# The Time Dimension of Neural Network Models

Richard Rohwer
Dept. of Computer Science and Applied Mathematics
Aston University
Aston Triangle
Birmingham B4 7ET
UK
rohwerrj@uk.ac.aston.cs

## Abstract

This review attempts to provide an insightful perspective on the role of time within neural network models and the use of neural networks for problems involving time. The most commonly used neural network models are defined and explained giving mention to important technical issues but avoiding great detail. The relationship between recurrent and feedforward networks is emphasised, along with the distinctions in their practical and theoretical abilities. Some practical examples are discussed to illustrate the major issues concerning the application of neural networks to data with various types of temporal structure, and finally some highlights of current research on the more difficult types of problems are presented.

## 1 Introduction

Neural network models can be used to process time-varying data for various purposes using various methods. Research aimed at improving the methodology makes contact with a broad range of disciplines and raises a panoply of difficult, though enticing, questions. This happens largely because the brain not only lives in an environment filled with time-varying data, but also generates internal time-varying signals of its own. The internal signals vary from direct responses to environmental events to obscure mechanisms for internal processing. Therefore the subject of temporal data processing with neural networks draws one towards a study of the brain's inner mechanisms, about which very little is known. For engineering purposes, we hope to understand some of the basic computational principles involved without getting bogged down in the intricate biological details.

Section 2 reviews a simple recurrent network model, which can have highly complex dynamics, and two feedforward models which are inherently static, but never the less have applications in temporal problems. The essentials of how networks are trained are explained, with attention to some major issues which are present whether or not the problem involves time in any important way. Some particular issues and training techniques involving time are explained in section 3. Section 4 introduces one of the most important distinctions in the subject, that between Markovian and non-Markovian problems. Techniques for the easier category, the Markovian problems, are discussed in section 5 with a few practical examples, especially from the speech recognition application. Some of the research areas for non-Markovian problems are introduced in section 6 the final, concluding section 7.

## 2 Basic Neural Network Models

Neural network models specify rules for changing the *output values* of model neurons, or *nodes*, with time. These output values are sometimes regarded as crude models of neural firing rates. Figure 1 and equation (1) illustrate the operation of the most popular type of neural network node. Let $y_{it}$ denote the value of node $i$ at discrete time $t$. In a widely-used class of models, a subset $I$ of the nodes are designated as *inputs*. These are assigned values $Y_{it}$ taken from a model of the network's external environment. The values of the remaining nodes are computed for time $t+1$ from their values at time $t$ according to the rule:

$$y_{i,t+1} = \begin{cases} f\left(\sum_j w_{ij} y_{j,t}\right) & i \notin I \\ Y_{i,t+1} & i \in I \end{cases} \qquad (1)$$

where the *weight* $w_{ij}$ models the strength of a synaptic connection *from* node $j$ *to* node $i$, and $f$ is a differentiable function with constant asymptotes, such as

$$f(x) = 1/(1+e^{-x}) \qquad (2)$$

which varies smoothly from 0 at $-\infty$ to 1 at $\infty$. One input, say node 0, is traditionally assigned the constant value 1.0 so that $w_{i0}$ provides a constant offset or *bias* for the weighted sum computed by node $i$. Through rule (1) the weights specify the network *dynamics*, the manner in which the state (the set of node values) changes with time.

The weighted sum of inputs to a node is called its *activation*. The nonlinearity (2) crudely represents the idea that the firing rate of a neuron should increase with its activation, but should saturate at a maximum value for very large activations and at zero for large inhibition (highly negative activation). Be that as it may, the nonlinearity provides two computationally useful properties, in contrast to linear systems (which can be obtained using $f(x) = x$). One is a guarantee that the node outputs remain bounded, and the other is the ability to approximate a very general class of systems (See Funahashi [8]).

### 2.1 Training

Some network models can be *trained* to produce a desired sequence of *target* values on a subset $T$ of the non-input nodes. The target nodes are also called the *output* nodes, a usage not to be confused with the *output values* possessed by all nodes. The set $H$ of nodes which are neither input nor target nodes are called *hidden* nodes. Let $Y_{it}$ (for $i \in T$) denote the target value for node $i$ at time $t$. (There is no confusion with (1) because $I \cap T = \emptyset$.) A scalar measure of the network's performance is given by

$$E = \tfrac{1}{2} \sum_{i \in T} \sum_t (y_{it} - Y_{it})^2. \qquad (3)$$

If the network operates perfectly, $E = 0$; otherwise $E > 0$.

The sum over time steps in (3) ranges over all time steps for which there is target training data. The precise meaning of this depends on the structure of the training problem.

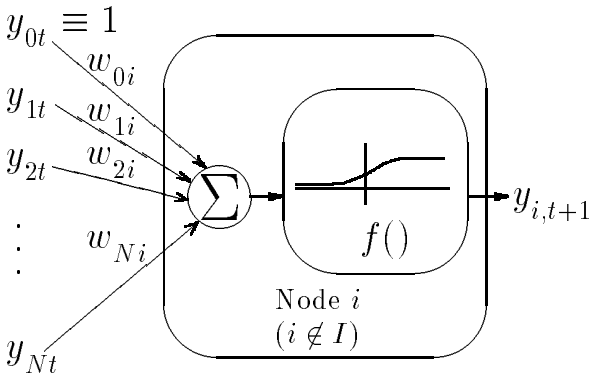


Figure 1: A non-input node of a neural network model containing N+1 nodes, including the bias node.

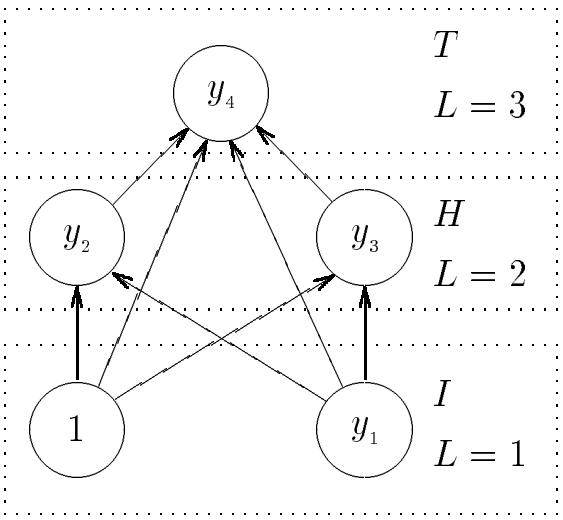


Figure 2: A 3-layer feedforward network with 2 input nodes in the first layer ($I$), one layer of 2 hidden nodes ($H$) , and a third layer with a single target node ($T$). One of the input nodes is a bias node labeled "1". In general there could be any number of hidden layers, including zero. If the weight from node 1 to node 4 were removed, this would be a strictly layered network. Many writers do not count the inputs as a layer, in which case this would be called a 2-layer network. Furthermore the bias node is not normally included when reporting the number of inputs.

Sometimes there is a set of sequences to learn. If the $p^{\underline{\text{th}}}$ of a total of $P$ sequences has $T(p)$ time steps for which there are targets (The net might have to run a few time steps before it produces the first target.), then $\sum_t$ actually means $\sum_{p=1}^{P} \sum_{t=1}^{T(p)}$. Sometimes there is one infinite sequence of data. This can be handled by summing over a finite subsequence, in hopes that it is fairly representative of the whole. In many *on-line* problems the data is generated by sampling a slowly changing environment. By summing over the most recent $T$ samples, where $T$ is a number of timesteps characterising the degree of stationarity of the environment, an appropriate slowly varying performance measure can be obtained. The choice of $T$ in such cases can often be more of an art than a science. Many variations are possible, such as using weight factors with an exponential time dependence.

A popular procedure for training a network is *back propagation of error through time*, which proceeds by computing the derivatives $dE/dw_{ij}$ and using these to incrementally adjust the weights to slightly better values. This procedure is often highly effective, but one is guaranteed neither that a perfect solution $E = 0$ exists, nor that the smallest value of $E$ will be found by this procedure. A *local* minimum can always be found, but not necessarily a *global* minimum. Even if a global minimum is found, there still may be other global solutions which may be preferable for some reason not encoded in (3). In types of problems where $E$ itself is time-varying, little can be said with certainty.

For most real-world problems, a solution with $E = 0$ would not be desirable even if it were obtained, because the resulting network would *generalise* poorly. Usually the training data for a neural network model is selected to be representative of the system to be modeled, but does not exhaustively specify the desired response to every possible input. One simply hopes that the continuity of the functions in (1) will imply that inputs similar to those on which a network was trained will result in outputs similar to those which would be desired. Real-world data sets tend to contain some random noise which an $E = 0$ solution would model precisely, thereby acquiring a poor basis for generalisation. This is called *overfitting*. But even if the data is noise-free, there would typically be an infinite number of possible $E = 0$ solutions, most of which have ridiculous generalisation properties. This problem intensifies as the number of adjustable weights is increased. Choosing the best solutions requires the use of prior knowledge about the properties which characterise "reasonable" solutions for the problem at hand. For example, it might be known that networks whose outputs change the most slowly with respect to their inputs are the best solutions. Both noisy data and prior knowledge can be handled in a Bayesian probabilistic framework which has been specialised to neural networks by MacKay [17], and reviewed briefly by Rohwer [30]. Reasonable results are often obtained with a variety of less complicated heuristics expounded in the neural network textbooks.

### 2.2 Feedforward and recurrent networks

A useful subclass of network models are the *feedforward* networks. Considered as directed graphs, feedforward networks have no cycles. Thus (1) allows no feedback effects whereby the output value of a node at one time can affect its value at a later time. In feedforward networks the nodes can be numbered so that the matrix of weights amongst the non-input nodes is lower triangular. In particular there are no self-connections ($w_{ii} = 0$). A non-feedforward network is called *recurrent*.

Typically the nodes of feedforward networks are arranged in

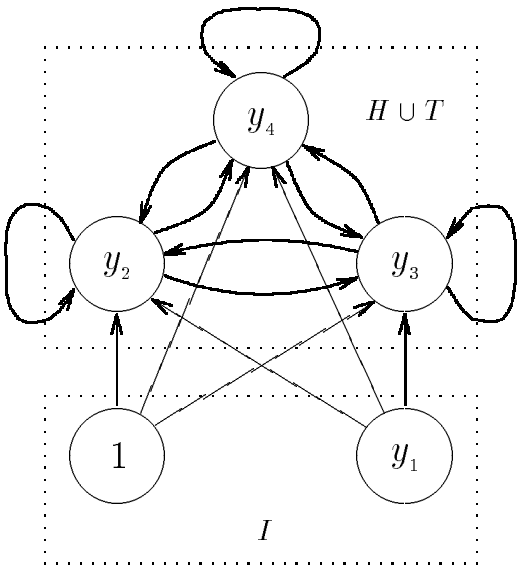


Figure 3: A recurrent network with 2 input nodes, including the bias node indicated by the "1". Any of the 3 non-input nodes could be a target or hidden node.

numbered layers, with nonzero weights from each layer going to higher-numbered layers, but none going in the reverse direction and none going within a layer. The lowest layer is assigned input from the environment. If the input remains fixed in an $L$-layer feedforward net, then every node value in the network remains fixed after $L-1$ time steps. By specifying target values for the highest layer, and using its derivatives to minimise a function similar to (3), these networks can be trained to implement a mapping from input vectors to target vectors which agrees with a set of examples. The hidden nodes in such networks are those in the middle layers between the inputs and targets. Figures 2 and 3 illustrate a layered feedforward network and a recurrent network. A layered feedforward network of the type shown in Figure 2 is also called a *multilayer perceptron (MLP).*

A network is *strictly* layered if, aside from the bias node, any node of layer $L$ has weights leading only to nodes in layer $L+1$. It has been proven by Funahashi [8] and others that a strictly layered feedforward network with a single hidden layer can approximate any continuous mapping to arbitrary accuracy (although a large number of hidden units may be required).

Strictly layered networks also have the sometimes useful property that the state of any hidden layer contains enough information to compute the state of any higher-numbered hidden layer or the target layer. The state of each hidden layer can be interpreted as a re-expression of the input which becomes increasingly appropriate for expressing the targets as the layer number increases.

Note that for fixed inputs, the fixed node values expressed by a strictly layered feedforward network after $L-1$ time steps can be computed by replacing the time index $t$ with the layer index $L$ in (1). The catchphrase expressing this parallel is that a recurrent network can be *unwound in time* to produce a computationally equivalent feedforward network. This observation is useful when developing expressions for the derivatives of (3) used to train a recurrent network.

### 2.3 Radial basis functions

Another popular class of feedforward networks is *radial basis function networks* introduced by Broomhead and Lowe [5]. These networks have one layer of hidden nodes, each of which implements a radial basis function. Hidden node $a$ is assigned a *centre* in the input space having input coordinates $c_{ia}$ and a *radius* $r_a$, and computes an output which is large only for inputs near its centre (on a scale set by its radius). The output of this network is fed through a linear transformation. Specifically, the output $y_{ip}$ produced by example $p$ is

$$y_{ip} = \sum_a w_{ia} g\left(\frac{\sqrt{\sum_i (Y_{ip} - c_{ia})^2}}{r_a}\right) \qquad (4)$$

where $Y_{ip}$ is the input for example $p$ and $g$ is typically a Gaussian $g(x) = e^{-x^2}$. Non-Euclidian distance measures are sometimes used. The loosely-defined region for which a radial basis function has a significant output is its *receptive field.* For the Gaussian, the receptive field is a localised hyper-ellipsoid, in contrast to the linear half-space given by (2). Usually the centres and radii of a radial basis function network are assigned using one of a variety of simple algorithms which take advantage of the locality of the recptive fields to ensure that each input data point falls within just a few receptive fields. Then the weights are adapted to minimise an error measure similar to (3). This problem amounts to solving a large linear system of equations, which can be accomplished using textbook methods [20] much more quickly than a minimisation algorithm can be applied to a multilayer perceptron. This feature is a major attraction of radial basis functions.

The name "radial basis functions" derives from the spherical symmetry of the receptive fields. However this is not an important property of the method and generalisations to less symmetric basis functions are commonly used.

## 3 Training recurrent networks

Any neural network textbook, such as Hertz, Krogh, and Palmer [12] or Beale and Jackson [4], explains the back propagation procedure for training a network, so only a few remarks of special relevance to recurrent networks are worthwhile here. Essentially, back propagation is a clever way of arranging terms to save time and memory in the computation of $dE/dw_{ij}$ from (3) and (1). Although the basic concepts have a long history (See LeCun [16]) and were applied to neural network models as early as 1974 by Werbos [41], the first widely-read treatment was by Rumelhart, Hinton, and Williams in 1986 [31]. This was oriented around feedforward networks, although the possibility of unwinding recurrent networks in time was noted. Recurrent networks for which (1) eventually results in a constant state were of particular interest at that time, and remain an important special case. Treatments of this case based on variations of (3) lacking the sum over time were given by Rohwer and Forrest [28] and Almeida [2]. Almeida arrived at a calculation essentially identical to back propagation through time, and provided a convergence proof which is important for this variation of the method. Rohwer and Forrest's method is also equivalent

to back propagation through time, though less obviously so. The relationship is spelled out by Rohwer and Renals in [29], with a review of Almeida's method.

The sum over time in (3) finds its way into expressions which must be re-evaluated every time the weights are adjusted in the gradient descent procedure. Thus, the entire history of a learning problem should be reviewed many times during training. This is not possible in *on-line* problems, in which an agent only has the opportunity to adapt to environmental data as it arrives. It turns out that this case can be handled by simply not making the change of variables which transforms the direct expression for the derivatives of (3) into the back propagation formulas. This practice was introduced by Robinson [24], and popularised by Williams and Zipser [44], as *Real Time Recurrent Learning*. In typical circumstances it requires more time and memory than back propagation to an extent which is unfortunate but not prohibitive. More recent, separate lines of research by Toomarian and Barhen ([39]) and Schmidhuber ([34]) have raised possibilities for combining the advantages of these methods.

## 4 Recurrent network capabilities in principle and in practice

As noted in section 2.2, feedforward multilayer perceptrons can approximate an arbitrary continuous mapping. Recurrent networks have a further universality property; they can simulate an arbitrary finite state machine. This can be proven on the back of an envelope. It is easy to show that a single node connected to itself can be configured to act as a flip-flop memory element, and equally simple to show that a single node can be configured to perform the NOT-AND Boolean function. It is possible to emulate any computer by building a machine out of these two components, so the result follows. Therefore, in principle the neural network model (1) can do any calculation of practical interest. Furthermore, extensive numerical studies by Renals and Rohwer have shown that this model typically produces complex motion involving long time-scales [21].

Given an infinite number of nodes with which to simulate an infinite Turing tape, a recurrent network model is Turing universal. Furthermore Pollack [19] has shown that models based on a variant of (1) using products of inputs can pack the Turing tape into an infinite-precision real node value, providing Turing universality in a finite network.

Neural networks therefore have the expressive power to unite computation with statistical model fitting. But unfortunately, this does not imply that networks are easily trained from examples to do complex temporal tasks.

### 4.1 Practical difficulties with non-Markovian problems

Most existing training methods work effectively only in what Schmidhuber [33] calls *Markovian* or nearly Markovian environments. This means that the target values at any time step can be determined uniquely from the input and target values from one, or a small number of time steps in the recent past. If the present state of the environment does not contain enough information to enable a unique prediction of the targets at the following time step, then there is no hope unless the hidden nodes happen to encode the missing information. This may be the case if the necessary information lies somewhere in the past, and the network was clever enough to respond to that information by encoding it in some hidden nodes, and to arrange the dynamics so that this information is preserved until it is needed. The task of deciding what the hidden nodes should have done in the past to reduce errors in the future is called the *temporal credit assignment problem* (See Williams [43]).

Thus, the hidden nodes play an essential role in transmitting temporally distant relevant contextual information to the future. This is superposed on the role they already play in feedforward networks, of usefully re-expressing input data in a form amenable for producing the targets. Perhaps it is unsurprising, then, that the most complicated aspects of the back propagation calculation in recurrent networks involve the hidden nodes. To make matters worse, inspection of the derivative formulas shows that unless certain improbable cancellations occur, the expressions for these derivatives are dominated by near-context information. Essentially this is because the near-constancy of (2) over most of it's domain makes it unlikely that the state at one time will be sensitive to small variations of the state at a much earlier time. Hence, the derivatives $dE/dw_{ij}$ will not suggest a weight adjustment which makes use of distant context until an optimum based on recent context has been found to absurdly high accuracy. Temporal credit assignment is done in a manner appropriate only for Markovian, or nearly Markovian environments.

## 5 Techniques for Markovian problems

In order to carry information into the future in a non-Markovian problem, hidden nodes must employ feedback amongst themselves. In Markovian problems the need for feedback disappears with the need for memory. Hence Markovian problems can be handled using feedforward networks, or relatively innocuous forms of feedback.

### 5.1 Teacher Forcing

If the input data at each time step contains enough information to determine the target at that time, then the problem reduces to using a feedforward network to estimate a mapping. Time plays no role except as a pattern label.

If the target data for the previous time step is needed as well, then the network model needs to have feedback from the target nodes. However, for training, the problem can still be reduced to a feedforward network using the *teacher forcing* technique, in which the target nodes are also used as input nodes. A feedforward network is trained to predict the targets at time $t$ based on the inputs at time $t$ and the targets at time $t-1$. When training is finished, the feedforward weights from the target nodes which were masquerading as inputs are used for feedback weights from the ordinary target nodes. If training is reasonably successful, then correct target values at time $t$ will result in correct values at time $t+1$, which in turn give correct values at step $t+2$, etc.

### 5.2 Delay lines

Many prediction problems of interest are sufficiently Markovian to be treated this way. Nearly Markovian problems are often converted into Markovian problems by using *delay lines*. The current state is augmented with copies of the past $\tau_{\max}$ states, so that (1) becomes

$$y_{i,t+1} = \begin{cases} f\left(\sum_{\tau=0}^{\tau_{\max}} \sum_j w_{ij}^{(\tau)} y_{j,t-\tau}\right) & i \notin I \\ Y_{i,t+1} & i \in I \end{cases} . \qquad (5)$$

The delay lines ($w_{ij}^{(\tau)}$, $\tau > 0$) transmit all information $\tau_{\max}$ steps into the future. If distant context is not needed then $\tau_{\max}$ can be relatively small, making this a practical technique. If large $\tau_{\max}$ is required, the network is likely to be overwhelmed with large amounts of redundant and irrelevant information. This problem is compounded by the increased number of variable parameters represented by the extra index on $w$, which lead to poor generalisation as noted in section 2.1. The technique of using *tied weights*, to be explained in section 5.4 can combat this problem.

### 5.3 Prediction with feedforward networks

The teacher forcing idea in various guises has been used to good effect in a variety of prediction problems. Radial basis function networks are popular for these applications because of their training speed, but better results are sometimes obtained from multilayer perceptrons. Lapedes and Farber [15] trained a feedforward network to predict a chaotic time series generated by the Mackey-Glass equation, a scalar delay-differential equation. This has become a standard test problem on which much improved results have since been obtained by, for instance, Plutowski and others [18]. A similar technique was used by Adomaitis, et. al. [1] to predict properties of an electrochemical reaction governed in principle by local differential equations. Many other prediction problems have been modeled in this way. For example Hoptroff [13] discusses prediction of various economic time series and Bulsari and Palosaari report accurate modelling of chemical processes in an adsorption column [6].

### 5.4 Time Delay Neural Networks for speech recognition

A popular technique for reducing the number of adjustable parameters in a network model is *weight tying*. The back propagation procedure can be easily modified to accommodate the constraint that given sets of weights must have the same, unspecified numerical value. Thus the number of adjustable parameters is reduced without reducing the number of weights.

The method makes sense if the problem has symmetries which can justify such constraints. Speech recognition problems provide examples like this. Signal processing techniques can reduce the speech waveform to a time series of vectors thought to express more explicitly the information needed to identify speech units such as phonemes and words. Typically, information from several such vectors is required for phoneme identification. The *time delay neural network (TDNN)* uses delay lines for this purpose, but the term has a more specific technical meaning than just a neural network with delay lines. The structure of a TDNN is illustrated in Figure 4. The delays are grouped into (possibly overlapping) sets, such that the delays in one set are offset by a constant amount from the delays in any other. A separate set of hidden nodes is provided for each set of delays, and each such set of hidden nodes receives input only from the correspondingly delayed inputs. Each resulting set of weights specifies a transformation from input vectors to hidden vectors such that, hopefully, the hidden vectors more explicitly express the information relevant to classifying phonemes. It seems reasonable to assume that the optimal transformation for this purpose will not depend strongly on the overall delay, so the same set of weight values is used for every group. This basic strategy is repeated in a few further layers of hidden nodes. Finally the target nodes receive input from all of the final layer of hidden nodes.

The TDNN has been used with some success by Waibel and others to classify phonemes and words segmented by hand from continuous speech [14, 11, 40]. In combination with dynamic programming it has been used for spotting a small number of words in continuous speech [45].

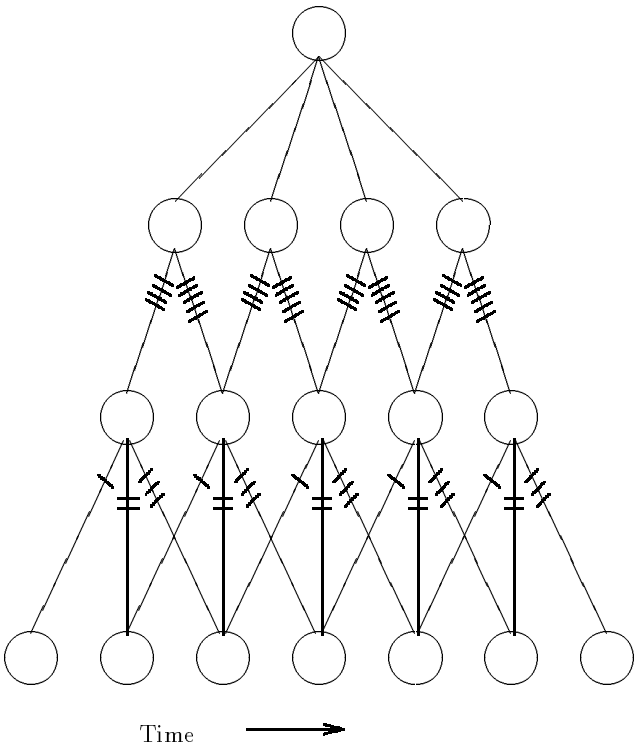


Figure 4: A simplified illustration of the structure of a Time Delay Neural Network (TDNN). Weights with corresponding markings have equal values. Therefore only 9 of the 27 weights shown are independent. The bias weights are not shown. Each node of this diagram would be replaced by a set of nodes in a realistic network.

It is interesting to note a formal similarity between recurrent networks and TDNNs. If a recurrent network is unwound in time, turning time steps into layers, then the weight matrices for each layer are tied to each other. The detailed pattern of tying is different than for the TDNN, however.

### 5.5 A speech recognition system using hidden node feedback

Although it tends to be viewed as somewhat of a last resort, recurrent networks with hidden node feedback have found nontrivial applications. In the example presented here, it is not wildly unrealistic to regard the problem as nearly Markovian, but using recurrence rather than delay lines keeps down the number of parameters and produces excellent results.

Speech recognition presents formidable temporal credit assignment problems because the identity of a word, phoneme or other speech unit present at a given point in time can de-

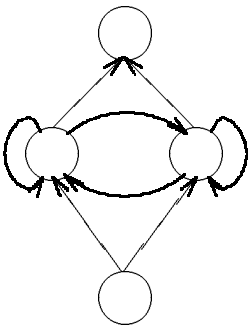

Figure 5: A schematic illustration of a recurrent network architecture used for speech recognition. The bias weights are not shown. The network discussed in the text used 22 input nodes, 176 hidden nodes, and 61 output nodes.

pend on acoustic data from the relatively distant past and near future. Robinson [23, 22] and others have obtained state of the art results using a large recurrent network for this problem. Standard signal processing methods are used to convert the time-domain speech signal into a sequence of 22-component vectors. One vector at a time is presented to the inputs of the neural network, while 61 output nodes are targeted to represent one of 61 sub-phonetic speech units (called *phones*) assigned by a phonetician. As illustrated in Figure 5, there is full feedback among the hidden nodes, but not from the target nodes. Small but important variants of (2) and (3) were used, and considerable effort went into hand tuning the details of the minimisation algorithm.

On a standard database (TIMIT) providing a total of 3360 sentences of training data from 420 speakers and 1680 sentences of testing data from 210 speakers, the system classified about 69% of the speech units accurately. There are considerable subtleties involved in defining what counts as a correct classification because one speech unit can be present for a substantial and variable number of time steps. Furthermore, for speech recognition it only matters that correct speech units should be hypothesised in the correct order at roughly the correct times. In this case a dynamic programming method was used to align the network outputs with the correct transcriptions in order to judge the accuracy.

Results have been published for the TIMIT database with several of the best speech recognition systems in existence, and the recurrent network ranks with the best of them on this task. In particular, the TDNN has not been demonstrated on a comparably difficult task. However, this is not an unqualified victory for recurrent networks. To continue to word and sentence level speech recognition, the recurrent network outputs have to be plugged into a Hidden Markov Model (HMM) system, the traditional way to do speech recognition. The HMM then handles most of the modelling of the temporal properties of the speech.

# 6 Methods for highly non-Markovian problems

Highly non-Markovian problems present severe difficulties, but various lines of research are producing significant inroads into this borderland between statistics and computation. Most lines of inquiry explore methods other than standard back propagation for solving the temporal credit assignment problem. One of these, the *Moving Targets method* adopts a set of variables which make this issue more explicit. Others attempt to blend neural network techniques with standard computational data structures such as stacks.

## 6.1 Moving Targets

The back propagation algorithm for feedforward networks has a natural extension to temporal problems, but it turns out that this algorithm's ability to utilise contextual information diminishes exponentially with time. The "Moving Targets" algorithm of Rohwer [25, 26, 27] reduces this to a linear decrease, but suffers from serious practical difficulties. The basic idea of this algorithm is to treat hidden nodes as target nodes with variable target values. This allows errors to be allocated directly to the hidden nodes, so that the sum in the error measure (3) can be extended to

$$E = \tfrac{1}{2} \sum_{(i\,t)\in T\cup H} \{y_{it} - Y_{it}\}^2. \qquad (6)$$

The "moving target" variables, $Y_{it}$ for $i \in H$, are lumped in with the weights in the minimisation problem; they are initialised randomly and optimised by a derivative-based procedure. If minimisation is successful, the moving targets are discarded and the weights retained. In the course of minimisation, errors on target nodes can be traded for errors on hidden nodes at possibly quite distant time steps if that helps to reduce this sum. This provides greater flexibility in temporal credit assignment than is possible with the standard method in which the weights are the only variables. Figure 6 illustrates this credit assignment mechanism.

The moving targets algorithm has been successfully applied to a problem which requires contextual information from 100 time steps in the past. The training data for this example contains 2 sequences. In each sequence a single input node is given a value of 1.0 at time step 100. It is 0.0 at all other times in sequence 2, and 0.0 at all other times in sequence 1 except at time step 1, when it is 1.0. A single target node is asked to respond with 0.0 at all times for sequence 2, and for time steps 0 to 100 of sequence 1, but with 1.0 after time step 100 in sequence 1. Thus, the input sequences are distinguished only by an event at time 1, and the targets are identical until time 101. Using 1 hidden node it is easy to "hand-wire" a weight matrix which will solve this problem; the hidden node needs to "turn on" in response to the first input 1.0-value in sequence 1, and to stay on (using a positive self-weight) for all time. That way the two sequences will be distinguished at time step 100 by the state of the hidden node. When the moving targets algorithm is applied to this problem, the network quickly adjusts so that the largest errors are on the target nodes at time step 100 in each training sequence. The moving target value of the hidden node settles to 0.5 for most time steps. As training progresses, the moving target values on the hidden nodes at time step 100 increase for sequence 1 and decrease for sequence 2, thereby providing the distinction needed to reduce the target node error at time 101. The moving targets at time step 99 then respond similarly in order to accommodate the errors at step 100. This process carries on until the moving targets are distinguished at time step2,

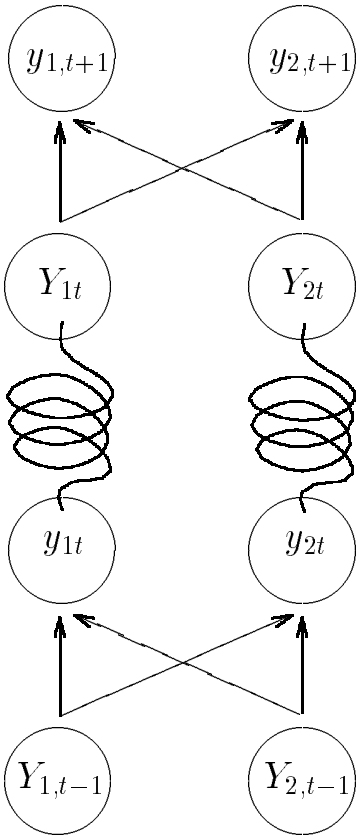


Figure 6: The Moving Targets credit assignment mechanism implied by the error measure (6). A recurrent network is represented as a feedforward network unwound in time. Furthermore each state transition is "separated" from the others by introducing variable "surrogate" training data, indicated by upper-case $Y$. The springs symbolise that error minimisation "pulls" moving targets $Y_{1t}$ and $Y_{2t}$ toward the output values $y_{1t}$ and $y_{2t}$ of the "separate" network representing the previous time step. $Y_{1t}$ and $Y_{2t}$ are also pulled toward values which would reduce the error at time $t+1$. Errors at other time steps do not have any influence via the network, as they would do in back propagation through time, but errors from all time steps appear on an equal footing in (6).

at which point they can be "anchored" on a distinction in the inputs at step 1.

Although this example demonstrates that this algorithm has considerable capabilities for non-Markovian problems, practical experience shows that it has serious disadvantages. In a large problem the minimisation process is beset by a large number of moving target variables which must be optimised. Presumably for this reason, the minimisation proceeds at an impractically slow pace, and local minima are frequently encountered.

### 6.2 Combining neural networks with external memory devices

Giles, Sun, [9, 10] and others have done a substantial series of experiments on training recurrent networks to learn formal grammars from the Chomsky hierarchy. Unsurprisingly, they find that standard architectures such as (1) are limited to learning simple regular grammars, as these only require the use of information distributed over finite, and typically short timescales. They also find that generalisations of (1) involving products of node values are more successful at these tasks. In order to learn context free grammars, which can involve correlations over arbitrary timescales, they have devised a generalisation of a stack which can be expressed in terms of differentiable functions [7]. Thus, a practical trainable Turing machine may be one step closer to reality.

## 7 Conclusions

Simple neural network models have the power to do arbitrary computations with time-varying data, and are amenable to learning from examples. However, existing training methods are either unable to handle problems requiring attention to distant temporal context, or are highly impractical from a computational point of view. Nevertheless there is a usefully large class of problems in which distant temporal context is not particularly important. Methods using techniques for training feedforward networks can be easily applied to this type of problem, and often produce very good results. Recurrent network techniques have been used to a lesser extent in this area, but some of the results are excellent.

Recurrent networks are required for problems in which the network must learn to remember specific information for relatively long periods of time, but standard training methods perform poorly in this situation. Considerable research is now taking place in this area. Perhaps the most promising methods are based on the *Adaptive Critic* concept, in which an auxiliary adaptive learning system, such as an extra neural network, learns to solve the temporal credit assignment problem for another network. After all, having recognised that the temporal credit assignment problem is a fundamental issue for non-Markovian problems, and that neural network models provide trainable learning systems, why not train neural networks to solve temporal credit assignment problems?

Sutton [35] used the term *Adaptive Critic* to describe the use of one linear system (the critic) to provide error information to train another linear system, in this case a controller. The basic idea goes back at least as far as Samuel [32] who used an adaptive method to evaluate board positions in the game of Checkers (also known as Draughts). Drawing on later ideas, particularly Sutton's *Temporal Difference Method* [36] whereby the critic provides part of its own target information with which to train itself, Tesauro [37, 38] has produced a world computer champion Backgammon program. Dynamic programming draws on the same basic idea of using an adaptive evaluation function together with a learning controller (or *policy*), and dynamic programming techniques now play a major part in this line of research. This area is reviewed by Barto [3] and Werbos [42].

Encouraging results are starting to appear, especially when neural networks models are used to model the temporal credit assignment problem itself. Substantial progress in this area may open the door to qualitative advances in the use of neural networks as a vehicle for bringing powerful new inductive, statistical tools to computation and automated reasoning.

## References


[1] R. A. Adomaitis, R. M. Farber, J. L. Hudson, I. G. Kevrekidis, M. Kube, and A. S. Lapedes. Applications of neural nets to system identification and bifurcation anjalysis of real world experimental data. Technical Report LA-UR-90-515, Los Alimos National Laboratory, Los Alamos, NM, 1990.

[2] L.B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In M. Caudill and C. Butler, editors, *IEEE International Conference on Neural Networks*, volume 2, pages 609–618, New York, 1987. (San Diego 1987), IEEE.

[3] A. Barto. Reinforcement learning and adaptive critic methods. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control*, chapter 12. Van Nostrand Reinhold, New York, 1992.

[4] R. Beale and T. Jackson. *Neural Computing: an Introduction*. Adam Hilger, Techno House, Redcliffe Way, Bristol, England, 1990.

[5] D. S. Broomhead and David Lowe. Multi-variable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[6] A. Bulsari and S. Palosaari. Application of neural networks for system identification of an adsorption column. *Neural Computing and Applications*, 1:160–165, 1993.

[7] S. Das, C. L. Giles, and G. Z. Sun. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Indiana University, July 1992.

[8] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183, 1989.

[9] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee. Extracting and learning an unknown grammar with recurrent neural networks. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 317–324, San Mateo CA, 1992. Morgan Kaufmann.

[10] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4:380, 1992.

[11] John B. Hampshire II and Alex Waibel. Connectionist architectures for multi-speaker phoneme recognition. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 203–210. Morgan Kaufmann, San Mateo CA, 1990.

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

[13] R. Hoptroff. The principles and practice of time series forecasting and business modelling using neural nets. *Neural Computing and Applications*, 1:59–66, 1993.

[14] Kevin J. Lang, Alex H. Waibel, and Geoffrey E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3:23–44, 1990.

[15] Alan Lapedes and Robert Farber. How neural nets work. In Dana Z. Anderson, editor, *Neural Information Processing Systems, Denver CO 1987*, pages 442–457. American Institute of Physics, New York, 1988.

[16] Yann Le Cun. A theoretical framework for back-propagation. In David Touretzky, Geoffrey Hinton, and Terrence Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–29. Morgan Kaufmann, San Mateo CA, 1988.

[17] D. MacKay. A practical bayesian framework for back-propagation networks. *Neural Computation*, 4:448–472, 1992.

[18] M. Plutowski, H. White, and G. Cottrell. Learning mackey-glass from 25 examples, plus or minus 2. In *Advances in Neural Information Processing Systems*, volume 5, San Mateo CA, To appear, 1994. Morgan Kaufmann.

[19] J. B. Pollack. On connectionist models of natural language processing. PhD thesis, Urbana: Computer Science Dept., Univ. of Illinois. (Also Tech. Report. MCCS-87-100, Computing Research Lab., NMSU, Las Cruces, NM.), 1987.

[20] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vettering. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK, 1988.

[21] S. Renals and R. Rohwer. A study of network dynamics. *Journal of Statistical Physics*, 58:825–847, 1990.

[22] A. Robinson, M. Hochberg, and S. Renals. IPA: Improved phone modelling with recurrent neural networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 1994.

[23] A. J. Robinson. An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks*, 5, March 1994.

[24] A. J. Robinson and F. Fallside. A dynamic connectionist model for phoneme recognition. In L. Personnaz and G. Dreyfus, editors, *Neural Networks: From Models to Applications*, pages 541–550. I. D. S. E. T., Paris, 1989.

[25] R. Rohwer. The 'moving targets' training algorithm. In L. B. Almeida and C. J. Wellekens, editors, *Lecture Notes in Computer Science 412, Neural Networks*, pages 100–109. Springer-Verlag, Berlin, 1990.

[26] R. Rohwer. The 'moving targets' training algorithm. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 558–565, San Mateo CA, 1990. Morgan Kaufmann.

[27] R. Rohwer. The 'moving targets' training algorithm. In J. Kindermann and A. Linden, editors, *Distributed Adaptive Information Processing (DANIP)*, pages 175–196, Munich, 1990. R. Oldenbourg Verlag.

[28] R. Rohwer and B. Forrest. Training time-dependence in neural networks. In M. Caudill and C. Butler, editors, *IEEE International Conference on Neural Networks*, volume 2, pages 701–708, New York, 1987. (San Diego 1987), IEEE.

[29] R. Rohwer and S. Renals. Training recurrent networks. In L. Personnaz and G. Dreyfus, editors, *Neural networks from models to applications*, pages 207–216. I. D. S. E. T., Paris, 1988.

[30] R. Rohwer, M. Wynne-Jones, , and F. Wysotzki. Neural networks. In D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors, *Machine Learning, Neural and Statistical Classification*, chapter 6, pages 84–106. Prentice-Hall, 1994.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Procressing*, volume 1, pages 318–362. MIT Press, Cambridge MA, 1986.

[32] A. Samuel. Some studies in machine learning using the game of Checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.

[33] J. Schmidhuber. Making the world differentiable: On using self-supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments. Technical Report FKI-126-90, Technische Universitat Munchen, Institut fur Informatik, Arcisstr. 21, 8000, Munchen 2, Germany, 1990.

[34] J. Schmidhuber. A fixed size storage $o(n^3)$ time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4:243–248, 1992.

[35] R.S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, 1984.

[36] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[37] G. J. Tesauro. Practical issues in temporal difference learning. Technical Report RC 17223 (#76307) 9/30/91, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA, 1991.

[38] G. J. Tesauro. Td-gammon a self-teaching backgammon program achieves master-level play. *Neural Computation*, To appear.

[39] N. Toomarian and J. Barhen. Adjoint-functions and temporal learning algorithms in neural networks. In R. P. Lippmann J. E. Moody and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 113–120, San Mateo CA, 1991. Morgan Kaufmann.

[40] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37:328–339, 1989.

[41] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[42] P. Werbos. Approximate dynamic programming for real-time control and neural modelling. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control*, chapter 13. Van Nostrand Reinhold, New York, 1992.

[43] R. Williams. Towards a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Computer Science, Northeastern University, Boston, MA, 1988.

[44] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

[45] T. Zeppenfeld and A. Waibel. A hybrid neural network dynamic programming word spotter. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 1992.