models reported in this thesis ran as software simulations, and when parallel machines were used it was found to be more efficient to run a coarse grained parallelism which simulated each model sequentially (see appendix A).

0.1.5 Biological Inspiration

Analogies may be drawn between connectionist information processing and human information processing. Each processing element may be likened to a neuron and the strength of the links between elements to synaptic strength. Lindsay and Norman (1977) provide a good introduction to human information processing, Kuffler, Nicholls and Martin (1984) to neurobiology and many authors have linked these subjects to connectionist models (for example: Ferry, 1987). Whilst this aspect is of interest to the author (Robinson and Fallside, 1987a) this thesis adopts an 'engineering' approach which uses connectionist models for convenience and does not consider biological plausibility.

0.2 Uses of Connectionist Models

Existing computational techniques have had a considerable influence on the models of the mind used by psychologists and connectionist models are becoming popular in their turn. In contrast, the analogy between processing elements and neurons is almost certainly too simple to be taken seriously by neurobiologists.

Many comparisons between connectionist models and conventional techniques for pattern classification tasks have been made (for example: Huang and Lippmann, 1987). However, to date, there is no area where they are universally regarded as the best way to solve a given problem. The most promising areas of application are those in which the explicit means of solving the problem is not known and does not need to be known, those problems which require large computational resources which can use dedicated connectionist hardware, and those areas whose data structures can be expressed as distributed representations.

The fields of speech and vision processing fit these requirements. As vision problems tend to involve larger amounts of data than speech problems and connectionist models are computationally expensive to train, the major examples in this thesis are taken from the field of speech processing.

Chapter 1

Types of Connectionist Models

Connectionist models consist of many simple processing elements which are highly interconnected so as to collectively perform a complex computational task. The model is defined by a large number of parameters, or weights, whose values are selected so that the model performs a particular task out of all the possible tasks that could be performed. The aim of this chapter is to provide a brief review of the most popular connectionist models within a coherent framework, and so provide some context for the work that is presented in the remaining chapters. For more detailed reviews see Hinton (1987) and Lippmann (1987).

1.1 Learning Algorithms

Connectionist models learn by minimising a 'cost function' or 'energy' which measures how well the model fits a set of examples given in a training set. This cost function can be minimised in many ways, but as current models may use 10 to 100,000 weights, an exhaustive search through the weight space is usually not feasible. One popular approach is to start at a possible solution and iteratively make small changes to the values of the weights so as to improve the solution. The simplest algorithm for making these changes simply generates a random change and updates the weights if the cost function is reduced. A variation on this search maintains several likely points and generates new points by combining members of the current population with the addition of some noise. This is termed a the 'genetic algorithm' (Holland, 1975; Goldberg, 1989), named by analogy with the Darwinian process of natural selection. The search for a minimum can be quickened if the direction of steepest descent in weight space is known, and many algorithms are based on this procedure of 'gradient descent' (for example: Scales, 1985). However, if a strict gradient descent procedure is adopted there is no guarantee of finding the global minimum of the cost function. This deficiency may be overcome by using a stochastic algorithm which may make changes which increase the cost function and so escape from a local minima. If the influence of the random input is initially high and is sufficiently slowly reduced, then the global minimum is guaranteed. This processes is called simulated annealing (Kirkpatrick, Gelatt, Jr. and Vecchi, 1983).

For some problems an analytical solution may be found. For example an unsupervised network for performing dimensionality reduction may be solved using principle component analysis (Bourlard and Kamp, 1987). These methods have advantages over iterative methods as the final solution is a global minimum of the cost function and is often computed faster.

Connectionist models may be divided into those which make no distinction between the inputs and outputs of the networks, autoassociative networks, and those which do make this distinction, associative networks. Most of these networks can be described as a set of units and weights between these units. To provide a uniform framework for describing these models the notation o_{pi} will be used to represent the value of the activation of the $i^{\rm th}$ unit for the $p^{\rm th}$ example in a N unit network, θ_i the bias weight in the $i^{\rm th}$ unit and $w_{\rm ij}$ the value of the weight between the $i^{\rm th}$ and $j^{\rm th}$ units.

1.2 Autoassociative Networks

Perhaps the most popular form of autoassociative network is the Hopfield net (Hopfield, 1982). The units have a value of 0 or $+1$ and the weight matrix is formed in a non-iterative way by summing the outer product of all pairs of zero-mean training examples, as in equation 1.1. To run the network the elements in a corrupted version of a stored pattern are asynchronously updated according to equation 1.2.

$$w_{ij} = \sum_{j=0}^{N-1} (2o_{pi} - 1)(2o_{pj} - 1) \quad (1.1)$$

$$\begin{array}{l} o_{pi} \rightarrow 1 \\ o_{pi} \rightarrow 0 \end{array} \quad \text{if} \quad \sum_{j \neq i} w_{ij} o_{pj} \begin{array}{l} \geq 0 \\ < 0 \end{array} \quad (1.2)$$

This process is repeated until the pattern converges onto a stable state which, in many cases, will be one of the training patterns. In a later paper Hopfield (1984) extends this network to the case of continuously valued units.

Another powerful form of autoassociative network is the Boltzmann machine (Hinton, Sejnowski and Ackley, 1984; Ackley, Hinton and Sejnowski, 1985). These machines have both 'visible' units whose value is defined by the example pattern, and 'hidden' units whose value is a stochastic function of the weighed sum of its inputs. Equations 1.3 and 1.4 give the probability P_i that $o_{pi} = 1$, otherwise $o_{pi} = 0$. The 'temperature' term, T, is reduced during run time so that the value of P_i assumes the value 0 or 1.

$$\Delta E_i = \sum_{j=0}^{N-1} w_{ij} o_{pj} - \theta_i \quad (1.3)$$

$$P_i = \frac{1}{1 + e^{\Delta E_i / T}} \quad (1.4)$$

The network is trained by observing the probability that the $i^{\rm th}$ and $j^{\rm th}$ units are on at the same time when the visible units are clamped, ρ_{ij}, and the corresponding probability when they are not clamped, ρ'_{ij}. The relevant cost function, G, is known as the 'asymmetric divergence' or 'Kullback information' and its derivative is given in equation 1.5 which may be used to update the weights in the direction of steepest descent.

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T} \left(\rho_{ij} - \rho'_{ij} \right) \quad (1.5)$$

All autoassociative networks are a form of content addressable memory as part of the pattern can be used to recall the whole.

1.3 Associative Networks

The class of associative networks can be subdivided according to how the information in the output vector is used to train the network. In unsupervised networks the desired output vector is not explicitly specified but it is implicitly specified by some principle or heuristic. In reinforcement learning the network is trained using only a scalar which measures the closeness of the desired output vector to the actual output vector. Finally, in supervised learning every element of the output vector is directly compared with the desired output vector.

1.3.1 Unsupervised Learning

In unsupervised learning the true form of the desired output is unknown, instead a general principle is known which defines the form of the output. Hebb (1949) used a heuristic which strengthens a link when both the input and the output units are simultaneously active, as in equation 1.6.

$$\Delta w_{ij} \quad \propto \quad \sum_p o_{pi} o_{pj} \tag{1.6}$$

This can be formalised into a learning algorithm which minimises the information lost by the processing at each layer of units (Plumbley and Fallside, 1988). A network consisting of several layers of such units can extract interesting features from the input, for example Linsker (1986; 1988) demonstrates the emergence of orientation selective cells from visual input.

Supervised learning algorithms, such as the error propagation algorithm described below, can operate in an unsupervised mode by using the input vector as the desired output vector. If the network has a hidden layer with fewer units than the input then a dimensionality reduction must occur at this 'bottleneck'. Such an identity mappings using error propagation nets is used in chapter 6 (Robinson and Fallside, 1987a) and has also been used by Elman and Zipser (1987) for a speech coding problem and Harrison (1988) as a preprocessor for a speech recogniser. Hinton and McClelland (1987) have used a 'recirculation algorithm' to do this task without computing the cost function gradient explicitly, and Bourlard and Kamp (1987) have shown that the Karhunen-Loève transform using singular value decomposition can be used in the case of linear units.

1.3.2 Reinforcement Driven Learning

The weights in networks implementing reinforcement driven learning are adapted on a single scalar value which measures the closeness of the actual output from the desired output. This is an interesting problem from the biological and psychological view point as it is often assumed that learning in animals and man is driven by a reinforcement signal from the environment. As yet, reinforcement driven learning has been restricted to simple tasks when compared with those done by supervised or unsupervised learning.

Barto, Sutton and Anderson (1983) have used an 'associative search element' and an 'adaptive critic element' to solve a pole balancing task using reinforcement driven learning. The associative search element chooses an action with the highest predicted reinforcement for the given input vector, whilst the adaptive critic element learns the expected reinforcement. Sutton (1984) has also analysed a range of reinforcement learning algorithms and developed the 'adaptive heuristic critic' algorithm. Williams (1986) provides a mathematical analysis of these reinforcement driven learning procedures and describes a gradient descent technique for 'generalised stochastic learning automata'.

It is also possible to use supervised learning algorithms for reinforcement driven learning. One supervised net is used to compute the overall output from the input, and a second net takes both these outputs and inputs and is used to compute the expected reinforcement. Training proceeds by maximising the expected reinforcement. A description of this procedure with simple examples is given in chapter 8 (Robinson and Fallside, 1987a) and also by Munro (1987).

1.3.3 Supervised Learning

In supervised learning all the elements of the output vector are available to the training algorithm. The first supervised learning network was the perceptron formulated by Rosenblatt (1958). In this model the output units o_{pi} are a thresholded weighted sum of the binary input units o_{pj} as equation 1.7 and 1.8.

$$o_{pi} \quad = \quad f\left(\sum_j w_{ij} o_{pj} + \theta_i\right) \tag{1.7}$$

$$f(x_{pi}) \quad = \quad \begin{cases} 1 & \text{if } x_{pi} > 0 \\ 0 & \text{if } x_{pi} \leq 0 \end{cases} \tag{1.8}$$

The weight matrix and biases are changed if the calculated output is different from the target output, t_{pi}, as in equations 1.9 and 1.10.

$$w_{ij} \quad \leftarrow \quad w_{ij} + t_{pi} - o_{pi} \tag{1.9}$$

$$\theta_i \quad \leftarrow \quad \theta_i + t_{pi} - o_{pi} \tag{1.10}$$

This method can be generalised to the case of continuous valued input and output units to give an adaptive filter (Widrow and Hoff, 1960). Here the quantity to be minimised is the mean squared error between the actual and target outputs. As the cost function is quadratic in each of the parameters w_{ij}, a single minimum exists. At this minimum the derivative of the cost function with respect to each weight is zero, and this yields a set of simultaneous equations which can be solved using matrix inversion to yield the optimum weights in a non-iterative way (Rohwer, 1988).

Minsky and Papert (1969) showed that this single layer model was limited to learning linearly separable problems, which excluded some simple functions such as parity. This limitation may be overcome if two or more layers of weights are used. There are two approaches to determining the weights values in the two layers. One way is to implement an fixed non-linear mapping in the first layer, and then use the single layer approach outlined above. This method of dimensionality expansion has been used as the basis of many connectionist models (for example: Kanerva, 1984; Prager and Fallside, 1988; Powell, 1985; Broomhead and Lowe, 1988; Kohonen, 1988; Rayner and Lynch, 1988) and is discussed in section 3.1. The alternative is to extend the perceptron learning rule to cope with multilayer networks. This was accomplished by Rumelhart, Hinton and Williams (1985) and also by Parker (1982). The method, known as the 'error propagation algorithm', allows the training of arbitrary non-linear functions by gradient descent. This powerful and popular learning algorithm is the subject of the next chapter.

Chapter 2

Error Propagation Networks

This chapter defines the error propagation network used throughout the rest of this thesis. The formalism and notation used in this chapter is based on that of Rumelhart, Hinton and Williams (1986), but the network presented here is of a more general type in that the operation performed by each unit is not limited to that of a weighted sum. This is useful as it allows the error propagation algorithm to be used with those networks which define a volume in the input space, for example the Modified Kanerva Model (Prager and Fallside, 1988), networks of Spherical Graded Units (Hanson and Burr, 1987b), networks of Localised Receptive Fields (Moody and Darken, 1988) and the method of Radial Basis Functions (Powell, 1985; Broomhead and Lowe, 1988). Niranjan and Fallside (1988) give a description of these networks and make a comparison for pattern classification tasks. This also section gives examples of two different node types, the commonly used weighted sum node and a volume defining Gaussian node, and discusses two methods for changing the step size during training.

2.1 Formalism

An error propagation network is defined by a set of units and links between the units. Denoting o_{pi} as the value of the i^{th} unit for the p^{th} example in the training set, and w_{ij} as the weight of the link between o_{pi} and o_{pj}, we may divide up an array of units into input units, hidden units and output units. If we assign o_{p0} to a constant to form a bias, the values of the input units are defined by the problem and the values of the remaining units are defined by the node type. The most common node type is that which performs a weighted sum (otherwise known as a dot product or inner product) of its inputs and the weight vector:

$$x_{pi} = \sum_{j=0}^{i-1} w_{ij} o_{pj} \tag{2.1}$$

$$o_{pi} = f(x_{pi}) \tag{2.2}$$

In a more general node, the links, w_{ij}, are characterised by more than one scalar value and they can used in a different manner. The additional suffix, k, is used to denote an index into the weight vector, $\mathbf{w}_{ij}$, and the arbitrary differentiable function, $g(\cdot,\cdot)$, replaces the product term in equation 2.1:

$$x_{pi} = \sum_{j=0}^{i-1} g\left(\mathbf{w}_{ij}, o_{pj}\right) \tag{2.3}$$

The activation function, $f(\cdot)$, is any continuous non-linear function. These equations define a feed-forward net which has the maximum number of interconnections. This arrangement is commonly restricted to a layered structure in which units are only connected to the immediately preceding layer. The formalism, and most of the examples in this thesis, do not use this popular form. This decision was based on a minimal constraint design philosophy. The error propagation algorithm is not restricted to layered networks and if a layered structure is applicable to a particular problem then the gradient descent algorithm should set the interlayer links to zero (Robinson, 1986).

Throughout this thesis the input and output vectors will be drawn as broad arrows and error propagation nets as rectangular boxes. Thus figure 2.1 shows a static net which transforms the an input vector u_{pi}, to the output vector y_{pi}.

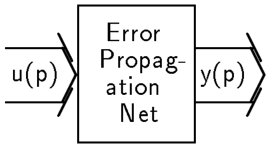

Fig. 2.1: Static net

2.2 Training by Gradient Descent

A network of N nodes may be trained by minimising an energy, E, defined as half the summed squared difference between the target output, t_{pi}, and the actual output of the net, o_{pi}, over all the training examples:

$$E = \frac{1}{2}\sum_p \sum_i (o_{pi} - t_{pi})^2 \tag{2.4}$$

This quantity can be minimised using gradient descent in which small changes in the weights are made in the direction of steepest descent, $-\partial E/\partial w_{ijk}$. The constant of proportionality, η, determines the step size and is sometimes called the 'learning rate'.

$$w_{ijk} \leftarrow w_{ijk} + \Delta w_{ijk} \tag{2.5}$$

$$\Delta w_{ijk} = -\eta \frac{\partial E}{\partial w_{ijk}} \tag{2.6}$$

$$\frac{\partial E}{\partial w_{ijk}} = \sum_p \frac{\partial E}{\partial o_{pi}} \frac{\partial o_{pi}}{\partial w_{ijk}} \tag{2.7}$$

For output nodes:

$$\frac{\partial E}{\partial o_{pi}} = o_{pi} - t_{pi} \tag{2.8}$$

For all other nodes we may apply the chain rule to yield:

$$\frac{\partial E}{\partial o_{pi}} = \sum_{j=i+1}^{N-1} \frac{\partial E}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial o_{pi}} \tag{2.9}$$

$$= \sum_{j=i+1}^{N-1} \frac{\partial E}{\partial o_{pj}} \frac{\partial}{\partial o_{pi}} f\left(\sum_{l=0}^{j-1} g\left(\mathbf{w}_{jl}, o_{pl}\right)\right) \tag{2.10}$$

$$= \sum_{j=i+1}^{N-1} \frac{\partial E}{\partial o_{pj}} \frac{\partial f(x_{pj})}{\partial x_{pj}} \frac{\partial g(\mathbf{w}_{ji}, o_{pi})}{\partial o_{pi}} \tag{2.11}$$

Lastly $\partial o_{pi}/\partial w_{ijk}$ is given by:

$$\frac{\partial o_{pi}}{\partial w_{ijk}} = \frac{\partial}{\partial w_{ijk}} f\left(\sum_{l=0}^{i-1} g\left(\mathbf{w}_{il}, o_{pl}\right)\right) \tag{2.12}$$

$$= \frac{\partial f(x_{pi})}{\partial x_{pi}} \frac{\partial g(\mathbf{w}_{ij}, o_{pj})}{\partial w_{ijk}} \tag{2.13}$$

These equations differ from those of Rumelhart Hinton and Williams only in the introduction of the terms in the differential of $g(\cdot,\cdot)$.

The above equations define the error signal, $\partial E/\partial o_{pi}$, for the input units as well as for the hidden units. Thus any number of static nets can be connected together, the values of $\partial E/\partial o_{pi}$ being passed from input units of one net to output units of the preceding net. It is this ability of error propagation nets to be 'glued' together in this way that enables the construction of the recurrent nets described in chapter 5.

2.3 Example of Weighted Sum Nodes

The common form of the error propagation algorithm has one scalar associated with each link, w_{ij0}.

$$g(\mathbf{w}_{ij}, o_{pj}) = w_{ij0} o_{pj} \tag{2.14}$$

$$f(x_{pi}) = \frac{1}{1 + e^{-x_{pi}}} \tag{2.15}$$

This node defines a hyperplane in the input space, the output of the node is determined by the distance from this plane. Using these equations and equations 2.11 and 2.13:

$$\frac{\partial E}{\partial o_{pi}} = \sum_{j=i+1}^{N-1} \frac{\partial E}{\partial o_{pj}} o_{pj} \left(1 - o_{pj}\right) w_{ji0} \tag{2.16}$$

$$\frac{\partial o_{pi}}{\partial w_{ij0}} = o_{pi} \left(1 - o_{pi}\right) o_{pj} \tag{2.17}$$

The activation function used in all the examples, unless otherwise specified, is the symmetric sigmoidal function:

$$f(x_{pi}) = \frac{2}{1 + e^{-2x_{pi}}} - 1 \tag{2.18}$$

2.4 Example of Gaussian Nodes

A Gaussian node can be defined as:

$$g(\mathbf{w}_{ij}, o_{pi}) = \left(\frac{w_{ij0} - o_{pi}}{w_{ij1}}\right)^2 \tag{2.19}$$

$$f(x_{pi}) = e^{-x_{pi}/2} \tag{2.20}$$

This node delimits a hyperellipsoidal region in the input space. The weights w_{ij0} represent the coordinates of the centres of the Gaussians and w_{ij1} represents the range of influence in each dimension. Using these equations and equations 2.11 and 2.13:

$$\frac{\partial E}{\partial o_{pi}} = \sum_{j=i+1}^{N-1} \frac{\partial E}{\partial o_{pj}} o_{pj} \frac{w_{ji0} - o_{pj}}{w_{ji1}^2} \tag{2.21}$$

$$\frac{\partial o_{pi}}{\partial w_{ij0}} = -o_{pi} \left(\frac{w_{ij0} - o_{pj}}{w_{ij1}^2}\right) \tag{2.22}$$

$$\frac{\partial o_{pi}}{\partial w_{ij1}} = o_{pi} \left(\frac{(w_{ij0} - o_{pj})^2}{w_{ij1}^3}\right) \tag{2.23}$$

which may be used with equations 2.7 and 2.8 to compute the direction of steepest descent. This type of node has been used by Chong, Fallside, Marsland and Prager (1989) as part of a real time, transputer based, speech recognition machine.

2.5 Adapting the Step Size Parameter

The error propagation algorithm only performs a true gradient descent providing the step size scaling factor, η, is sufficiently small. However, a practical implementation must choose a reasonably large value of η so that the learning takes place as fast as possible. One solution to this problem is to change η throughout the training. Two methods for this are used in this thesis, the first is the simplest and the most robust, whilst the second can give a better performance.

The simplest form of adaptive step size algorithm is taken from work done by Vogl, Mangis, Rigler, Zink and Alkon (1988). In this method the energy at current position in weight space, E^1 is compared with the energy at the last position, E^0. If the new energy is smaller than the last energy then the algorithm is moving towards a solution, and it is possible that a larger step size might be taken. Accordingly the step size is multiplied by a factor $\phi > 1$, so that larger step will be taken in future. Alternatively, if the new energy is larger than the previous energy then the previous step size was too large and η is multiplied by a positive factor, $\beta < 1$, so that a smaller step will be taken in future. The effectiveness of this method can be analysed if the energy surface is assumed to be locally symmetric about the minimum in the direction of steepest descent. In this case, a step which overshoots the minimum and slightly increases the energy is twice the optimal value, and halving the step size scaling factor would give the minimum at the next iteration. Thus although this method is simple it can easily over-estimate the optimal value.

$$\eta \leftarrow \begin{cases} \phi\,\eta & \text{if } E^1 \le E^0 \\ \beta\,\eta & \text{if } E^1 > E^0 \end{cases} \tag{2.24}$$

Chan and Fallside (1987b; 1987a) propose a somewhat more complicated method of adapting the step size scaling factor. Their algorithm has three parts, and only one of these parts has been adopted. This calculates the angle between the average movement in weight space and the current direction of steepest descent, θ. If $\theta < 90^o$ then the previous step did not reach the minimum in the direction of travel, so the step size should be increased by a small amount. Similarly, if $\theta > 90^o$ the previous step overshot the minimum and the step size should be reduced. Equation 2.25 achieves the necessary updating of θ. The term $\cos\theta$ can be calculated by dividing the dot product of the average change vector and the direction of steepest descent by the magnitudes of these vectors.

$$\eta \leftarrow \left(1 + \frac{1}{2}\cos\theta\right)\eta \tag{2.25}$$

Both sets of authors describe some form of back-tracking for the case when the previous energy was lower than the current energy. This technique is not applicable if the weights are updated more than once on a pass through the training set as only a noisy estimate of the energy is available. In the case where the weights are updated only once, it has been found that the direction of steepest descent is close to the direction of back-tracking, so satisfactory performance is achieved without back-tracking. As a result, this aspect their work has not been implemented in any of the networks presented in this thesis.

Chapter 3

Data Representation in Connectionist Models

Many connectionist models, such as a sufficiently large error propagation network, can perform any arbitrary mapping from a continuous input space to a continuous output space. In practice, the representation of the data is found to be important in determining the size of network required to perform the task, the time taken to train the network, and the performance of the network when presented with noisy input. For these reasons, this chapter explores the representation of data in connectionist models.

3.1 Input Representation

Many connectionist models employ a node which performs a non-linear function on the weighted sum of the inputs to the node. Such a node defines a hyperplane in the input space, other types of nodes define different types of regions, for instance hyperspheres (for example: Niranjan and Fallside, 1988). If every element of the output vector requires no more processing power than available by a single node, then the problem can be solved with a single layer model. To take the example of the commonly used hyperplane nodes, a single processing layer can solve any linearly separable problem. Computing the required weights for single layer models is considerably faster than for multilayer models both when using the iterative technique of gradient descent and when using the non-iterative techniques for linear systems involving matrix inversion (Rohwer, 1988; Kawahara and Irino, 1988).

There are many problems which are not linearly separable, the most common example is that of the 'exclusive or' function. In such cases it may be possible to use an intital layer of weights which map the input space into a higher dimensional space in which the problem is linearly separable and many techniques for doing this 'dimensionality expansion' exist. Rosenblatts perceptron (Rosenblatt, 1962) can be configured with an initial layer of linear threshold units with fixed weights. The Kanerva model (Kanerva, 1984) thresholds the Hamming distance of the binary input vector from a large number of random binary vectors and the modified Kanerva model (Prager and Fallside, 1988) generalises this to the case of continuous inputs. The method of radial basis functions calculates the distance of the input vector from each of the training vectors (Broomhead and Lowe, 1988). Finally, dimensionality expansion may be performed with the low order terms of the Kolmogorov-Gabor polynomial (Kohonen, 1988; Rayner and Lynch, 1988) or by using several units to coarse code each input dimension (Prager, Harrison and Fallside, 1986a; Hinton, McClelland and Rumelhart, 1986).

3.2 Internal Representation

Several investigators have attempted to assign meaning to the activations of individual units in an internal representation. Whilst this may prove fruitful in individual cases, the nature of distributed representations is such that a meaningful interpretation is unlikely to be apparent in large network without the use of statistical techniques such as cluster analysis (Servan-Schreiber, Cleeremans and McClelland, 1988). In the case of unsupervised learning the nature of the representation is governed by the principle of maximum information preservation (Plumbley and Fallside, 1988; Linsker, 1988) which in the linear case, can be reduced to principle component analysis (Bourlard and Kamp, 1987).

There are differing views on the best dimensionality for the internal representation. Some researchers (for example: Niranjan and Fallside, 1988) argue that the dimensionality of the internal representation should be chosen to be as small as possible whilst still being able to represent the training set. This is justified on the grounds that reducing the dimensionality of the internal representation reduces the number of free parameters in the model, so making the model more likely to generalise to data not in the training set. Other researchers (for example: Kanerva, 1984; Prager and Fallside, 1988) advocate an internal representation of as large dimensionality as is practically obtainable. This is justified because extra dimensions can represent additional features of the training set, and these extra features are expected to make a positive contribution to the performance of the network. In this thesis, chapter 4 presents a vowel recognition task in which extra dimensions consistantly give better recognition performance, whilst chapter 7 presents a more general phoneme recognition task which shows no consistant trend.

3.3 Output Representation

The output of a connectionist model is, by its nature, a distributed representation. For some problems this may be what is desired, for instance when trying to model some unknown continuous function, such as the prediction of a chaotic series (Lapedes and Farber, 1987). Often the Least Mean Squares metric is used as a measurement of the error, although other metrics may be used (Hanson and Burr, 1987a; Baum and Wilczek, 1987). However, many problems, such as pattern classification, require a symbolic output. This section deals with the representation of the set of symbols in the output vector.

It is reasonable to take the recognised class as that one whose target vector most closely matches the actual output vector. So, for the mean squared error metric, the recognised class, n, for the set of targets t_{ij} and outputs y_{pj} is:

$$n = \operatorname*{argmin}_i E_{pi} \tag{3.1}$$

$$E_{pi} = \frac{1}{2}\sum_j (t_{ij} - y_{pj})^2 \tag{3.2}$$

3.3.1 Single Scalar

Whilst it is possible to code N pattern classes onto a single scalar value, this is not usually advantageous. Equation 3.3 demonstrates that the i^{th} class can be linearly mapped to a target output is in the range 0 to 1.

$$t_{i0} = \frac{i + \frac{1}{2}}{N} \qquad 0 \le i \le N-1 \tag{3.3}$$

One example when this is useful is is given in chapter 6 which is a speech coding task where a real valued input is to be coded by one network to a symbolic form for transmission and then decoded to recover the original speech signal. However, without a-priori knowledge of the pattern classes there is unlikely to be any correlation between the difference of two pattern class indexes and the distance between the pattern classes. This coding scheme is as economical as possible in the number of output dimensions, but requires a considerable amount of processing to achieve this.

3.3.2 Packed Binary Vectors

If each element in the output vector is restricted to one of two values then N pattern classes can be represented in an output vector with $\log_2 N$ elements. This representation may be useful when there are a large number of pattern classes, but again it has the disadvantage that it requires considerable processing if nothing is known about the similarity of the pattern classes. Conversion from a vector where one element has a 'high' activation and the remainder have a 'low' activation to the binary representation requires a layer of weighted sum nodes with threshold activation functions. The representation is suitable if each element in the vector is known to represent a feature of the pattern class. However, this information reduces the problem to designing $\log_2 N$ networks which accept or reject a single pattern class, and the choice of features is likely to influence the performance.

3.3.3 One-of-Many Vectors

The most common form of output representation for a pattern class, i, is to set the i^{th} to a value designated as 'high', and the remainder to zero. So, for the target values 0 and 1:

$$t_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{3.4}$$

This form of representation is functionally equivalent to training N machines to accept or reject a pattern class, but with the advantage that computations performed by hidden units can be shared. If τ_1 is the target value of the 'high' unit and τ_0 the target value of the remainder of the units, then equation 3.2 can be written as:

$$E_{pi} = \frac{1}{2}\left((\tau_1 - y_{pi})^2 + \sum_{j \neq i}\left((\tau_0 - y_j)^2\right)\right) \tag{3.5}$$

$$= \frac{1}{2}\left((\tau_1 - y_{pi})^2 - (\tau_0 - y_{pi})^2 + \sum_{j}\left((\tau_0 - y_j)^2\right)\right) \tag{3.6}$$

$$= -(\tau_1 - \tau_0)y_{pi} + \frac{1}{2}\left(\tau_1^2 - \tau_0^2 + \sum_{j}\left((\tau_0 - y_j)^2\right)\right) \tag{3.7}$$

Equation 3.7 shows that there is a linear relationship between the actual output, y_{pi}, and the distance between the output vector and the corresponding target vector. Thus the largest element in the target vector corresponds to the least value of E_{pi} and therefore the best estimate of the pattern class.

3.4 Towards Symbolic Representations

The conversion from a distributed to symbolic representation has so far assumed that the index of the largest value can be selected from an array of values. Whilst this is a trivial task for a conventional program, it is difficult for a connectionist model. This section describes a network which moves the largest element close to 1 and the remainder close to 0, so reducing the task of choosing the largest element to one of thresholding. This is useful if a connectionist model is interacting with a symbol processing machine. For example, chapter 8 presents a connectionist model which learns the game of noughts and crosses by maximising a reinforcement signal. Such a net must give an output which is interpreted as a symbol specifying the position to be played, and must be able to interpret this output in order to be able to predict the outcome of the move. A possible solution has been presented by Lippmann (1987). This net subtracts the mean from every element in the vector then passes it through an activation function which sets any negative value to zero. This process is repeated until there is only one positive value. If the i^{th} element of the output vector at iteration q is y_{pqi} then:

$$y_{pq+1\ i} = f\left(y_{pqi} - \frac{1}{N}\sum_{j} y_{pqj}\right) \tag{3.8}$$

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{3.9}$$

This has two disadvantages if it is to be implemented as part of an error propagation network. Firstly the activation function has a region of zero slope which inhibits the process of calculating the partial derivatives of the energy function in previous networks. Secondly the value of the remaining positive unit is undefined, so further processing may be required. An alternative suggested by Boothroyd (1989: personal communication) is:

$$y_{pq+1\ i} = \frac{y_{pqi}^2}{\sum_j y_{pqj}^2} \tag{3.10}$$

Convergence to the limit values of 0 and 1 is found to be rapid. Figure 3.1 shows the development of two probability distributions of elements in a four dimensional vector. In each case the initial probability distribution was uniform in the specified range. The distributions were calculated by averaging the values obtained in 100,000 runs.

To analyse the convergence properties take y_{pqa} as the value of the largest output at the q^{th} iteration t and y_{pqb} as the value of the second largest, after one more iteration we have:

$$\frac{y_{pq+1\ a}}{y_{pq+1\ b}} = \left(\frac{y_{pqa}}{y_{pqb}}\right)^2 \tag{3.11}$$

A rough calculation of convergence time may be obtained by observing that the maximum value of y_{pqa} is 1.0 and threshold discrimination can be achieved if $y_{pqa} > 0.5$ and $y_{pqb} < 0.5$. The number of iterations required to achieve $y_{pqa} = 2y_{pqb}$ is $-\log_2 \log_2 y_{p0a}/y_{p0b}$.

For completeness, the equations needed to calculate the gradient of the energy through one layer of this net, as in chapter 2 are:

$$\frac{\partial y_{pq+1\ i}}{\partial y_{pqj}} = \begin{cases} 2y_{pqi}\left(1 - y_{pqi}^2/\sum_k y_{pqk}^2\right)/\sum_k y_{pqk}^2 & i = j \\ -2y_{pqj}\,y_{pqi}^2/\sum_k y_{pqk}^2 & i \neq j \end{cases} \tag{3.12}$$

$$= \begin{cases} 2y_{pq+1\ i}\,(1 - y_{pq+1\ i})/y_{pqi} & i = j \\ -2y_{pqj}\,y_{pq+1\ i} & i \neq j \end{cases} \tag{3.13}$$

3.5 Discussion

This chapter has discussed several aspects of data representation in connectionist models. Input representation may be improved by dimensionality expansion and the next chapter makes a comparison of some of the methods for a vowel recognition problem. The use of the least squared error has been proposed for choosing a symbolic target from a distributed output, and is used in chapter 7. This has been shown to be the same as picking the largest in a one-of-many representation. Finally, a new type of node has been presented to convert between distributed and symbolic representations.

Chapter 4

Application to Vowel Recognition

This chapter describes the application of a variety of feed-forward networks to the task of recognition of vowel sounds from multiple speakers. Single speaker vowel recognition studies by Renals and Rohwer (1989) show that feed-forward networks compare favourably with vector-quantised hidden Markov

Fig. 3.1: Development of Probability Distributions....

models. The vowel data used in this chapter was collected by Deterding (1988), who recorded examples of the eleven steady state vowels of English spoken by fifteen speakers for a speaker normalisation study. A range of node types are used, as described in the previous section, and some of the problems of the error propagation algorithm are discussed.

4.1 The Speech Data

The International Phonetic Association (I.P.A.) symbol and the word in which the eleven vowel sounds were recorded is given in table 4.1. The word was uttered once by each of the fifteen speakers. Four male and four female speakers were used to train the networks, and the other four male and three female speakers were used for testing the performance.

vowel	word	vowel	word
i:	heed	ɒ	hod
ɪ	hid	ɔ:	hoard
ɛ	head	ʊ	hood
æ	had	u:	who'd
ɑ:	hard	ɜ:	heard
ʌ	hud		

Table 4.1: Words used in Recording the Vowels

4.2 Front End Analysis

The speech signals were low pass filtered at 4.7kHz and then digitised to 12 bits with a 10kHz sampling rate. Twelfth order linear predictive analysis was carried out on six 512 sample Hamming windowed segments from the steady part of the vowel. The reflection coefficients were used to calculate 10 log area parameters, giving a 10 dimensional input space. For a general introduction to speech processing and an explanation of this technique see Rabiner and Schafer (1978).

Each speaker thus yielded six frames of speech from eleven vowels. This gave 528 frames from the eight speakers used to train the networks and 462 frames from the seven speakers used to test the networks.

4.3 Details of the Models

All the models had common structure of one layer of hidden units and two layers of weights. Some of the models used fixed weights in the first layer to perform a dimensionality expansion (see section 3.1), and the remainder modified the first layer of weights using the error propagation algorithm for general nodes described in chapter 2. In the second layer the hidden units were mapped onto the outputs using the conventional weighted-sum type nodes with a linear activation function. When Gaussian nodes were used the range of influence of the nodes, w_{ij1} was set to the standard deviation of the training data for the appropriate input dimension. If the locations of these nodes, w_{ij0}, are placed randomly, then the model behaves like a continuous version of the modified Kanerva model (Prager and Fallside, 1988). If the locations are placed at the points defined by the input examples then the model implements a radial basis function (Broomhead and Lowe, 1988). The first layer of weights remains constant in both of these models, but can be also trained using the equations of section 2.4. Replacing the Gaussian nodes with the conventional type gives a multilayer perceptron and replacing them with conventional nodes with the activation function $f(x) = x^2$ gives a network of square nodes. Finally, dispensing with the first layer altogether yields a single layer perceptron.

The scaling factor between gradient of the energy and the change made to the weights (the 'learning rate', η) was dynamically varied during training, as described in section 2.5. If the energy decreased this factor was increased by 5%, if it increased the factor was halved. The networks changed the weights in the direction of steepest descent which is susceptible to finding a local minimum. A 'momentum' term (Rumelhart, Hinton and Williams, 1986) is often used with error propagation networks to smooth the weight changes and 'ride over' small local minima. However, the optimal value of this term is likely to be problem dependent, and in order to provide a uniform framework, this additional term was not used.

4.4 Recognition Results

This experiment was originally carried out with only two frames of data from each word (Robinson, Niranjan and Fallside, 1988). In the earlier experiment some problems were encountered with a phenomena termed 'overtraining' whereby the recognition rate on the test data peaks part way through training then decays significantly. The recognition rates for the six frames per word case are given in table 4.2 and are generally higher and show less

Classifier	no. of hidden units	no. correct	percent correct
Single-layer perceptron	-	154	33
Multi-layer perceptron	88	234	51
Multi-layer perceptron	22	206	45
Multi-layer perceptron	11	203	44
Modified Kanerva Model	528	231	50
Modified Kanerva Model	88	197	43
Radial Basis Function	528	247	53
Radial Basis Function	88	220	48
Gaussian node network	528	252	55
Gaussian node network	88	247	53
Gaussian node network	22	250	54
Gaussian node network	11	211	47
Square node network	88	253	55
Square node network	22	236	51
Square node network	11	217	50
Nearest neighbour	-	260	56

Table 4.2: Vowel classification with different non-linear classifiers.

variability than the previously presented results. However, the recognition rate on the test set still displays large fluctuations during training, as shown by the plots in figure 4.1. Some fluctuations will arise from the fact that the minimum in weight space for the training set will not be coincident with the minima for the test set. Thus, half the possible trajectories during learning will approach the test set minimum and then move away from it again on the way to the training set minima (Plumbley, 1988: personal communiation). In addition, continued training sharpens the class boundaries which makes the energy insensitive to the class boundary position (Niranjan, 1988: personal commu-

Fig. 4.1: Performance on test set during training

niation). For example, there are a large number planes defined with threshold units which will separate two points in the input space, but only one least squares solution for the case of linear units.

4.5 Discussion

From these vowel classification results it can be seen that minimising the least mean square error over a training set does not guarantee good generalisation to the test set. The best results were achieved with nearest neighbour analysis which classifies an item as the class of the closest example in the training set measured using the Euclidean distance. It is expected that the problem of overtraining could be overcome by using a larger training set taking data from more speakers. The performance of the Gaussian and square node network was generally better than that of the multilayer perceptron. In other speech recognition problems which attempt to classify single frames of speech, such as those described by McCulloch and Ainsworth (1988) and that of chapter 7 (Robinson and Fallside, 1988), the nearest neighbour algorithm does not perform as well as a multilayer perceptron. It would be interesting to investigate this difference and apply a network of Gaussian or square nodes to these problems.

The initial weights to the hidden units in the Gaussian network can be given a physical interpretation in terms of matching to a template for a set of features. This gives an advantage both in shortening the training time and also because the network starts at a point in weight space near a likely solution, which avoids some possible local minima which represent poor solutions.

The results of the experiments with Gaussian and square nodes are promising. However, it has not been the aim of this chapter to show that a particular type of node is necessarily 'better' for error propagation networks than the weighted sum node, but that the error propagation algorithm can be applied successfully to many different types of node.

Chapter 5

Dynamic Error Propagation Networks

Many problems involve time varying or dynamic patterns and it is natural to reflect this by sequential information processing. This chapter begins by reviewing the current approaches to temporal pattern processing with error propagation networks, then develops three forms of network to deal with dynamic patterns, and gives some simple examples.

5.1 Temporal Pattern Processing Networks

There are two approaches to processing inputs in the time domain: either window the inputs and then treat the time domain like another spatial domain; or use some internal storage to maintain a current state. Most of the work done in this area has concentrated on modifying the error propagation algorithm to deal with time domain inputs. To some extent it is possible to sidestep the temporal processing problem by using a standard networks and applying a filter to the inputs (for example: Stornetta, Hogg and Huberman, 1987), or to the outputs (for example: Harrison, 1988)

The essential quality of a 'dynamic net' is is that its behaviour is determined both by the external input to the net, and also by its own internal state, which is represented by the activation of a group of units. These units form part of the output of a net and also part of the input to another copy of the same net in the next time period. Thus the state units link multiple copies of a net over time to form a dynamic net.

There are two uses for these time domain networks. Firstly they can be used as a 'relaxation network' in which a static pattern is presented at the input and the network is allowed to settle into a stable state, for example the Hopfield net. Rohwer and Forest (1987), Almeida (1987) and Rohwer and Renals (1988) have demonstrated that the error propagation algorithm may be used in such networks. Relaxation networks may have advantages over the strictly feed-forward networks in that complex calculations may be carried out with fewer weights.

The second use of time domain networks is to map a time varying input to a time varying output, which reduces to the previous case when the input is stationary. The remainder of this section makes a comparison of networks which window the input stream, and those which store an internal state.

5.1.1 Windowed Input Networks

Perhaps the most well known example of a large connectionist network is that of NetTalk by Sejnowski and Rosenberg (1986) In this network a fixed length window is placed over a character stream from English text, and the network is trained to output the phoneme corresponding to the letter under the centre of the window. This is perhaps the most straightforward way to incorporate some context, and it has also been popular in the field of speech recognition (for example: Robinson, 1986; Bourlard and Wellekens, 1987a).

An interesting member of this family of networks, the 'Time Delay Neural Network' has recently been developed by Waibel, Hanazawa, Hinton, Shikano and Lang (1987). The new feature of this net is that it uses duplicated weights within its structure to take advantage of the fact that much of the initial processing on each frame is likely to be common to all the frames under the window. This duplication of weights reduces the overall computation needed to process continuous input, and may lead to better generalisation as there are fewer free parameters.

All fixed context networks of this type suffer from two major disadvantages. Firstly, the model has a hard limit as to the amount of temporal context that can be processed, so that anything outside the window can not influence the output. Secondly these models have no inbuilt mechanism for dealing with variations in the rate of input. This is especially important in the field of speech recognition where there are often large variations in the rate at which words are spoken. To overcome these limitations networks with an internal state were developed.

5.1.2 Internal State Networks

Two types of 'recurrent' or 'dynamic' error propagation networks may be defined, fully recurrent networks and partially recurrent networks. In each of these networks the context information is held in an internal state and thus have certain similarities with Hidden Markov Models (Bourlard and Wellekens, 1987b).

Fully Recurrent Networks

The first recurrent error propagation network was described by Rumelhart, Hinton and Williams in their original paper (Rumelhart, Hinton and Williams, 1986), but until recently this aspect of their work received little attention. The structure is simple, as well as having weights to units at the current time, units also have weights to units in the previous time period. In the past there has been a misconception that these recurrent networks are computationally much harder to train than the time-independent, or static networks. This is false as a buffer may be used to store the activations of units which means that these networks can be trained with the same order of computation as a static network, (see section 5.3.1). These networks are used in chapter 6 (Robinson and Fallside, 1987b) for speech coding and by Watrous and Shastri (1987), Watrous, Shastri and Waibel (1987), and in chapter 7 (Robinson and Fallside, 1988) for phoneme recognition.

It is possible to train a fully recurrent network without storing the activations of the units for all the training set. A description of this procedure can be found in section 5.3.2 (Robinson and Fallside, 1987a) with an expanded account and examples in Williams and Zipser (1988).

Partially Recurrent Networks

A cut down version of the fully recurrent network is popular. Instead of feeding back the error signal as far as required, the error signal is terminated after a single pass. Some links may be fixed, such as the 'state and plan' networks of Jordan (1986). To compensate for the fact that these nets can no longer be trained using true gradient descent the architecture is chosen to force some information to be fed forward in time. Commonly a three layer network is used where the activations of the middle layer of hidden units are used as inputs to the next network. Section 5.3.3 presents a network which is a superset of the Jordan network. Section 5.4 gives some simple examples of the use of these networks, another example of learning simple grammars using a similar net is given by Servan-Schreiber, Cleeremans and McClelland (Servan-Schreiber, Cleeremans and McClelland, 1988).

5.2 Development from Linear Control Theory

The analogy of a dynamic net in linear systems (for example: Jacobs, 1974) may be stated as:

$$x_{p+1} = Ax_p + Bu_p \tag{5.1}$$
$$y_p = Cx_p \tag{5.2}$$

where u_p is the input vector, x_p the state vector, and y_p the output vector at the integer time p. A, B and C are matrices.

The structure of the linear systems solution may be implemented as a non-linear dynamic net by substituting the matrices A, B and C by static nets, represented by the non-linear functions $A[\cdot]$, $B[\cdot]$ and $C[\cdot]$. The summation operation of Ax_p and Bu_p could be achieved using a net with one node for each element in x_p and u_p and with unity weights from the two inputs to the identity activation function $f(x_{pi}) = x_{pi}$. Alternatively this net can be incorporated into the A[.] net giving the architecture of figure 5.1. The input is coded by net $B[\cdot]$ and then the output is fed into $A[\cdot]$ along with the previous output of $A[\cdot]$ and the

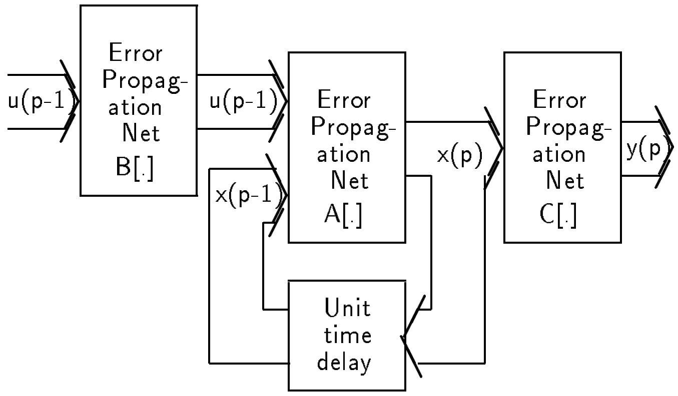

Fig. 5.1: Dynamic Net

resulting output is passed through $C[\cdot]$ to yield the overall output of the system. The three networks of the previous dynamic net architecture may be combined into one, as in figure 5.2. Simplicity of architecture is not just an aesthetic consideration. If three nets are used then each one must have enough computational power for its part of the task, combining the nets means that only the combined power must be sufficient and it allows common computations to be shared.

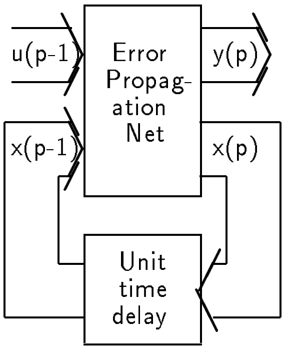

Fig. 5.2: Simplified Dynamic Net

The error signal for the output y_{p+1}, can be calculated by comparison with the desired output. However, the error signal for the state units, x_p, is only given by the net at time $p+1$, which is not known at time p. Thus it is impossible to use a simple backward pass to train this net. It is this difficulty which introduces the variation in the architectures of dynamic nets.

5.3 Architectures

This section presents three alternative architectures for dynamic nets, starting with the finite input duration dynamic net which is based on the recurrent net (Rumelhart, Hinton and Williams, 1986). This network relies on a buffer to store past activities of units, no other information on past context is available so the buffer length must be at least as long as the length of the context to be learned. The other two architectures have been formulated so that no such buffer is necessary.

5.3.1 The Finite Input Duration (FID) Dynamic Net

If the output of a dynamic net, y_p, is dependent on a finite number of previous inputs, u_{p-P} to u_p, or if this assumption is a good approximation, then it is possible to formulate the learning algorithm by expansion of the dynamic net for a finite time, as in figure 5.3.

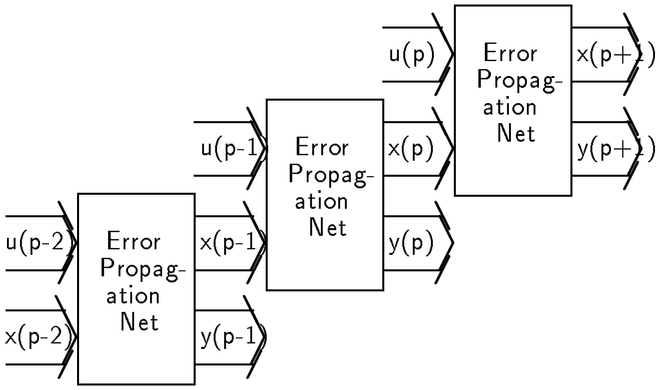

Fig. 5.3: Expanded Dynamic Net

Consider only the component of the error signal in past instantiations of the nets which is the result of the error signal at time p. The error signal for y_p is calculated from the target output and the error signal for x_p is zero. This combined error signal is propagated back though the dynamic net at p to yield the error signals for u_p and x_p. Similarly these error signals can then be propagated back through the net at $p-1$, and so on for all relevant inputs. The summed error signal is then used to change the weights as for a static net.

Formalising the FID dynamic net for a general time q, $q \le p$:

o_{qi}	is the output value of unit i at time q
t_{qi}	is the target value of unit i at time q
δ_{qi}	is the error value of unit i at time q
w_{ij}	is the weight between o_i and o_j
Δw_{qij}	is the weight change for this iteration at time q
Δw_{ij}	is the total weight change for this iteration

The values of o_{qi}, δ_{qj} and Δw_{ij} are calculated in the same way as in a static net. Here and in the remainder of this section it is assumed that weighted sum type nodes are being used.

$$x_{qi} = \sum_{j=0}^{i-1} w_{ij} o_{qj} \tag{5.3}$$

$$o_{qi} = f(x_{qi}) \tag{5.4}$$

$$\delta_{qi} = \begin{cases} f'(x_{qi})(t_{qi} - o_{qi}) & \text{for output units} \\ f'(x_{qi}) \sum_{j=i+1}^{N} \delta_{qj} w_{ji} & \text{for hidden units} \end{cases} \tag{5.5}$$

$$\Delta w_{qij} = \eta \delta_{qi} o_{qj} \tag{5.6}$$

The total weight change is given by the summation of the partial weight changes for all previous times.

$$\Delta w_{ij} = \sum_{q=p-P}^{p} \Delta w_{qij} \tag{5.7}$$

$$= \sum_{q=p-P}^{p} \eta \delta_{qi} o_{qj} \tag{5.8}$$

Thus, it is possible to train a dynamic net to incorporate the information from any time period of finite length, and so learn any function which has a finite impulse response.

In some situations the approximation to a finite length may not be valid, or the storage and computational requirements of such a net may not be feasible. For example, if the information in a signal decayed exponentially then only a finite storage is required to represent the state of the signal but all this information can not be obtained by observing the signal for a finite time. In such situations another approach is possible, the infinite input duration dynamic net.

5.3.2 The Infinite Input Duration (IID) Dynamic Net

Although the forward pass of the FID net of the previous chapter is a non-linear process, the backward pass computes the effect of small variations on the forward pass, and so it is a linear process. Thus the recursive learning procedure described in the previous chapter may be compressed into a single operation. The proof outlined below was first presented by Robinson and Fallside (1987a) and later by Williams and Zipser (1988).

Given the target values for the output of the net at time p, equation 5.5 defines values of δ_{pi} at the outputs. If we denote this set of δ_{pi} by D_p then equation 5.5 also states that any δ_{pi} in the net at time p is simply a linear transformation of D_p. Writing the transformation matrix as S:

$$\delta_{pi} = S_{pi} D_p \tag{5.9}$$

In particular the set of δ_{pi} which is to be fed back into the network at time $p-1$ is also a linear transformation of D_p

$$D_{p-1} = T_p D_p \tag{5.10}$$

or for an arbitrary time q:

$$D_q = \left(\prod_{r=q+1}^{p} T_r \right) D_p \tag{5.11}$$

so substituting this into equation 5.8:

$$\Delta w_{ij} = \eta \sum_{q=-\infty}^{p} S_{qi} D_q o_{qj} \tag{5.12}$$

$$= \eta \sum_{q=-\infty}^{p} S_{qi} \left(\prod_{r=q+1}^{p} T_r \right) D_p o_{qj} \tag{5.13}$$

which can be rewritten as:

$$\Delta w_{ij} = \eta M_{pij} D_p \tag{5.14}$$

where:

$$M_{pij} = \sum_{q=-\infty}^{p} S_{qi} \left(\prod_{r=q+1}^{p} T_r \right) o_{qj} \tag{5.15}$$

and note that M_{pij} can be written in terms of $M_{p-1\ ij}$:

$$M_{pij} = S_{pi} \left(\prod_{r=p+1}^{p} T_r \right) o_{pj} + \sum_{q=-\infty}^{p-1} S_{qi} \left(\prod_{r=q+1}^{p} T_r \right) o_{qj} \tag{5.16}$$

$$= S_{pi} o_{pj} + \left(\sum_{q=-\infty}^{p-1} o_{qj} S_{qi} \left(\prod_{r=q+1}^{p-1} T_r \right) \right) T_p \tag{5.17}$$

$$= S_{pi} o_{pj} + M_{p-1\ ij} T_p \tag{5.18}$$

Hence we can calculate the weight changes for an infinite recursion using only the finite matrix M.

If this approach is to be of practical use, we must consider the overall storage and computation requirements as compared with the FID net. The net has N units of which n_t are output units, the IID net requires Nn_t locations for storage of the M matrix. If a context of P time slots is required to solve the problem then the FID net requires PN locations. Thus the FID net requires less storage for $P < n_t$.

The computational requirements for the FID net are given in equations 5.7 and 5.8. Designating I as the number of instructions needed to compute a single backward pass of the net, as in equation 5.7, then PI instructions must be executed to compute the weight changes per iteration using a minimal length buffer.

The weight changes for the IID net are given by equations 5.14 and 5.18. M_{pij} and D_p are matrices of order n_t and so equation 5.14 requires order n_t multiplications per weight, a total of n_tI computations. Equation 5.18 first requires order of n_tI computations to calculate S_p, and then order of n_t^2 computations to compute $M_{p-1\ ij}T_p$ and order n_t computations for $o_{pj}S_{pi}$, each of which must be computed I times. Taking the highest order, the IID net requires order n_t^2I computations.

In comparison, the IID net is computationally more efficient if $P > n_t^2$. As the examples in this chapter use values of P in the range 1 to 4, and n_t in the range 1 to 17, the IID net is not computationally efficient and so has not been used in any of the examples. Examples may be found in the work of Williams and Zipser (1988).

5.3.3 The State Compression Dynamic Net

The previous two architectures for dynamic nets rely on the propagation of the error signal back in time to define the format of the information in the state units. An alternative approach is to use another error propagation net to define the format of the state units. The overall architecture is given in figure 5.4.

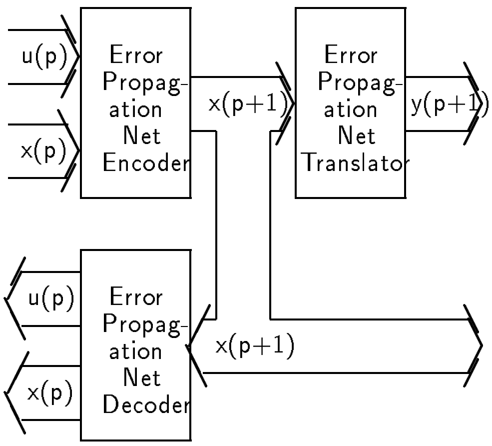

Fig. 5.4: State Compression Net

The encoder net is trained to code the current input vector, u_p, and current state vector, x_p, onto the next state vector, x_{p+1}, while the decoder net is trained to do the reverse operation. The translator net codes the next state, x_{p+1}, onto the desired output, y_{p+1}. This encoding and decoding attempts to represent the current input and the current state in the next state, and by the recursion, it will try to represent all previous inputs. As there is necessarily more information in the input vector and the state vector than in the state vector alone, not all the information can be stored in the state vector. Feeding errors back from the translator net biases this coding of past inputs in the state to those which are useful in computing the output. A feature of this architecture is that recent information tends to be stored in the state units whether it is required to compute the output or not.

5.4 Some Examples of Dynamic Nets

This section presents some simple examples of dynamic nets. Two forms of dynamic nets are used, the finite impulse response dynamic net and the state compression dynamic net. Conventional weighted sum nodes are used with the signed sigmoidal activation function:

$$f(x_{pi}) = \frac{2}{1+e^{-2x_{pi}}} - 1 \qquad (5.19)$$

This function has a maximum value of $+1.0$ and a minimum value of -1.0. However, these values can not be achieved with finite input, so the target values of $+0.8$ for high and -0.8 for low were used instead. Except when stated otherwise, the learning rate η was set to 0.1, and the nets were considered to have learned when the short term residual energy fell below 0.01.

5.4.1 Unit Time Delay

One of the simplest problems for a dynamic net is to reproduce a random one bit input after a unit time delay.

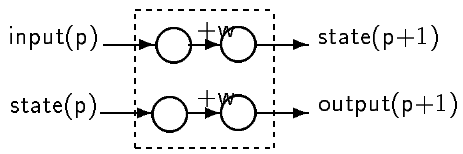

Fig. 5.5: Unit Time Delay

The FID net requires one state unit and no hidden units to learn this task. Figure 5.5 shows the nodes of this net as circles and the significant weights as solid lines. The net learns in about 200 iterations by forming connections of approximately unity strength between the input and the state unit at $p+1$ and between the state unit at p and the output, the connections between the input and output and between the state unit at t and the state unit at $p+1$ being of negligible strength. Thus, in each time slot the the input is copied to the new state unit and the old state unit is copied to the output.

The state compression dynamic net requires two state units, one to store the current input and one to store the previous input. Again about 200 iterations are required for solution.

5.4.2 Bistable

Another basic problem for dynamic nets is to oscillate between states with no external input. An FID net may learn to do this using no input units, one state unit and one output unit, as in figure 5.6.

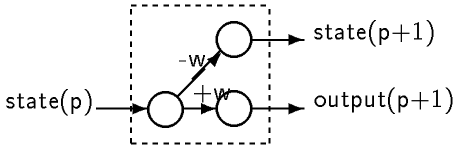

Fig. 5.6: Bistable

With the signed sigmoidal activation function, a negative weight between the state input and output reverses the sign of the state and the sign of the weight to the output unit ensures the desired phase. This net learns in about 250 iterations.

The state compression dynamic net is not able to learn this problem as it stands. With no information in the input units, the energy of the encoder and decoder part of the net is minimised by setting the state units to zero. Zeroed state units also contain no information and so the desired output can not be derived from these units. This may be overcome by presenting an input which is uncorrelated with the output. If a one bit binary input is fed to a net with eight state units, the bistable problem may be learned in about 500 iterations.

5.4.3 Movement Detection

The movement detection problem was inspired by neurobiological research which shows that the human brain contains single cells which can detect movement (Poggio and Koch, 1987). This section presents two problems which are analogous to movement detection on a retina with and without 'wraparound'. The term wraparound means that if the object goes off one side of the retina it appears on the other side. Although the problems may appear to be of similar computational complexity, the differences illustrate a limitation of dynamic nets.

The input to the nets is a one dimensional array of eight units. One of these units is activated whilst the remainder are off, and the activated unit is restricted to be a neighbour of the previously activated unit. There is a single output which is on if the movement is to the left and off if the movement is to the right and the direction of movement is random.

Movement Detection without Wraparound

The FID net architecture involved eight input units and one state unit. This net learns in about 5,300 iterations by developing weights from the input units to the state unit which monotonically increase with distance along the retina. Thus the activation of the state unit represents the current position, and thresholding the difference between this value and the last value of the state unit gives the direction of movement.

Two architectures of state compression nets were considered, one with two and one with nine state units. Two state units represents the minimum required to solve the problem, one to store the present position and one to store the previous position. The first version failed to learn in 2^{16} iterations, presumably because the input contains noise from the random directions and there is no spare information capacity in the state units to record this and still be able to solve the problem. The second architecture was designed to have one state unit for every input, so that there was no problem in replicating the input, and one more unit to store the previous position. This architecture learned in about 1,000 iterations.

Movement Detection with Wraparound

The problem of movement detection with wraparound is the same as the previous case except that an attempt to move off one end of the retina results in the active unit appearing at the other end. This generates some problems as the comparison of two scalars can no longer be used to judge the direction of motion.

The FID net with eight inputs, sixteen hidden units, sixteen state units and one output unit learned all but two transitions in about 6,000 iterations. However the final transition requires a considerable movement in weight space for a small decrease in energy. The direction of movement is determined by the training set, which is corrupted by noise because the direction of movement is random. In attempting to learn the remainder of the problem some activations became very large, reducing the slope of these units to near zero and so blocking the error signal from propagating back through them. The final transition was not learned.

The state compression net had the same number of input, state and hidden units as the FID net. This net uses a different form for the representation of context information and did not suffer from the same instabilities, learning in 1,500 iterations. Thus this state compression net gives a means of solving a problem which was not solved by the FID net.

5.4.4 Letter to Word Conversion

As an example of sequence recognition, the constituent letters of words were presented sequentially to a dynamic net and an output unit corresponding to the word was activated upon its completion. A connectionist solution to this problem has already been formulated with predefined weights (Tank and Hopfield, 1987) and this example shows that it is also possible to learn the weights of a connectionist network which solves this problem.

The nets had 26 inputs (one for each letter of the alphabet), 34 hidden units, 34 state units and 8 output units, one for each of the unique words in "the quick brown fox jumped over the lazy dog". One word was chosen at random and the letters presented sequentially to the net by activating one input unit and switching the rest off. The desired output was for all units to be off until the input after the completion of the word, when one output is activated. The letters of succeeding words were run together without punctuation, so the net had to learn to segment the letters into words and then label the segments. Example input is given in table 5.1. This problem is analogous to the problem in automatic speech recognition where a sequential input stream has to be segmented into symbols and appropriately labeled, as in chapter 7.

The time between repetitions of input-output pairs is considerably longer in this example than any of the previous examples, and the learning rate was correspondingly reduced to $\eta = 0.05$ to compensate.

Several versions of the FID architecture were used, differing in the number of previous letters of context that were considered. When the context was limited to the last letter, or to the last two letters, then the ambiguities can not be resolved. For instance, the letters 'ed' signify the end of the word 'the' followed by 'dog', but also occur in the context of 'jumped'. With three letter

time	input letter activated	output word activated
t-3	.	.
t-2	.	.
t-1	.	.
t	z	none
t+1	y	none
t+2	q	lazy
t+3	u	none
t+4	i	none
t+5	c	none
t+6	k	none
t+7	d	quick
t+8	o	none
t+9	.	.
t+10	.	.
t+11	.	.

Table 5.1: Letter to word conversion

context the FID net learned in 17,000 iterations and with four letter context only 7,500 iterations were required to solution.

The state compression net required the presentation of 15,000 iterations before reducing its energy to below the threshold of 0.01. As there are roughly three times as many weights in a state compression net than in an FID net, the two nets require about the same amount of computation to solve this problem.

5.5 Limitations of Dynamic Nets

Not all of the examples given in the previous section succeeded in learning their designated task. This section explores the problems encountered and gives some considerations for the design of dynamic nets.

The most obvious consideration is that of the complexity of processing which can be learned by the error propagation algorithm. FID and IID dynamic nets necessarily include a layer of units through which the error signals must pass in each time slot. Learning to map an input signal though many such layers is a difficult task, for if one layer is insufficiently trained, as it must be initially, then some of the information is lost and can not be used to train subsequent layers. The same problem manifests itself as a degradation of the error signal, as it propagates back through layers of units its magnitude decreases, thus the units far from the output receive a small degraded signal and take correspondingly longer to learn than those closer to the output. The practical limitation that this imposes is problem dependent and may be of the order of four or eight layers in a layered static net, or a maximum lookback of the same number in a FID or IID dynamic net. A major driving force in the development of the state compression net was to avoid this limit, since the error signals in this net only pass though a single layer of units.

Another consideration relates to the type of problem to be solved. The dynamic net is a finite state machine and thus can not fully emulate any more powerful machine such as a stack machine or a universal Turing machine. However, the dynamic net is able to make an approximate simulation of any machine within the restrictions of internal state space and processing capabilities. The speed of learning is directly related to the probability of the training inputs and outputs occurring in the context of the relevant state vector. For example, a net required to emulate a stack might succeed in emulating the first few elements which occur frequently, but have great difficulty with greater depths which occur infrequently, and can not hope to learn the stack to greater depths than the training examples.

In one example, that of learning movement detection with wraparound, the FID net became unstable and effectively locked in a partial solution. This is not a problem of the net settling in to a local minimum, but one of the net moving to a position in weight space at which the error signals for the training examples become so small as to make further movement impractically slow. This may be a feature of updating the weights after every example and may avoidable in the case of small training set size when the weights can be updated after all examples.

State compression dynamic nets have their own limitations which arise from their architecture. The bistable problem demonstrated that a state compression net must have variation in its input in order form the output, but this is not normally a restriction. The net is also less efficient in the use of state units for storage capacity, state units are used to record the information in the input whether it is required for computation of the output or not. This form of information storage has another effect, the requirements of efficient storage may lead to a change in the format used to store the input information at any time during the learning and such changes of format must then be learned by the translator net in order to maintain the desired output.

Chapter 6

Application to Speech Coding

The problem of speech coding is one of finding a suitable model to remove redundancy and hence reduce the data rate of the speech. The Boltzmann machine learning algorithm has already been extended to deal with the dynamic case and applied to speech recognition (Prager, Harrison and Fallside, 1986b). However, previous use of error propagation nets for speech processing has mainly been restricted to explicit presentation of the context (Robinson, 1986; Elman and Zipser, 1987) with some work using units with feedback links to themselves (Watrous, Shastri and Waibel, 1987). In a similar area, static error propagation nets have been used to perform image coding as well as conventional techniques (Cottrell, Munro and Zipser, 1986).

6.1 The Architecture of a General Coder

The coding principle used in this chapter is not restricted to coding speech data. The general problem is one of encoding the present input using past input context to form the transmitted signal, and decoding this signal using the context of the coded signals to regenerate the original input. Previous chapters have shown that dynamic nets are able to represent context, so two dynamic nets in series form the architecture of the coder, as in figure 6.1.

This architecture may be specified by the number of input, state, hidden and transmission units. There are as many output units as input units and, in this application, both the transmitter and receiver have the same number of state and hidden units.

The input is combined with the internal state of the transmitter to form the coded signal, and then decoded by the receiver using

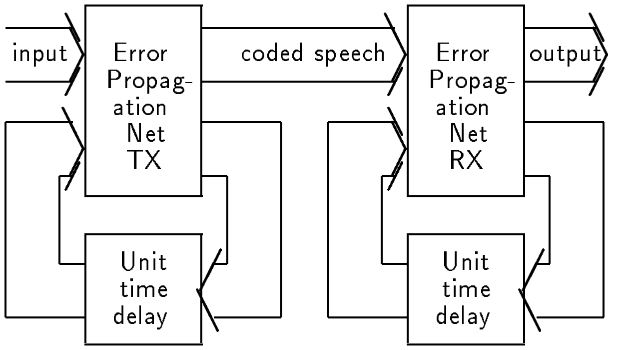

Fig. 6.1: The Architecture of a General Coder

its internal state. Training of the net involves the comparison of the input and output to form the error signal, which is then propagated back through past instantiations of the receiver and transmitter in the same way as a for a FID dynamic net.

It is useful to introduce noise into the coded signal during the training to reduce the information capacity of the transmission line. This forces the dynamic nets to incorporate time information, without this constraint both nets can learn a simple transformation without any time dependence. The noise can be used to simulate the quantisation noise of the coded signal so quantifying the transmission rate. Unfortunately, a straight implementation of quantisation violates the requirement of the activation function to be continuous, which is necessary to train the net. Instead quantisation to n levels may be simulated by adding a random value distributed uniformly in the range $+1/n$ to $-1/n$ to each of the channels in the coded signal.

6.2 Training of the Speech Coder

The chosen problem was to present a single sample of digitised speech to the input, code to a single value quantised to fifteen levels, and then to reconstruct the original speech at the output. Fifteen levels was chosen as the point where there is a marked loss in the intelligibility of the speech, so implementation of these coding schemes gives an audible improvement. Both nets had eight hidden units, with no state units for the static time independent case and four state units for the dynamic time dependent case. A context of the last four samples was used to train the dynamic net.

The data for this problem was 40 seconds of speech from a single male speaker, digitised to 12 bits at 10kHz and recorded in a laboratory environment. The speech was divided into two halves, the first was used for training and the second for testing.

The static and the dynamic versions of the architecture were trained on 21 passes through the training data. At this point the weights were frozen and the inclusion of random noise was replaced by true quantisation of the coded representation. A further pass was then made through both sets of data to yield the performance measurements.

The modified algorithm of Chan and Fallside discussed in section 2.5 was used to dynamically alter the step size scaling factor during training. Previously these machines have been trained with fixed learning parameters and weight updates after every sample (Robinson and Fallside, 1987a) and the use of the adaptive training algorithm resulted in a substantially deeper energy minima. Weights were updated after every 1000 samples, that is about 200 times in one pass of the training data.

6.3 Comparison of Performance

The performance of a coding schemes can be measured by defining the noise energy as half the mean squared error between the actual output and the desired output. This energy is the quantity minimised by the error propagation algorithm. The lower the noise energy in relation to the energy of the signal, the higher the performance. This error measure is often used to test speech coders, but as Thorpe (1987) points out, this does not guarantee a good perceptual quality.

Three non-connectionist coding schemes were implemented for comparison with the static and dynamic net coders. In the first the signal is linearly quantised within the dynamic range of the original signal. In the second the quantiser is restricted to operate over a reduced dynamic range, with values outside that range thresholded to the maximum and minimum outputs of the quantiser. The thresholds of the quantiser were chosen to optimise the signal to noise ratio. The third scheme used the technique of Differential Pulse Code Modulation (DPCM) (for example: Rabiner and Schafer, 1978) which involves a linear filter to predict the speech waveform, and the transmitted signal is the difference between the real signal and the predicted signal. Another linear filter reconstructs the original signal from the difference signal at the receiver. The linear filter is just a special case of the dynamic net with a linear activation function, a discussion of the relationship between linear predictive filters and connectionist models is given by Fallside (1988). The filter order of the DPCM coder was chosen to be the same as the number of state units in the dynamic net coder, thus both coders can store the same amount of context enabling a comparison with this established technique.

The resulting noise energy when the signal energy was normalised to unity, and the corresponding signal to noise ratio are given in table 6.1 for the five coding techniques. Figure 6.2 shows the waveforms for a small section of the test data taken from the start of the word 'did'.

coding method	normalised noise energy	signal to noise ratio/dB
linear, original thresholds	0.071	11.5
linear, optimum thresholds	0.041	13.9
static net	0.049	13.1
DPCM, optimum thresholds	0.037	14.3
dynamic net	0.028	15.5

Table 6.1: Comparison of five speech coding techniques

The static net may be compared with the two forms of the linear quantiser. Firstly note that a considerable improvement in the signal to noise ratio may be achieved by reducing the thresholds of the quantiser from the extremes of the input. This improvement is achieved because the distribution of samples in the input is concentrated around the mean value, with very few values near the extremes. Thus many samples are represented with greater accuracy at the expense of a few which are thresholded. The static net has a poorer performance than the linear quantiser with optimum thresholds. The form of the linear quantiser solution is within the class of problems which the static net can represent, It's failure to do so may be attributed to finding a

Fig. 6.2: Waveforms of Coded Speech

local minima, a plateau in weight space, or corruption of the true steepest descent direction by noise introduced by updating the weights more than once per pass through the training data.

The dynamic net may be compared with DPCM coding. The output from both these coders is the result of filtering the coded signal and, as a result, the output is no longer constrained to discrete signal levels. The dynamic net has a significantly lower noise energy than any other coding scheme, achieved by virtue of the non-linear processing at each unit, and the flexibility of data storage in the state units.

As expected from the measured noise energies, there is an improvement in signal quality and intelligibility from the linear quantised speech through to the DCPM and dynamic net quantised speech.

Chapter 7

Application to Continuous Speech Recognition

Connectionist models provide a new approach to the problem of automatic speech recognition. By adapting a large number of internal parameters, a connectionist machine can learn to map the relevant features of a speech waveform onto the desired word or phoneme label. The error propagation algorithm has had some success already in the field of automatic speech recognition, for example Elman and Zipser (1987) and Burr (1987), have used it to identify a stop followed by a vowel by presenting a fixed length utterance to their machines.

For continuous speech recognition it is important to be able to process the speech a frame at a time with contextual information. As discuused in section 5.1 one way to achieve this is to use a fixed length window which encompasses several frames of speech, training the machine to label the central frame. Robinson (1986), Bourlard and Wellekens (1987c; 1987a) and Prager and Fallside (1988) have built phoneme recognition machines employing this form of context. An alternative way of adding context is to allow feedback of contextual information within the error propagation network. A form of dynamic or 'recurrent' net has been used by Watrous, Shastri and Waibel (1987) to identify stops and vowels.

The work presented in this chapter extends the previous work in three directions. The input is continuous speech so the nets must perform the segmentation as well as the labelling operation. The dynamic net used to process the speech has no 'fixed links' and so it is more flexible than previously used nets. Finally, most of the English phonemes are represented by the output, which illustrates the generality of this approach.

7.1 Weight Update Strategy

Whilst the error propagation algorithm is based on the mathematically clean principle of gradient descent, several 'hacks' were used to improve the performance. If a reduction in energy at every weight change is enforced, then the machine must drop in to the first local minimum, which in practice was found to be at a high energy. This problem can be overcome by adding a 'momentum' term (Rumelhart, Hinton and Williams, 1986) which places a first order filter on the weight changes. Now, by allowing changes that increase the energy, the machine can 'ride over' small local minima. The smoothed estimate of the weight changes is a good approximation to the true direction of steepest descent and the information is available before all the data has been presented. This gives the additional advantage that the weights can be updated more frequently, which results in significantly faster learning.

The choice of step size is also crucial to finding a low energy solution. If a good solution is found with a low step size and the step size is subsequently increased, the machine generates weight changes which considerably overshoot the minimum, resulting in a larger energy. However, a large step size is necessary at the start to achieve a reasonable speed of learning. The Chan and Fallside procedure (section 2.5) for adapting the step size during training was used to provide a large initial step size which decayed when the machine found a minimum. The adoption of this adaptive step size algorithm made a significant contribution to the final performance of the machine.

7.2 Speech Database

The complete database consisted of four utterances of 31 sentences spoken by seven speakers, sampled at 10kHz and digitised to 12 bits. The sentences used in the database, known as the C.U.E.D. Hotel database, are given in table 7.1, and the phonetic description of this database is discussed in section 7.4.

We wish to see the tallest room
What did you tell me
I need to repair this car
Could you send a taxi to meet the bus
Where can I see some art
I must return to the airport
Yes I shall need a credit card
We do not need to see the docks
Should I wear boots on the sea shore
Can you see past the mesh
Can you direct me to the shops
I need a rare treat when I come back
The popular Blackpool rock can be bad
I shall be in my room at six
Can you send the picnic party to the sea shore
Are there any night clubs in Sidney street
Can you list all the banks
This book costs sixty yen
I need a ride to the red light district
We all need sunshine and a palm tree
My little red car might possibly be rusty
See that pretty red pot under my bed
My solicitor shall be in contact soon
This room might be a little bit too dark
Did you put 'yes' on the sheet
You might unwind and relax there
Where do tourists retreat by night
You can expect a canteen at the castle
We all insist on the aquarium
I wish to discuss additional treatment
We remarked on the obscure club across the street

Table 7.1: Sentences used in the Hotel database

For the single speaker work, two of the four utterances from a single speaker (speaker "mha") were used for training and the other two used for testing. For the multiple speaker work, a single utterance was taken from each speaker for training and a second for testing.

The training sentences were displaced by eight different temporal offsets before presentation to the preprocessor, resulting in eight slightly differing versions. Training using all eight versions improved the performance of the net when presented with test data of unknown temporal shift. In total there were about 300,000 input and output pairs in the multiple speaker training data, making this one of the largest problems tackled to date using the error propagation algorithm. The achievement of reasonable training times was only made possible by running the simulation on a 64 processor array of T800 transputers, as described by Chong and Fallside (1988) and in appendix A.

7.3 Front End Processing

The preprocessor takes a raw speech waveform and transforms it into a form suitable for presenting to the net. The requirements for a good preprocessor are to perform as much of the initial stages of the speech-to-phoneme transformation as possible whilst losing as little of the information in the speech signal which is needed in later stages. Although it is possible not to use a preprocessor, it is computationally advantageous not to have to learn the initial processing, and not to have to repeat this processing every time the data is presented to the net.

Two assumptions are common in speech processing: the first is that the power spectrum of the speech waveform is slowly varying; and the second is that the phase and pitch of the speech signal is unimportant. These may be realised by Hamming windowing the speech then performing a Fourier transform to give a short term power spectrum. The power spectrum contains pitch and vocal tract information but by grouping adjacent frequencies into 'bins' the pitch information is blurred. Twenty bins were used, spaced evenly on a Bark scale, which is a non-linear frequency scale derived from psychoacoustic measurements into masking by Zwicker and Terhart (1980). The energy in each of these bins display a large dynamic range, which was reduced by taking the cube root of each value, so distributing the example points more evenly in the input space.

The behaviour of the dynamic net is dependent on the time scale of the preprocessor. If a small frame size is chosen then the contextual information must be stored in the state units for many frames of speech. If a large frame size is chosen then there will be a loss of resolution on short time scale events, such as bursts. A common choice is to use a 25.6ms Hamming window repeated every 12.8ms which gives a stable spectrum over steady state portions of speech. In addition to this spectrum the four short term energies which fall in the repeat distance were used, so allowing for the timing and identification of bursts.

7.4 Phoneme Labels

Phonemes are regarded as the smallest linguistic unit that can be used to distinguish meaning (for example: Ladefoged, 1982: p 23). By their symbolic nature they provide a natural boundary for artificial speech recognition systems between the lower level distributed representations such as the acoustic waveform and its transformations, and the higher level symbolic representations such as words and the representation of syntactic and semantic knowledge.

The phoneme set used in this chapter consists of 27 of the phonemes of English and a label for silence. Diphthongs were not represented explicitly; instead occurrences were labelled as two adjacent vowels. Table 7.2 gives the International Phonetic Association (I.P.A.) symbols, the machine readable symbols which are modified from ARPABET symbols, and an example of the occurrence of these symbols. The output the net consisted of one unit per label which had a target activation of 0.8 if the label was active, and −0.8 if not.

I.P.A.	ARPA-BET	example	I.P.A.	ARPA-BET	example
/iː/	IY	bEAd	/l/	L	LoyaL
/ɪ/	IH	bId	/r/	R	Rear
/ɛ/	EH	bEd	/m/	M	MiMe
/æ/	AE	bAd	/n/	N	NoNe
/ɑː/	AA	bARd	/b/	B	BarB
/ɒ/	OH	bOdy	/d/	D	DeeD
/ɔː/	AO	bAWdy	/p/	P	PiP
/ʊ/	UH	bOOk	/t/	T	TighT
/uː/	UW	bOOt	/k/	K	KicK
/ɜː/	XX	bIRd	/ð/	DH	oTHer
/ə/	AX	bAnanA	/ʒ/	ZH	meaSure
/ʌ/	AH	bUd	/s/	S	CeaSe
/j/	Y	Year	/ʃ/	SH	SHeepiSH
/w/	W	Weal		__	*silence*

Table 7.2: Phonetic labels used in the database

7.5 Back End Processing

To measure the performance of the net, and to interface to higher level processing, a symbolic output is useful. One method of converting the distributed output representation of the net to a symbolic form is simply to choose the unit with the largest activation, and output the corresponding symbol. The largest activation may then be considered as a certainity measure on this decision. Whilst this is a reasonable scheme, it makes poor use of the information contained in the activations of the remaining units.

The approach adopted here is to use the minimum mean squared error to define an energy for each of the possible target outputs, as discussed in section 3.3. This forms a local distance between the actual output and each different phoneme label and the error propagation algorithm minimises this distance for the correct label.

This local distance may be used directly to measure the performance of labelling frames, or it can be used to recognise templates consisting of a string of phoneme labels. The problem of finding the correct sequence of templates for an unknown chapter of speech is efficiently solved using the technique of Dynamic Time Warping (D.T.W.). This technique has the advantage that it finds the minimum value of the summed local distances over the whole of the speech, precisely the quantity minimised by the error propagation algorithm for the training data.

An example plot of the activations of the output units, and the corresponding local distances for the single speaker dynamic net is given in figures 7.1 and 7.2. A continuous line is plotted for each of the phoneme labels, and where this coincides with the hand label it is marked with a cross. The symbolic form of the hand label and the best machine label are given under the plot. The performance figures quoted in the remainder of this chapter refer to the proportion of occurrences where the label of a hand labelled frame is also the label with lowest energy.

Fig. 7.1: Activations of Output Units

Fig. 7.2: Energy of Output Units

7.6 Static Net Results

The fixed context static net was trained on both the single speaker and the multiple speaker training data, for a range of contexts and numbers of hidden units. The performance is given in table 7.3. The entries marked '-' were not available as the number of weights exceeded limitations on memory capacity, which corresponded to about 20,000 weights. From the table it can be seen that increasing the number of hidden units makes a small contribution to the performance as compared with increasing the number of frames of context. Indeed, in the cases considered here, the performance of the nets was often worse with 128 hidden units than with 32 hidden units.

no. of 12.8ms frames	no. of hidden units					
	single speaker			seven speakers		
	0	32	128	0	32	128
1	52.7%	53.1%	53.6%	41.2%	43.1%	41.0%
3	57.5%	60.0%	59.3%	44.5%	47.7%	45.6%
7	64.2%	69.2%	-	50.3%	53.4%	-
15	70.8%	73.0%	-	54.7%	57.5%	-
31	74.0%	-	-	59.2%	-	-

Table 7.3: Fixed context performance for single and multiple speakers

7.7 Dynamic Net Results

When training the dynamic network it is possible, and indeed advantageous, to displace the phoneme labels from the portion of speech to which they refer. Placing the labels later in time than the speech segment has the advantage that the net can not only use the information from the speech before the segment (backward context) but also a finite amount of speech after the segment (forward context). Thus the later the output of the phoneme label is delayed, the greater the amount of context available to the machine which can be used to classify the segment. The disadvantage of a long delay is that the information relating to the output must be held in the finite store of the state units over the period of the delay. The performance of a dynamic net with 64 state units on the single speaker and multiple speaker tasks is given in table 7.4.

delay/ frames	single speaker	seven speakers
2	75.2%	67.0%
4	76.5%	69.0%
6	78.1%	70.8%
8	77.3%	69.2%

Table 7.4: Dynamic net performance for single and multiple speakers

7.8 K Nearest Neighbour (KNN) Results

The previous results may be compared to the technique of k nearest neighbour analysis which takes a majority vote on the k training frames which have the smallest Euclidean distance from the test frame. Table 7.5 gives the best single speaker recognition performance over a range of contexts for values of k up to 16. The recognition accuracy decreases with increasing context because, unlike the fixed context net, equal weighting is given to all frames.

context	best k	performance
1	3	48.1%
3	3	45.9%
7	3	44.2%
15	2	44.0%

Table 7.5: KNN single speaker performance

KNN analysis was also carried out for all seven speakers with a single frame of context. The KNN and best dynamic net results are broken down in table 7.6 according to speaker. This table shows that there is a large variation in recognition accuracy across speakers, and the speech of one speaker in particular (speaker "jcn") was considerably harder to recognise than the remainder.

speaker	best k	KNN	dynamic net
jcn	3	39.1%	60.9%
doc	5	40.5%	72.4%
rwp	5	43.1%	72.3%
mvs	5	43.2%	68.0%
rjn	3	47.0%	73.2%
reb	4	48.5%	73.0%
mha	6	49.2%	72.4%
mean	-	44.4%	70.3%

Table 7.6: Speaker variation

7.9 Discussion

This chapter has presented two forms speech recogniser based on error propagation nets and one based on KNN analysis. Both forms of net gave better performance on single and multiple speaker recognition tasks than the KNN analysis, and did so with considerably less computation during recognition. The nets require little more than one multiplication and addition operation per weight, which is also an order of magnitude less computation than required by either the preprocessor or a D.T.W. back end processor.

The dynamic net shows better recognition performance than the fixed context static net in both recognition tasks, and does so with fewer weights than the best performing static nets. The two most common types of error made by the nets are the misplacement of a boundary between phonemes and mis-labelling with a similar sounding phoneme. Indeed these decisions are difficult in principle, there are no formal rules and the 'right' answer may be subjective. It has been the aim of this chapter to show that dynamic error propagation nets are a valid approach to time dependent problems that must be trained by example.

Chapter 8

Reinforcement Driven Dynamic Nets

Both the static net and the dynamic net presented so far have been trained pairs of input and output vectors. In contrast, a reinforcement driven net is trained by repreated presentation of an input and a judgement on the calculated output, the reinforcement signal. Reinforcement driven dynamic nets learn to calculate the output which maximises the reinforcement signal for a given input stream. This is a very powerful property, since all that is required is that the reinforcement signal is a function of the input and output data streams within the considered context. Such nets have the potential to create complex internal models of the external world based on their own interrelationship with it. Applications may include the control of industrial plant where only the performance of the system as a whole is known, or the building of 'intelligent' machines.

8.1 Architectures

A reinforcement driven dynamic net can be formed from two dynamic nets and a possible architecture is shown in figure 8.1. The first dynamic net (net Y) computes the overall output from the input and the second net (net Z) takes both the output and input and learns their relationship to the reinforcement signal.

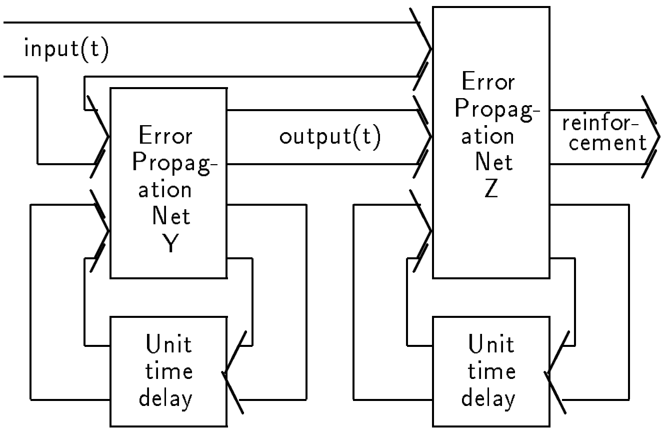

Fig. 8.1: The Reinforcement Driven Dynamic Net

Both nets train at the same time, although without net Z it is impossible to train net Y at all. Net Z learns the relationship between the behaviour of net Y and the reinforcement signal using the learning procedure for dynamic nets without any modification. Net Y can not be trained directly as the desired output is only specified as that which maximises the reinforcement signal. However, the error signal for the output can be calculated by setting the output of net Z to its desired value, that is with the reinforcement signal high, and propagating this error signal through net Z and subsequently though net Y. This is a new use for the error signal in error propagation networks, for it is simultaneously used to train a net and to generate the error signal for training another net.

In practice net Y and net Z may be combined to achieve a more compact net, as in figure 8.2. This net may be trained

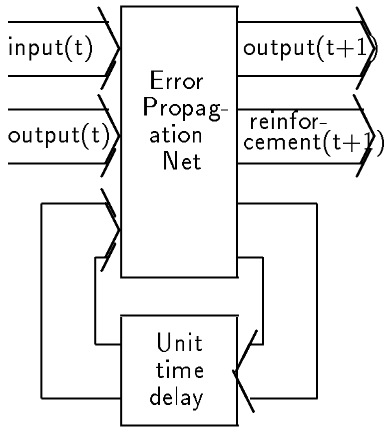

Fig. 8.2: A Compact Reinforcement Driven Dynamic Net

in a similar way to a FID or IID net. Knowing the observed reinforcement and output, an error signal can be generated and propagated though successive sets of state units to learn the mapping function. Now by setting the target reinforcement high a second error signal is generated which is propagated back through the current net to form the error signal for the previous outputs. This error signal is then propagated through previous sets of state units to maximise the reinforcement. As observed earlier, the propagations of errors is a linear process, so the error signals may be combined by addition and propagated back as a single signal, so reducing the computation. This reinforcement driven net can be specified by the number of input, output, hidden and state units. A simple example of this net has been presented by Robinson and Fallside (1987a) in which the net received a high reinforcement if the output has the same sign as the previous input, otherwise the reinforcement was low.

8.2 A General Game Playing Program

A subclass of the general net outlined above can be used for game playing, in which case the reinforcement signal is defined only at the end of the game. The technique adopted here is to play a game using the network of figure 8.1, storing the intermediate activations of all units. At the end of the game two separate computations are performed, one to make a more accurate prediction of the reinforcement signal signal at the end of the game, and the second to bias this signal to the 'high' reinforcement state.

8.2.1 Noughts and Crosses

The game of 'noughts and crosses' was chosen for several reasons:

- It is well known and regarded as a 'simple' childrens game. However, it may be classified as 'difficult' when judged by the current standard of connectionist reinforcement driven learning procedures.

- The mapping of a board position onto the optimal next move is a complex non-linear function requiring the learning of disjoint pattern classes.
- The state of the game is uniquely defined by the board position, so that net Z does not need any state units.
- The board may be represented in relatively few bits. Each of the nine locations may be unoccupied or occupied by either a 'O' or a 'X'. Thus an upper bound on the number of legal states is $3^9 = 19683$, which can be represented in 15 bits.
- The game has a short duration as no player may place more than five pieces on the board. Thus as far as assigning credit or blame for the outcome of the game, the error signal must be propagated back through a maximum of five states.

The 'opponent' to the net was a simple algorithm that would win by completing a line of two if possible, otherwise a piece would be placed randomly. If the net places pieces randomly, as is the case before any learning, then the net wins about 30% of the games which are not drawn. A suitably experienced player would never lose and only occasionally draw against this algorithm.

8.2.2 Input and Output Representations

Two 3x3 matrices were used to represent the board position. One has each element set high if the corresponding board position is occupied, the other is used to record the owner of the piece placed on the occupied site. No attempt was made to take advantage of the symmetry of the game.

The output representation was chosen to be a 3x3 matrix, the legal move with the largest value in this matrix was taken as the move to be made. A strict winner-takes-all rule was found to be unstable, as the magnitude of the difference between the largest and the second largest activation is unimportant which results in a discontinuity in the weight space. For example, a marginal difference of activity in an output unit will change the move made during a game, so changing the outcome of the game. So, for the same reason as step activation functions can not be used within a network, a step response in interpreting the output must be avoided. An alternative probabilistic representation of the the output vector was used to improve the stability. This was implemented by adding random noise to the output vector before choosing the largest element. The noise was generated by the difference of two random numbers with range 0.1. Thus if one output was more than 0.2 above all the others this noise is unimportant, otherwise the noise results in random decisions which, when averaged, blur out the discontinuities in the weight space.

8.2.3 Implementation

The machine was trained the 65 processor array of T800 transputers described in Appendix A. Two hundred games were played per processor per update, so the weights were updated on gradient information collected over 13000 games. Each net had 18 hidden units and the target values for the outputs were chosen to be in the linear region of the signed activation function, $+0.1$ for positive reinforcement and -0.1 for negative reinforcement.

8.2.4 Problems with Reinforcement Driven Learning

Reinforcement driven learning is a harder task than supervised learning for the simple reason that less information is provided about the desired output. For the game playing program presented here there is the additional problem that changes to the weights change the response given to early moves, so the whole style of the game can change. For example, the initial moves are random, so the prediction and maximisation of the reinforcement signal is carried out for nearly fully populated boards of randomly placed pieces. However, towards the end of the learning period the game length has become shorter and there are correspondingly fewer pieces on the board. Thus the prediction and maximisation functions must relearn for this new set of training data. Because the form of the input is dependent on the current performance of the net, the net does not perform a gradient descent in a single function throughout the training, but performs a gradient descent in a continually changing function. Thus there is no guarantee of convergence.

The algorithm used as an 'opponent' to the net used a random number generator to pick a legal move if it could not place a piece to win the game. This randomness means that some responses would be given more often than others in an unpredictable way, which hinders the learning by the introduction of noise into the error signal.

The problem with needing a symbolic representation in the middle of the network has already been discussed. In a 'real world' environment, such as robot control, the analogue output may be appropriate so avoiding the problem.

8.2.5 Results

The initial performance of the net was to win about 30% of the games played. After playing 300,000 games this figure improved to 59%, and playing a further 10^8 games produced no further improvement. Whilst the performance of the net is lower than the optimal performance, the net did learn sufficiently to perform better than the 'opponent' algorithm which it played against.

Chapter 9

Conclusion

The original work presented in this thesis concerns the development of supervised, unsupervised and reinforcement driven dynamic error propagation nets. More specifically:

- The generalisation of node types for the error propagation network which broadens the scope of these networks to include connectionist models such as the modified Kanerva model and those of radial basis functions.
- The formulation of the Infinite Input Duration Dynamic Network. This network has the same properties as a standard recurrent network which extends back in time to the first input. This is achieved with fixed storage an computational requirements, unlike the standard recurrent network whose resources required increase linearly with time.

- The formulation of the State Compression Network which can be viewed as a superset of the networks used by Jordan (Jordan, 1986) and followers. Like the Infinite Input Duration Dynamic Network this network has the advantage that the computational and memory resources are fixed, and lower than those required for the previous network. This advantage is achieved at the expense that the network no longer performs a true gradient descent.
- The application of dynamic nets to speech coding. The popular method of Differential Pulse Code Modulation employs two linear filters to encoded and decode speech. By generalising these to non-linear filters, implemented as dynamic nets, a reduction in the noise imposed by a limited bandwidth channel was achieved.
- The application of dynamic nets to a continuous speech recognition task. The field of Automatic Speech Recognition is one where the required mapping from the acoustic waveform to the symbolic phoneme string is unknown. The use of dynamic nets was found to give a higher recognition rate both in comparison with a fixed window net and with the established k nearest neighbour technique. This is one of the largest connectionist problems reported to date.
- The development of the reinforcement driven dynamic error propagation network. The example problem of learning to play the game of noughts and crosses has shown that whilst the performance is inferior in comparison to symbolic programs, the connectionist approach was able to perform better than the opponent algorithm.

The author has two positive hopes for connectionism, firstly that it will yield useful products like a voice operated wordprocessor and other aids for the disabled, and secondly that it will stir people up into realising that there is nothing special about the human brain other than as a powerful information processing system. On the negative side, there is already considerable interest being shown by the military who are already far too good at killing people without the help or hindrance of connectionist models.

Appendix A

Implementation on a Transputer Array

The simulations described in chapters 7 and refch:reinforce were implemented on a 65 processor array of T800 transputers developed under the ParSiFal project IKBS/146 using a scheme similar to that used by Chong and Fallside (1988). The T800 transputer contains a floating point unit and four hardware links for connection with other transputers. The transputers are hosted by a Sun 3/110 which contained one of the T800's, (the 'tadpole' transputer), and the remaining 64, (the 't-rack' transputers), were housed externally. The layout of this system is given schematically in Figure A.1.

For the speech recognition experiment, data was downloaded onto the transputer array over an Ethernet connection to the Sun. The Sun performed some data type conversion before writing to an area of dual port memory shared with the tadpole transputer. Handshaking of this data transfer was achieved by using a link adapter that was polled by the Sun to receive synchronisation bytes from the tadpole transputer. The data were divided up over all the transputers such that the n^{th} transputer in the ring started its data buffer $n/64$ of the way into the whole of the data. As each data buffer was larger than 1/64 of the whole data there was some overlap of data between transputers. Typically 10 Mbytes of data were downloaded from a remote host in 3 minutes.

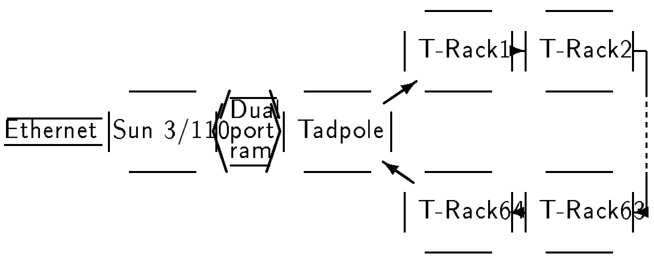

Fig. A.1: The architecture of the transputer array

All t-rack processors executed the same piece of code, so approximating a 'Single Instruction Multiple Data' (SIMD) architecture. This was advantageous in that it simplified the design of the software and reduced the amount of code that had to be written. The speech recognition simulations were written in OCCAM and the reinforcement driven learning experiments were written in C. Each learning cycle was split up into four phases, as given in table A.1.

step	tadpole process	t-rack process
0	broadcast weights	receive new weights
1	write old weights to Sun	calculate new changes
2	send null gradient vector	add local gradient
3	update weights	idle

Table A.1: The four phases of a learning cycle

An LED was switched on when the processor was waiting for or performing communications, this was used to confirm that negligible time was spent in internal communications. The performance limiting step on initialisation varied between the data transfer rate over the Ethernet and converting between data types on the Sun, depending on the loadings of the respective systems. The performance limiting step during the learning phase was the speed of the floating point unit internal to the T800.

Some problems were encountered in parallelising the original sequential code. In the original sequential implementation of the speech recognition code the weights were updated after every 200 frames. However, in the parallel implementation at least 16 frames were required to provide sufficient context for the feedback, which when multiplied by the 64 parallel processes gives an update every 1024 frames. This was found to be a problem in the initial stages when a small amount of data was used from a single speaker as 1024 frames represented a significant proportion of the data set. However, with the increased processing power available, the number of data frames could be increased to about 300,000 at which point the quantisation of the weight updating was no longer a problem.

A rough comparison between the transputer array and a VAXstation II with a floating point unit yielded a speed up of about 300. Chong and Fallside (1988) investigated a similar problem and found a linear speed up by a factor of 280. The results contained in this thesis could not have been attained without access to this, or equivalent, hardware.

Bibliography

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, **9**, 147–169.

Almeida, L. B. (1987). 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.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, **13**(5), 834–846.

Baum, E. B. and Wilczek, F. (1987). Supervised learning of probability distributions by neural networks. In *Proceedings of Neural Information Processing Systems* (ed. D. Z. Anderson), American Institute of Physics, Denver.

Boothroyd, C. B. (1989). Department of Material Science, Cambridge University. Personal communication.

Bourlard, H. and Kamp, Y. (1987). *Auto-Association by Multilayer Perceptrons and Singular-Value Decomposition.* Manuscript M217, Philips Research Laboratory, Brussels.

Bourlard, H. and Wellekens, C. J. (1987a). *Speech Pattern Discrimination and Multilayer Perceptrons.* Manuscript M211, Philips Research Laboratory. Submitted to *Computer Speech and Language*.

Bourlard, H. and Wellekens, C. J. (1987b). *Links between Markov Models and Multilayer Perceptrons.* Manuscript M263, Philips Research Laboratory.

Bourlard, H. and Wellekens, C. J. (1987c). Multilayer perceptrons and automatic speech recognition. In *Proceedings of the IEEE First Annual International Conference on Neural Networks*, pages IV:407–416, San Diego.

Broomhead, D. and Lowe, D. (1988). *Multi-variable Interpolation and Adaptive Networks.* RSRE memo #4148, Royal Signals and Radar Establishment, Malvern.

Burr, D. J. (1987). Speech recognition results with perceptrons. In *Proceedings of Neural Information Processing Systems* (ed. D. Z. Anderson), American Institute of Physics, Denver.

Chan, L. W. and Fallside, F. (1987a). *An Adaptive Learning Algorithm for Back Propagation Networks.* Technical Report CUED/F-INFENG/TR.2, Cambridge University Engineering Department.

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

Chong, M. W. H. and Fallside, F. (1988). *A Parallel Implementation of a Neural Network for Speech Recognition.* Technical Report CUED/F-INFENG/TR.8, Cambridge University Engineering Department.

Chong, M. W. H., Fallside, F., Marsland, T. P., and Prager, R. W. (1989). *Parallel Processing for Interactive Speech Recognition.* Technical Report in preparation, Cambridge University Engineering Department.

Cottrell, G. W., Munro, P., and Zipser, D. (1986). *Image Compression by Back Propagation: An Example of Existential Programming.* ICS Report 8702, Institute for Cognitive Science, University of California, San Diego.

Deterding, D. H. (1988). *Speaker Normalisation for Automatic Speech Recognition.* PhD thesis, University of Cambridge.

Elman, J. L. and Zipser, D. (1987). *Learning the Hidden Structure of Speech.* ICS Report 8701, University of California, San Diego.

Fallside, F. (1988). *On the Analysis of Linear Predictive Data such as Speech by a Class of Single Layer Connectionist Models.* Technical Report CUED/F-INFENG/TR.27, Cambridge University Engineering Department. Presented at the 2$^{\text{nd}}$ Symposium on Advanced Man-Machine Interface, Hawaii.

Feldman, J. A. (1986). *Neural Representation of Conceptual Knowledge.* Technical Report TR189, Department of Computer Science in the University of Rochester, Rochester, New York 14627.

Ferry, G. (1987). Networks on the brain. *New Scientist*, July, 54–58.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learing.* Addison Weslay.

Hanson, S. J. and Burr, D. J. (1987a). Minkowski-r back-propagation: learning in connectionist models with non-euclidean error signals. In *Proceedings of Neural Information Processing Systems* (ed. D. Z. Anderson), American Institute of Physics, Denver.

Hanson, S. J. and Burr, D. J. (1987b). *Knowledge Representation in Connectionist Networks.* Technical Report, Bell Communications Research, New Jersey.

Harrison, T. D. (1988). *A Connectionist Framework for Continuous Speech Recognition.* PhD thesis, Cambridge University Engineering Department.

Hebb, D. O. (1949). *The Organization of Behavior.* Wiley, New York.

Hinton, G. E. (1987). *Connectionist Learning Procedures.* Technical Report CMU-CS-87-115, Computer Science Department, Carnegie-Mellon University.

Hinton, G. E. and McClelland, J. (1987). Learning representations by recirculation. In *Proceedings of Neural Information Processing Systems* (ed. D. Z. Anderson), American Institute of Physics, Denver.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representations. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. I: Foundations.* (eds. D. E. Rumelhart and J. L. McClelland), chapter 3, Bradford Books/MIT Press, Cambridge, MA.

Hinton, G. E., Sejnowski, T. J., and Ackley, D. H. (1984). *Boltzmann machines: constraint satisfaction networks that learn.* Technical Report CMU-CS-84-119, Carnegie Mellon University.

Holland, J. H. (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science U.S.A.*, **79**, 2554–2558.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Science U.S.A.*, **81**, 3088–3092.

Huang, W. Y. and Lippmann, R. P. (1987). Neural net and traditional classifiers. In *Proceedings of Neural Information Processing Systems* (ed. D. Z. Anderson), American Institute of Physics, Denver.

Jacobs, O. L. R. (1974). *Introduction to Control Theory*. Clarendon Press, Oxford.

Jordan, M. I. (1986). *Serial Order: A Parallel Distributed Processing Approach*. ICS Report 8604, Institute for Cognitive Science, University of California, San Diego.

Kanerva, P. (1984). *Self Propagating Search: A Unified Theory of Memory*. PhD thesis, Stanford University Centre for the Study of Language and Information.

Kawahara, H. and Irino, T. (1988). Introduction to saturated projection algorithm for artificial neural network design. NTT Basic Research Laboratories, 3-9-11 Midori-cho Musashino, Tokyo 180, Japan.

Kirkpatrick, S., Gelatt, Jr., C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, **220**, 671–680.

Kohonen, T. (1988). *Self-Organization and Associative Memory*. Springer-Verlag, New York, second edition.

Kuffler, S. W., Nicholls, J. G., and Martin, A. R. (1984). *From Neuron to Brain: A Cellular Approach to the Function of the Nervous System*. Sinauer Associates Inc., Sunderland, MA, second edition.

Ladefoged, P. (1982). *A Course in Phonetics*. Harcourt Brace Jovanovich, New York, second edition.

Lapedes, A. and Farber, R. (1987). How neural nets work. In *Proceedings of Neural Information Processing Systems* (ed. D. Z. Anderson), American Institute of Physics, Denver.

Lindsay, P. H. and Norman, D. A. (1977). *Human Information Processing: An Introduction to Psychology*. Academic Press, Inc., Orlando, Florida, second edition.

Linsker, R. (1986). From basic network principles to neural architecture. *Proceedings of the National Academy of Science U.S.A.*, **83**, 7508–7512; 8390–8394; 8779–8783.

Linsker, R. (1988). Self-organization in a perceptual network. *IEEE Computer*, **21**(3), 105–117.

Lippmann, R. P. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, **4**(2), 4–22.

McClelland, J. L. and Rumelhart, D. E. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. II: Psychological and Biological Models*. MIT Press, Cambridge, MA.

McCulloch, N. and Ainsworth, W. A. (1988). Speaker independent vowel recognition using a multi-layer perceptron. In *Proceedings of Speech'88*, Edinburgh.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA.

Moody, J. and Darken, C. (1988). *Learning with Localised Receptive Fields*. Research Report YALEU/DCS/RR-649, Yale University Department of Computer Science.

Munro, P. W. (1987). A dual back-propagation scheme for scalar reinforcement learning. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seatle, WA.

Niranjan, M. (1988). Cambridge University Engineering Department. Personal communication.

Niranjan, M. and Fallside, F. (1988). *Neural Networks and Radial Basis Functions in Classifying Static Speech Patterns*. Technical Report CUED/F-INFENG/TR.22, Cambridge University Engineering Department.

Parker, D. B. (1982). *Learning Logic, Invention Report*. Technical Report S81–64, File 1, Office of Technology Licencing, Stanford University.

Plumbley, M. D. (1988). Cambridge University Engineering Department. Personal communication.

Plumbley, M. D. and Fallside, F. (1988). *An Information-Theoretic Approach to Unsupervised Connectionist Models*. Technical Report CUED/F-INFENG/TR.7, Cambridge University Engineering Department.

Poggio, T. and Koch, C. (1987). Synapses that compute motion. *Scientific American*, May, 42–48.

Powell, M. J. D. (1985). *Radial Basis Functions for Multivariable Interpolation: A Review*. Report DAMTP 1985/NA12, Department of Applied Mathematics and Theoretical Physics, University of Cambridge.

Prager, R. W. and Fallside, F. (1988). *The Modified Kanerva Model for Automatic Speech Recognition*. Technical Report CUED/F-INFENG/TR.6, Cambridge University Engineering Department.

Prager, R. W., Harrison, T. D., and Fallside, F. (1986a). *Boltzman Machines for Speech Recognition*. Technical Report CUED/F-CAMS/TR.260, Cambridge University Engineering Department.

Prager, R. W., Harrison, T. D., and Fallside, F. (1986b). Boltzmann machines for speech recognition. *Computer Speech and Language*, **1**(1), 3–27.

Rabiner, L. R. and Schafer, R. W. (1978). *Digital Processing of Speech Signals*. Prentice Hall, Englewood Cliffs, New Jersey.

Rayner, P. J. W. and Lynch, M. R. (1988). A new connectionist model based on a non-linear adaptive filter. Cambridge University Engineering Department.

Renals, S. and Rohwer, R. (1989). Phoneme classification experiments using radial basis functions. Submitted to the International Joint Conference on Neural Networks.

Robinson, A. J. (1986). *Speech Recognition with Associative Networks*. M.Phil. Computer Speech and Language Processing Thesis , Cambridge University Engineering Department.

Robinson, A. J. and Fallside, F. (1987a). *The Utility Driven Dynamic Error Propagation Network*. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department.

Robinson, A. J. and Fallside, F. (1987b). Static and dynamic error propagation networks with application to speech coding. In *Proceedings of Neural Information Processing Systems* (ed. D. Z. Anderson), American Institute of Physics, Denver.

Robinson, A. J. and Fallside, F. (1988). A dynamic connectionist model for phoneme recognition. In *Proceedings of nEuro'88*, Paris.

Robinson, A. J., Niranjan, M., and Fallside, F. (1988). *Generalising the Nodes of the Error Propagation Network*. Technical Report CUED/F-INFENG/TR.25, Cambridge University Engineering Department.

Rohwer, R. (1988). Instant solutions for perceptron-like nets. Submitted to *Neural Computation*.

Rohwer, R. and Forrest, B. (1987). Training time-dependence in neural networks. In *Proceedings of the IEEE First Annual International Conference on Neural Networks*, pages II:701–710, San Diego.

Rohwer, R. and Renals, S. (1988). Training recurrent networks. In *Proceedings of nEuro'88*, Paris.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, **65**, 386–408.

Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan, New York.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). *Learning Internal Representations by Error Propagation*. Technical Report ICS-8506, University of California, San Diego.

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

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

Scales, L. E. (1985). *Introduction to non-linear optimisation*. Macmillan.

Sejnowski, T. J. and Rosenberg, C. R. (1986). *NETtalk: A Parallel Network that Learns to Read Aloud*. Technical Report JHU/EECS-86/01, The John Hopkins University Electrical Engineering and Computer Science Department.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. L. (1988). *Encoding Sequential Structure in Simple Recurrent Networks*. Technical Report CMU-CS-88-183, Carnegie Mellon University.

Stornetta, W. S., Hogg, T., and Huberman, B. A. (1987). A dynamical approach to temporal pattern processing. In *Proceedings of Neural Information Processing Systems* (ed. D. Z. Anderson), American Institute of Physics, Denver.

Sutton, R. R. (1984). *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Department of Computer and Information Science.

Tank, D. W. and Hopfield, J. J. (1987). Neural computation by concentrating information in time. *Proceedings of the National Academy of Science U.S.A.*, **84**, 1896–1900.

Thorpe, T. F. (1987). *Performance Bounds for Digital Coding of Speech*. PhD thesis, Cambridge University Engineering Department.

Vogl, T. P., Manglis, J. K., Rigler, A. K., Zink, W. T., and Alkon, D. L. (1988). *Accelerating the Convergence of the Back-Propagation Method*. Technical Report, Enviromental Research Institute of Michigan, Arlington.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. (1987). *Phoneme Recognition Using Time–Delay Neural Networks*. Technical Report, ATR Interpreting Telephony Research Laboratories.

Watrous, R. L. and Shastri, L. (1987). Learning phonetic features using connectionist networks: An experiment in speech recognition. In *Proceedings of the IEEE First Annual International Conference on Neural Networks*, pages IV:381–388, San Diego.

Watrous, R. L., Shastri, L., and Waibel, A. H. (1987). Learned phonetic discrimination using connectionist networks. In *Proceedings of the European Conference on Speech Technology* (eds. J. Laver and M. A. Jack), CEP Consultants Ltd, Edinburgh.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *Proceedings WESCON*, pages 96–104.

Williams, R. J. (1986). *Reinforcement Learning in Connectionist Networks: A Mathematical Analysis*. ICS Report 8605, Institute for Cognitive Science, University of California, San Diego.

Williams, R. J. and Zipser, D. (1988). *A Learning Algorithm for Continually Running Fully Recurrent Neural Networks*. ICS Report 8805, Institute for Cognitive Science, University of California, San Diego.

Zwicker, E. and Terhart, E. (1980). Analytical expressions for critical-band rate and critical bandwidth as a function of frequency. *Journal of the Acoustical Society of America*, **68**, 1523–1525.