The Autoregressive Backpropagation Algorithm*

Russell R. Leighton and Bartley C. Conrath
The MITRE Corporation
7525 Colshire Drive, McLean, VA 22102

Abstract

This paper describes an extension to error backpropagation [7] that allows the nodes in a neural network to encode state information in an autoregressive "memory." This neural model gives such networks the ability to learn to recognize sequences and context-sensitive patterns. Building upon the work of Wieland [9] concerning nodes with a single feedback connection, this paper generalizes the method to n feedback connections and addresses stability issues. The learning algorithm is derived, and a few applications are presented.

Introduction

The commonly-used error backpropagation (BP) algorithm contains no "memory," hindering the learning of temporal patterns. The addition of an autoregressive (AR) memory to individual nodes allows the network to encode state information. Autoregressive backpropagation networks are recurrent networks where the nodes only have *linear* feedback to their individual outputs, unlike other recurrent networks [1,2,5,6,8,11]. This effort expands upon the work done by Wieland [9] on nodes with a single feedback connection, by generalizing the method to n feedback connections and addressing stability issues.

Node Model

In backpropagation learning [7], the inputs to a neuron are multiplied by feedforward weights and summed, along with a node bias term. The sum is then passed through a smooth sigmoidal transfer function, producing the neuron's output value. This neural model has no memory, because the output value is independent of previous inputs or outputs.

The memoryless model described above has been extended to include an autoregressive memory, $m_i(t)$ (figure 1), a form of self-feedback where the output depends on the current output plus a weighted sum of previous outputs. The AR node equations are:

$$
\begin{aligned}
o_i(t) &= f(net_i(t)) + m_i(t) \\
net_i(t) &= bias_i + \sum_{j=1}^{ninputs_i} w_{j,i} o_j(t) \\
f(x) &= \frac{1}{1+e^{-x}} \\
m_i(t) &= \sum_{n=1}^{order_i} a_{i,n} o_i(t-n),
\end{aligned}
$$

*To appear in the Proceedings of the International Joint Conference on Neural Networks, 1991

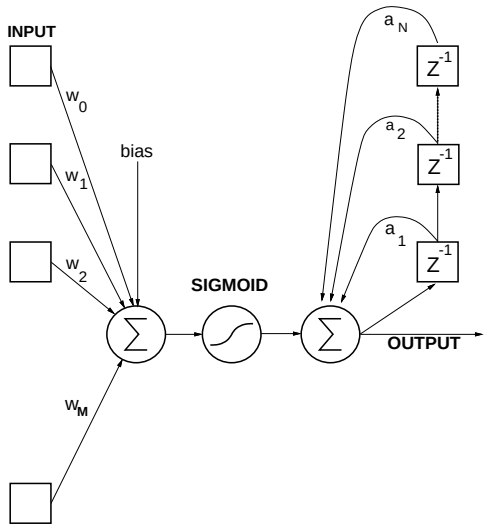

Figure 1: AR Model Neuron

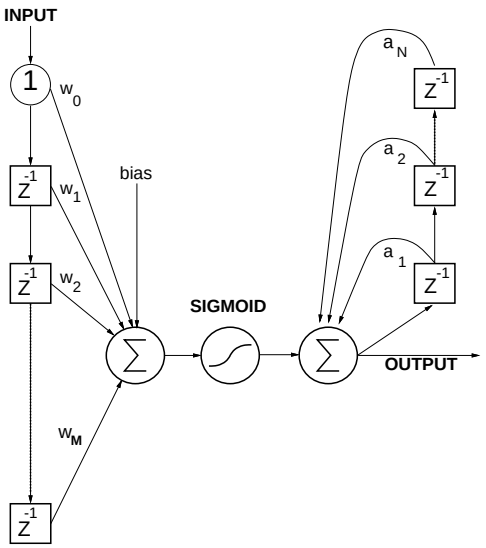

Figure 2: AR Model Neuron as a Digital Filter

where $o_i(t)$ is the output of node o_i at time t, $w_{j,i}$ is the feedforward weight from node j to node i [1], $a_{i,n}$ is the feedback weight from the nth delay of node i, $ninputs_i$ is the number of input nodes to the ith node and j is an index through this set of input nodes, and $order_i$ is the number of feedback delays for the ith node. The feedback weights ($a_{i,n}$) and the feedforward weights ($w_{j,i}$) are adaptive. The backpropagation learning algorithm can be modified such that each model neuron can change its $w_{j,i}$'s and $a_{i,n}$'s to minimize the output error. In this way, networks can "remember" past events and make context-sensitive decisions.

Analysis of the Node Model

Figure 2 depicts an AR node viewed as a digital filter by considering the input layer as a tapped delay line. This realization makes the AR node equations similar to the standard ARMA (autoregressive-moving average) difference equation:

$$y(n) \quad = \quad w_0 x(n) + w_1 x(n-1) + \cdots + w_M x(n-M) + a_1 y(n-1) - \cdots a_N y(n-N). \qquad (1)$$

The fundamental difference between an AR node and the linear filter above is the sigmoid nonlinearity. The following analysis assumes a sigmoid nonlinearity centered at zero (e.g. $f(x) = \frac{1}{1+e^{-x}} - 0.5$). By carefully choosing the feedforward and feedback weight values, this model neuron can perform linear, as well as nonlinear processing (figure 3). Figure 4 depicts a model neuron with one input and the bias set to zero. This model uses different values of the feedforward weight, w_0, along with the autoregressive weights, a's, to characterize possible memory encodings. The analysis is segmented into three special cases: w_0 is very small such that the input to the sigmoid falls in the linear region, w_0 is very large such that the input to the sigmoid falls in the clipped region, and w_0 is such that the input to the sigmoid falls in the curved region.

Very Small Input to the Sigmoid

If the constant w_0 is very small, then the scaled input signal will intercept the sigmoid function in the linear region. Therefore, the output from the sigmoid will be approximately equal to the input scaled

[1] Node i must be contained in a layer farther from the input than the layer containing node j.

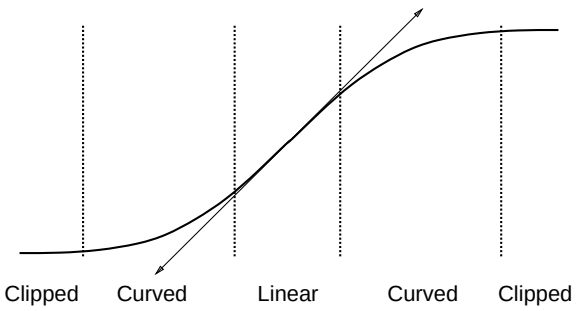

Figure 3: Regions of the Sigmoid

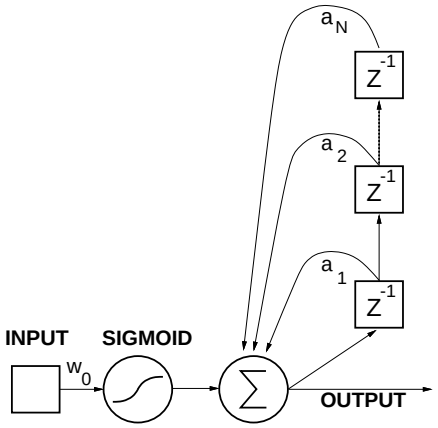

Figure 4: A Simplified AR Neuron

by a very small constant. For simplicity, the constant w_0 will be set to one. Equation 1 can be modified to read:

$$y(n) \quad = \quad x(n) + a_1 y(n-1) + \cdots + a_N y(n-N).$$

Scaling the input to continuously intercept the linear region results in a linear autoregressive network.

Very Large Input to the Sigmoid

If $|w_0|$ is very large, then the neuron simply clips the signal. There are two cases where clipping occurs. First, if the product of w_0 and the input signal is very large, then the output of the sigmoid remains on the flat parts of the sigmoid. In other words, if the sigmoid extends from $-A$ to $+A$ and w_0 is very large, then the the node output appears to be a constant A times a step function $u(n)$. The input is assumed to be large and positive in this example. By substituting the input into equation 1, the following difference equation can be written:

$$y(n) \quad = \quad Au(n) + a_1 y(n-1) + \cdots + a_N y(n-N).$$

The second case is when the input to the sigmoid times w_0 is very large and intersects the sigmoid at alternating maximum or minimum regions. In this case, the output of the sigmoid looks like a summation of step functions, alternating between $-A$ and $+A$ with zero crossings.

Interceptions on the Nonlinear Regions of the Sigmoid

The extreme cases illustrate the similarity of the AR network with conventional digital filters. The power of the neural network approach becomes apparent when the input to the sigmoid falls in the nonlinear region. A single equation representing the output of the sigmoid for an arbitrary input is not possible when the input to the sigmoid falls in the nonlinear region, so an analysis will not be given here. The nonlinear nature of the AR node is analogous to the nonlinear nature of the standard backpropagation node. The curved region allows curved decision surfaces [3,10]. Therefore, the AR node has the same benefits as the standard BP node with the additional advantage of memory.

Learning Algorithm

Error backpropagation is a learning algorithm for feedforward neural networks that attempts to minimize the mean squared error on the output by adjusting the feedforward weights. AR backpropagation

extends this algorithm to discrete time-varying systems by including feedback weights that are also adjusted to minimize the mean squared error. The output error measure is defined as:

$$\begin{aligned} E &= \sum_{t=T_0}^{T_1} E(t) \\ E(t) &= \frac{1}{2}\sum_i (d_i(t) - o_i(t))^2, \end{aligned}$$

where o_i is the value of the ith output node and d_i is the desired value of the ith output node. The weight changes are accumulated between T_0 and T_1. The weights are then changed and the process is repeated. The interval from T_0 to T_1 is called the *update interval.*

Gradient Descent

In order to perform gradient descent in weight space, each feedforward weight, $w_{j,i}$, and each feedback weight, $a_{i,n}$, need to change proportionally to the negative of the gradient with respect to that weight:

$$\begin{aligned} \Delta w_{j,i} &= -\alpha \frac{\partial E(t)}{\partial w_{j,i}} \\ \Delta a_{i,n} &= -\alpha \frac{\partial E(t)}{\partial a_{i,n}} \end{aligned}$$

where α is a constant of proportionality.

The partial derivatives with respect to the feedforward weights can be expanded to:

$$-\frac{\partial E(t)}{\partial w_{j,i}} = -\frac{\partial E(t)}{\partial o_i(t)} \frac{\partial o_i(t)}{\partial w_{j,i}}$$

where

$$\begin{aligned} \frac{\partial o_i(t)}{\partial w_{j,i}} &= \frac{\partial f(net_i(t))}{\partial w_{j,i}} + \frac{\partial m_i(t)}{\partial w_{j,i}} \\ &= \frac{\partial f(net_i(t))}{\partial net_i(t)} \frac{\partial net_i(t)}{\partial w_{j,i}} + \frac{\partial m_i(t)}{\partial w_{j,i}} \\ &= \acute{f}(net_i(t)) o_j(t) + \sum_{n=1}^{order_i} a_{i,n} \frac{\partial o_i(t-n)}{\partial w_{j,i}}. \end{aligned}$$

The *credit* for an output node o_i at time t is defined as:

$$\delta_i = -\frac{\partial E(t)}{\partial o_i(t)} = d_i(t) - o_i(t).$$

Brute force calculation of the δ_i for a hidden node is computationally undesirable. However, assuming that the error at the output changes slowly, the recursive calculation used in standard backpropagation may be used. If the error changes slowly, the δ_i for a hidden unit o_i is:

$$\delta_i = -\frac{\partial E(t)}{\partial o_i(t)} \approx -\sum_{\kappa=1}^{noutputs_i} \frac{\partial E(t)}{\partial o_\kappa(t)} \frac{\partial o_\kappa(t)}{\partial o_i(t)} = \sum_{\kappa=1}^{noutputs_i} \delta_k \frac{\partial o_\kappa(t)}{\partial o_i(t)}.$$

The index of a node receiving input from o_i is represented as κ. The partial derivative, $\frac{\partial E(t)}{\partial o_\kappa(t)}$, has been calculated previously. The partial derivative, $\frac{\partial o_\kappa(t)}{\partial o_i(t)}$, is calculated as :

$$\begin{aligned} \frac{\partial o_\kappa(t)}{\partial o_i(t)} &= \frac{\partial f(net_\kappa(t))}{\partial o_i(t)} + \frac{\partial m_\kappa(t)}{\partial o_i(t)} \\ &= \acute{f}(net_\kappa(t)) w_{i,\kappa} + \sum_{n=1}^{order_i} a_{i,n} \frac{\partial o_\kappa(t-n)}{\partial o_i(t)}. \end{aligned}$$

Therefore, $w_{j,i}$ is updated to minimize E:

$$\Delta w_{j,i}(t) = \alpha \left(-\frac{\partial E(t)}{\partial o_i(t)} \right) \frac{\partial o_i(t)}{\partial w_{j,i}} = \alpha \delta_i \left[\acute{f}(net_i(t)) o_j(t) + \sum_{n=1}^{order_i} a_{i,n} \frac{\partial o_i(t-n)}{\partial w_{j,i}} \right]$$

$$w_{j,i}^{new} = w_{j,i}^{old} + \frac{1}{T_1 - T_0} \sum_{t=T_0}^{T_1} \Delta w_{j,i}(t).$$

The $bias_i$ terms are considered feedforward weights from nodes with values of 1.0 and updated by the above equations.

The partial derivative, $-\frac{\partial E(t)}{\partial a_{i,n}}$, can be expanded to:

$$-\frac{\partial E(t)}{\partial a_{i,n}} = -\frac{\partial E(t)}{\partial o_i(t)} \frac{\partial o_i(t)}{\partial a_{i,n}},$$

where

$$\frac{\partial o_i(t)}{\partial a_{i,n}} = \frac{\partial f(net_i(t))}{\partial a_{i,n}} + \frac{\partial m_i(t)}{\partial a_{i,n}}$$

$$= o_i(t-n) + a_{i,n} \frac{\partial o_i(t-n)}{\partial a_{i,n}}.$$

The δ_i for o_i has already been calculated above. Therefore, $a_{i,n}$ is updated to minimize E:

$$\Delta a_{i,n}(t) = \alpha \left(-\frac{\partial E(t)}{\partial o_i(t)} \right) \frac{\partial o_i(t)}{\partial a_{i,n}} = \alpha \delta_i \left[o_i(t-n) + a_{i,n} \frac{\partial o_i(t-n)}{\partial a_{i,n}} \right]$$

$$a_{i,n}^{new} = a_{i,n}^{old} + \frac{1}{T_1 - T_0} \sum_{t=T_0}^{T_1} \Delta a_{i,n}(t).$$

Stability Issues

The AR node is asymptotically stable if the poles of the feedback transfer function are located within the unit circle of the Z-plane. The AR1 node has only one pole, which is identical to the feedback weight. Therefore, the stability criterion is:

$$|a_{i,1}| < 1.$$

Feedback weights are initialized to small random values that do not violate the stability criterion. At the end of an update interval, if $a_{i,1}^{new}$ violates the stability criterion, it is recalculated. The change from $a_{i,1}^{old}$ to $a_{i,1}^{new}$ is halved until $a_{i,1}^{new}$ satisfies the stability criterion. From a computational standpoint, it is not desirable to compute the poles for higher-order nodes (i.e., AR2+). Stability checks can be made using only the feedback weights and their respective desired changes. These tests are based upon the Routh-Hurwitz stability criterion [4]. As in the first-order case, if the stability criterion is violated, the changes in the feedback weights are halved until the resulting weights satisfy the stability criterion.

Weight Update Issues

The evaluation of $\frac{\partial o_i(t)}{\partial w_{j,i}}$ during learning represents the effect of $w_{j,i}$ through past time. Similarly, $\frac{\partial o_i(t)}{\partial a_{i,n}}$ represents the effect of $a_{i,n}$ through past time. At the end of each update interval, the weights are changed. The partial derivatives[2] associated with each adaptable parameter are set to zero to ensure that the new sums reflect only the effect of the current update interval. The network can be tuned to patterns of a particular duration by choosing an update interval roughly equal to the duration of the patterns of interest. In order to improve convergence, the AR backpropagation equations can be modified to include a momentum term [7]. Momentum was not used in the following examples.

[2] The partial derivatives are implemented as running sums.

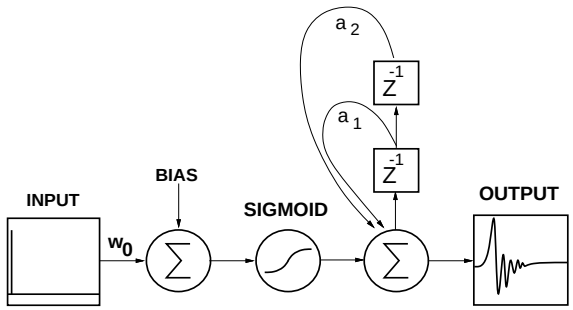

Figure 5: The Ringing Network

Example Experiments

In the following simple experiments, some of the properties of AR backpropagation are illustrated. The ringing experiment illustrates the use of AR backpropagation to learn an impulse response. The prediction experiment shows the ability of the AR node to predict a path based on past values when trained with an example path. The sequence experiment illustrates the ability to exchange input space for feedback connections. Although the network architecture and weights needed to solve each example problem was known *a priori*, these experiments remain interesting because they illustrate the use of AR backpropagation to *learn* the correct weights[3].

Ringing

The ringing problem can be performed successfully using a linear ARMA model. This example illustrates that an AR neural network can learn to perform the same tasks as a conventional, linear ARMA system. The task for the network is to learn to "ring" given an impulse. A single node network is used in this problem with one input (figure 5). The node has order two (i.e., two feedback connections) with a sigmoid output range of [-0.5,0.5]. The input signal is an impulse, consisting of a unit-valued zero-time sample and zero-valued subsequent samples. The training signal is an exponentially decaying sinusoid, consisting of 128 sample points.

The network was trained using a learning rate of $\alpha = 0.5$ and an update interval of 64. The network, by definition, converged if the magnitude of the error was less than 0.01. The network converged in 2915 epochs. Using a less conservative convergence criterion of 0.05, the network converged in 850 epochs. This problem illustrates the network's ability to effectively learn to use the linear region of the sigmoid.

Trajectory Prediction

The ability of AR neural networks to utilize past input values is very helpful in prediction problems. The recursive architecture of the AR1 neural network is better suited to address this inherently recursive problem than the architecture of the standard BP network (AR0). The problem is to predict the next sample point given the current location on a spiral with a radius decreasing by fixed amounts:

$$r(t) \quad = \quad r(t-1) - 0.0625, \; \textit{Radius of spiral}.$$

All patterns were scaled within -1 and +1. The samples were taken coarsely, every $\pi/8$ radians. All networks contained two inputs that accepted the current x, y coordinate and two outputs that provided the predicted x, y coordinate. These networks did not contain a hidden layer, due to the simplicity of the problem. The order of the output nodes was either zero (i.e., no feedback) or one. Both the AR1

[3] Feedforward and feedback weights were initialized from a uniform distribution in the range [-0.5,0.5]

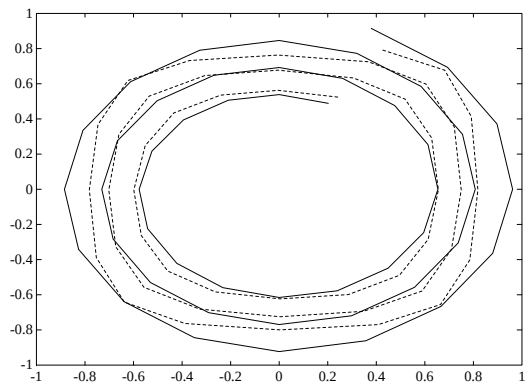

Figure 6: Results of AR0 (dashed line)

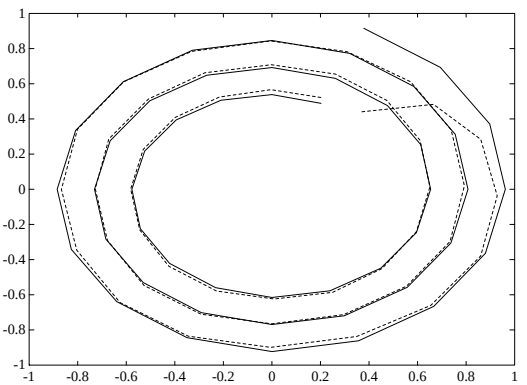

Figure 7: Results of AR1 (dashed line)

and AR0 networks were trained for 5000 iterations with a learning rate of 0.05, using an update interval of 13.

The results are shown in figures 6 and 7. Unlike the boxy prediction of the AR0 network, the AR1 neural network predicted the next location on the path with greater accuracy after some charge-up time, typically lasting 3 to 4 samples. This charge-up time is an artifact to consider when using AR neural networks. The feedback weights cause inaccurate responses for the first few samples, until the feedback delays contain enough past values to contribute positively to the output.

Learning Sequences

An important property of the AR neural network is the ability to learn to recognize both spatial and temporal patterns. The following experiments are a series of sequence recognition tests that illustrate the spatio-temporal nature of an AR backpropagation network. This problem requires the neural network to compute 1-to-n mappings, by responding differently to the identical input in different contexts. The network must discriminate using previous values stored in the delays of the feedback connections. The deterministic task is to learn to recognize **A**=>**B**=>**C** (sequence) from any other combination of **A**'s, **B**'s, and **C**'s (non-sequence). The network receives one input token at a time:

$$A = [\,1\,0\,0\,] \quad B = [\,0\,1\,0\,] \quad C = [\,0\,0\,1\,].$$

The network is trained such that the output is high at the end of the sequence **A**=>**B**=>**C** and low otherwise. This problem requires a network with hidden layers.

The network architecture consisted of an output node connected to a hidden layer of three nodes that connect fully to three inputs. The sigmoid range was [-0.5, 0.5]. Four networks using different AR nodes were simulated: one with no feedback (AR0), one with nodes having one feedback weight (AR1), one with nodes having two feedback weights (AR2), and one with nodes having three feedback weights (AR3). This architecture was chosen, because the feedforward and feedback weights for this problem can be calculated by hand without learning. This experiment tests the ability of the learning algorithm to develop a set of parameters that solves the problem from random initial conditions. A standard BP network can be used for this task, if the input layer is a window, receiving three input tokens at a time. However, if the length of the sequences are very long or variable, the standard BP approach does not scale well. AR backpropagation trades input space for feedback connections, eliminating this problem.

AR0, AR1, AR2, and AR3 networks were trained using a learning rate of 0.01 and an update interval of 2 on a sequence set consisting of combinations of **A**, **B**, and **C**, separated by padding[4]. The correct sequence was embedded twice within the training set of 38 samples. The training signal was low (-0.5)

[4] Padding consisted of zero-valued input patterns: 0,0,0.

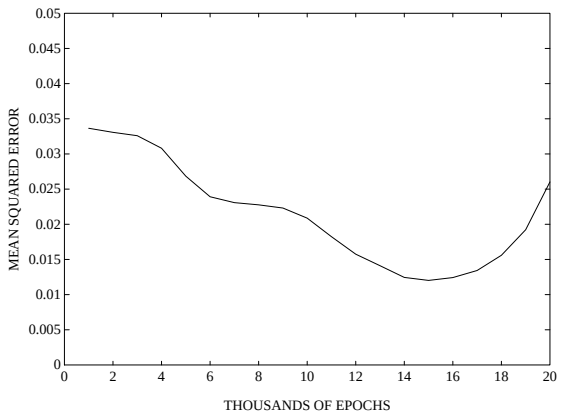

Figure 8: Training Data - Non-sequence

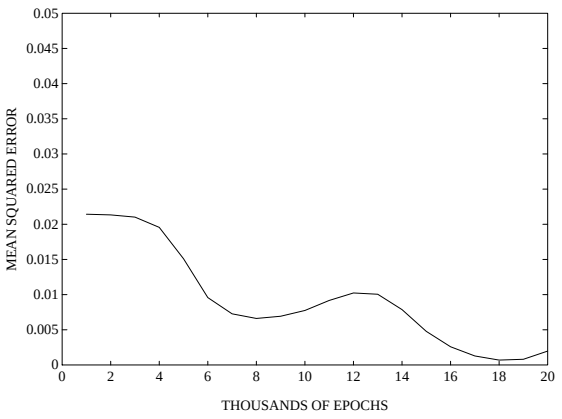

Figure 9: Training Data - Sequence

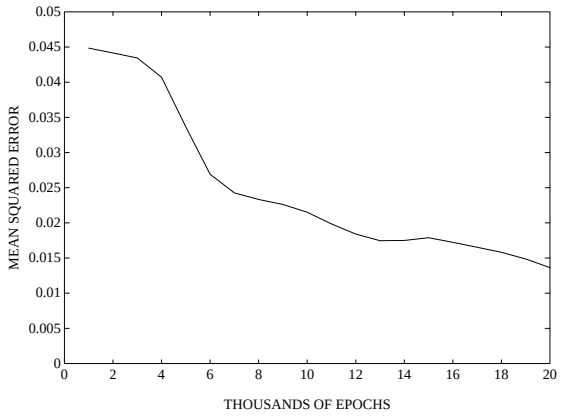

Figure 10: Testing Data - Non-sequence

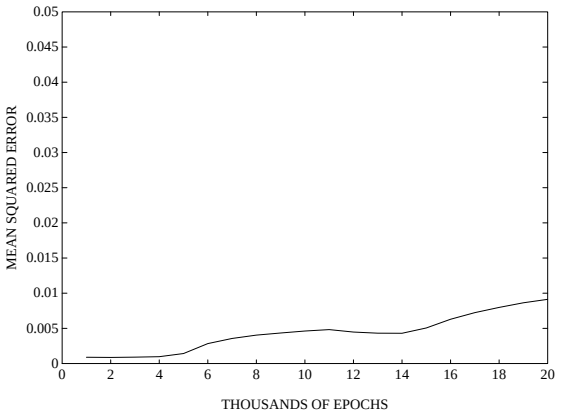

Figure 11: Testing Data - Sequence

AR3 Network Performance 1k-20k Epochs

for all input patterns, except the sequence **A**=>**B**=>**C**. For the sequence **A**=>**B**=>**C**, the training signal was: **A**, -0.5; **B**, 0.0; and **C**, 0.5. This ramped training signal, emphasizing **B**, was used to assist learning of the correct sequence by providing context within the training signal.

The networks were trained for 1000, 2000, and 20,000 epochs. Successful training was determined by separation of the two classes of data: sequence or non-sequence. If the output of the network can be thresholded such that all of the sequences can be detected with no false alarms[5], then the data are said to be separated by the network. Not surprisingly, the AR0 network consistently failed to separate the data, while the AR1, AR2, and AR3 networks were able to widely separate the data for each training length.

A previously unseen test set of 86 samples, containing three instances of the correct sequence, was presented to the trained networks. The AR0 network failed to separate the data for all training intervals. After 1000 epochs, the AR1, AR2, and AR3 networks separated the correct sequences from the non-sequences. After 2000 and 20,000 epochs, the AR1, AR2, and AR3 networks *further* separated the training data. However, the networks overfitted the training data and failed to separate one of the three correct sequences in the test set. The AR3 network overfitted the training data more than the other networks.

[5] In this context, a false alarm is defined as the incorrect classification of a non-sequence as a sequence.

Figures 8-11 plot the performance of the AR3 network measured as mean squared error (MSE) versus training epoch. The MSE is plotted for the non-sequences and the sequences during training and testing. The plots begin at epoch 1000, where the network can separate both the training and the testing data. As the training continues, the combined MSE for non-sequence and sequence data decreases. The MSE abruptly rises in figure 8 during epochs 15,000-20,000, and abruptly falls in figure 9 during epochs 15,000-20,000, as the network trades false alarm performance (non-sequence MSE) for better detection performance (sequence MSE). At epoch 20,000, the AR3 network has learned the training set extremely well. However, figure 11 illustrates that the network has overfit the data. The MSE on sequences in the test set rises as learning continues, illustrating the trade-offs of using higher-order networks. If the training data are not a representative set, then higher-order networks may generalize more poorly than lower-order networks. However, if the training data are a representative set, then higher-order networks may generalize well, by exploiting complex temporal relationships in the training set.

Conclusions

AR neural networks offer several advantages over standard BP networks. The ability to trade input space for the feedback connections results in smaller networks. This property is very important in sequence recognition, where the number of input samples can grow very large. AR backpropagation networks also perform context-sensitive tasks, such as 1-to-n mappings, while standard BP cannot. Two important considerations, when using AR backpropagation, are the optimal order for the various nodes and data set structuring. As the order of a node increases, the network may respond with more variation due to the increased number of degrees of freedom. This property allows higher-order networks to learn the training set better than lower-order networks. However, this property can also cause higher-order networks to overfit the training set, reducing the ability to generalize. Since the past inputs influence the AR neural network, the relative position of patterns within the training set becomes significant, especially in sequence recognition problems. AR neural networks show considerable promise as a robust method for detection/classification of time-varying signals. Our current research focuses on the application of AR backpropagation to speech and sonar signal processing.

References

[1] L. B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proceedings of the International Joint Conference on Neural Networks*, 1987.

[2] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins. Learning and sequential decision making. Technical Report COINS Technical Report 89-95, Massachusetts Institute of Technology, 1989.

[3] K. Lang and M. Witbrock. Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Models Summer School*, 1988.

[4] S. Pandit and S. Wu. *Time Series and System Analysis with Applications.* John Wiley and Sons, 1983.

[5] B. Pearlmutter. Dynamic recurrent neural networks. Technical Report CMU-CS-90-196, Carnegie-Mellon University, 1990.

[6] F. J. Pineda. Generalization of backpropagation to recurrent and higher order neural networks. In *Proceedings of the IEEE Conference on Neural Information Processing Systems*, 1987.

[7] D. Rumelhart, J. McClelland, et al. *Parallel Distributed Processing.* MIT Press, 1986.

[8] P. Werbos. Backpropagation through time: What it does and how to do it. In *Proceedings of the IEEE*, pages 1550–1560, 1990.

[9] A. Wieland. Networks of nodes with adaptable time constants. Technical Report MP-90W00048, The MITRE Corporation, McLean, Virginia, 1990.

[10] A. Wieland and R. Leighton. A geometric analysis of neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, 1987.

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