Training Multi-loop Networks

Roger L. Schultz, roger.schultz@halliburton.com, Halliburton Energy Services
Martin T. Hagan, mhagan@master.ceat.okstate.edu, Oklahoma State University
Orlando De Jesus, dorland@okstate.edu, Oklahoma State University

Abstract

In this paper we investigate the training of time-lagged recurrent networks having multiple feedback paths and tapped-delay inputs. Network structures of this type are useful in approximating nonlinear dynamical systems. The introduction of additional feedback loops into a network structure may improve the modeling capability of the network, but a significant price can be paid in complexity and computational burden when calculating the dynamic derivatives needed for training. The focus of this paper is on the calculation of the dynamic derivatives which must be determined or approximated in order to use any of the popular methods employed in training neural networks. In this paper we illustrate the effect of multiple feedback loops on the formulation of the equations needed for calculating the dynamic derivatives. We also investigate the effect on network performance and computational complexity when various dynamic derivative approximations are used in training multiple feedback loop networks.

1. Introduction

There is much interest in applying neural networks to solve complex problems in areas such as controls and signal processing. These applications often require neural network modeling or control of real dynamic systems which can only be accomplished by using recurrent network structures which have dynamic modeling capabilities. In working with these systems it is generally necessary to construct neural models having multiple connections which appear as feedback loops or feed-forward paths. Additionally, these connections often contain tapped-delay lines. The presence of feedback loops or feed-forward connections containing tapped-delays necessitates time-based computation of *dynamic derivatives* [1]. This paper focuses on how the dynamic derivative equations are determined for multi-loop networks and what effect they have on computational burden for training. The first section of this paper describes time-lagged recurrent networks, dynamic backpropagation, and presents some multi-loop network examples. Descriptions for the associated dynamic derivative equations are given as well. In the next section we discuss the impact of feedback loops and tapped-delay lines on computational and storage requirements. Finally, simulation results for a multi-loop network simulation using full dynamic backpropagation and two derivative approximations are presented.

2. Time-lagged Recurrent Networks

Neural networks can assume many different structures. Static network structures contain only "forward" connections in which the output of a particular layer is always applied as input to a subsequent layer, or in the case of the last layer, becomes the network output. In static networks the network output is only a function of the current inputs, and is not dependent on previous network inputs. If however, the output of any layer in a network is used as feedback input to that layer or any previous layer, the network is not static, but is dynamic in nature, and is commonly called a recurrent network [2]. In a recurrent network the current output is dependent on previous network outputs. Many real systems exhibit dynamic behavior, and thus there is a good deal of interest in using recurrent neural networks as system models in many applications. There is an analogous relationship between FIR and IIR systems models and static and recurrent neural networks respectively. Static neural networks can be considered a subset of recurrent networks. In filtering and controls applications it is common to use tapped-delay lines to construct the inputs to a neural network both for input sequences and in feedback loops. Figure 1 shows an example recurrent network with tapped-delay inputs.

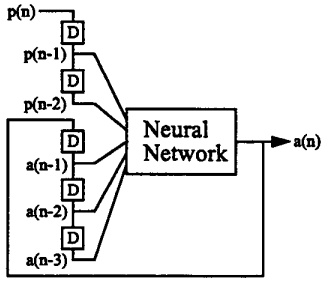

Figure 1 Time-lagged Recurrent Network

The box labeled "Neural Network" contains a neural network with a feed-forward structure which will be referred to as a "subnet" throughout the remainder of this paper. In some applications two or more subnets are connected by

0-7803-5529-6/99/$10.00

tapped-delay lines either directly as the output of one subnet forming the input to another subnet, or through a feedback loop which may direct output from one subnet back as input to another subnet. The presence of multiple feedback loops connecting several subnets can create a complex overall network structure. In static networks containing no feedback, standard backpropagation [3] may be used to compute the gradients necessary for network training. This is not true for recurrent networks. The gradients of networks which have feedback loops must be computed using dynamic backpropagation. Dynamic backpropagation takes into account the time dependence of the network output on previous outputs. The complexity of the dynamic derivatives and the computational burden associated with computing them increases with the addition of feedback loops in a network structure. This will be shown in the following sections.

2.1 Dynamic Backpropagation

In training a neural network using supervised learning the goal is usually to minimize a cost function V, by adjusting the network weights in an appropriate manner as to accomplish the minimization. In the case of a recurrent network, the cost function must reflect the dynamic nature of the function represented by the network. It can be written more descriptively as $V(\underline{a}_i(\underline{w}), \underline{w})$, where $\underline{a}_i$ is the network output at time i, and $\underline{w}$ represents the network weights. This representation shows the time-dependent qualities of the cost function. In order to properly adjust the network weights we must compute the derivative of $V(\underline{a}_i(\underline{w}), \underline{w})$ with respect the network weights. Because each subsequent output of a recurrent network is a function of the previous output, the derivative of the cost function with respect to the weights must be performed through time. This can be accomplished by applying the chain rule to the cost function $V(\underline{a}_i(\underline{w}), \underline{w})$ with respect to $\underline{a}_i$ and $\underline{w}$. This application of the chain rule can be approached in two ways which leads to,

$$\frac{\partial V}{\partial \underline{w}} = \sum_i \frac{\partial \underline{a}_i}{\partial \underline{w}}^T \frac{\partial^e V}{\partial \underline{a}_i} \tag{1}$$

and,

$$\frac{\partial V}{\partial \underline{w}} = \sum_i \frac{\partial^e \underline{a}_i}{\partial \underline{w}}^T \frac{\partial V}{\partial \underline{a}_i} \tag{2}$$

If the time dependence of the cost function derivative is accounted for in calculating the derivative of the outputs with respect to the networks weights we use Eq. (1). If the time dependence is accounted for in computing the derivative of the cost function with respect to the network outputs we use Eq. (2). The derivatives denoted with the superscript e are explicit derivatives and can be computed using standard backpropagation. In Eq. (1) the implicit derivative of the network output with respect to the network weights can computed as,

$$\frac{\partial \underline{a}_i}{\partial \underline{w}} = \frac{\partial^e \underline{a}_i}{\partial \underline{w}} + \frac{\partial^e \underline{a}_i}{\partial \underline{a}_{i-1}} \frac{\partial \underline{a}_{i-1}}{\partial \underline{w}} \tag{3}$$

Eq. (1) and Eq. (3) are used together to comprise the forward perturbation method [1] of computing dynamic derivatives. In Eq. (2) the implicit derivative of the cost function with respect to the network output can computed using,

$$\frac{\partial V}{\partial \underline{a}_i} = \frac{\partial^e V}{\partial \underline{a}_i} + \frac{\partial^e \underline{a}_{i+1}}{\partial \underline{a}_i}^T \frac{\partial V}{\partial \underline{a}_{i+1}} \tag{4}$$

Eq. (2) and Eq. (4) are used together to comprise the backpropagation through time [1] method of computing dynamic derivatives. For the remainder of this paper we will only consider the forward perturbation method for computing dynamic derivatives.

2.2 Simple Recurrent Network Example

The simple recurrent network shown in Figure 2 will now be used to illustrate the application of Eq. (3) in computing the dynamic derivatives of the network outputs with respect to the network weights.

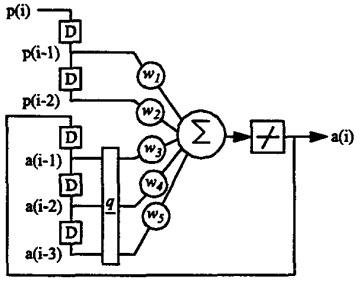

Figure 2 Simple Recurrent Network

This example will help demonstrate the effect of tapped-delay inputs in the formulation of the dynamic derivative equations. Eq. (3) can be rewritten as:

$$\frac{\partial \underline{a}_i}{\partial \underline{w}} = \frac{\partial^e \underline{a}_i}{\partial \underline{w}} + \frac{\partial \underline{q}}{\partial \underline{w}}^T \frac{\partial \underline{a}_i}{\partial \underline{q}} \tag{5}$$

Expanding Eq. (5) we have:

$$\frac{\partial \underline{a}_i}{\partial \underline{w}} = \begin{bmatrix} \frac{\partial^e \underline{a}_i}{\partial w_1} \\ \frac{\partial^e \underline{a}_i}{\partial w_2} \\ \vdots \\ \frac{\partial^e \underline{a}_i}{\partial w_5} \end{bmatrix} + \begin{bmatrix} \frac{\partial a_{i-1}}{\partial w_1} & \frac{\partial a_{i-2}}{\partial w_1} & \frac{\partial a_{i-3}}{\partial w_1} \\ \frac{\partial a_{i-1}}{\partial w_2} & \frac{\partial a_{i-2}}{\partial w_2} & \frac{\partial a_{i-3}}{\partial w_2} \\ \vdots & \vdots & \vdots \\ \frac{\partial a_{i-1}}{\partial w_5} & \frac{\partial a_{i-2}}{\partial w_5} & \frac{\partial a_{i-3}}{\partial w_5} \end{bmatrix} \begin{bmatrix} \frac{\partial^e \underline{a}_i}{\partial a_{i-1}} \\ \frac{\partial^e \underline{a}_i}{\partial a_{i-2}} \\ \frac{\partial^e \underline{a}_i}{\partial a_{i-3}} \end{bmatrix} \tag{6}$$

The first term on the right-hand side of Eq. (6) is the explicit derivative of the network outputs with respect to the weights. This term can be computed using standard backpropagation. The second term in Eq. (6) is the product of the derivative of the inputs q with respect to the network weights and the explicit derivative of the current output with respect to the past outputs which are used as network inputs. Notice that all elements in the derivative matrix of

Eq. (6) are stored values from previous recursions of the dynamic derivative equation. The number of taps in the feedback loop directly affects the size of the derivative matrix. The size of this matrix in turn affects the storage and computational requirements. The explicit derivatives of the current output with respect to the past outputs can be computed using a slight modification of the standard backpropagation method. Eq. (6) can be used to recursively compute the implicit derivatives in Eq. (1).

2.3 Multiple Loops and Tapped-delay Inputs

In network structures containing multiple loops Eq. (1) and Eq. (3) must be applied carefully to construct the equations needed for computing the gradients used in training. As multiple loops are added to a network structure containing two or more subnets, the dynamic derivative equations often become nested or coupled in some other way. Each new loop typically produces an additional recursive dynamic derivative equation, or at least adds a derivative product contribution to an existing recursive equation. These additional equations or product terms can add significantly to the computational burden associated with calculating network gradients. Each feedback loop contains at least one delay due to the discrete nature of neural networks. Delays anywhere in a structure of connected subnets necessitates the storage of delayed quantities, thereby increasing the storage requirements associated with training. Tapped delay lines placed within feedback loops or between subnets can substantially increase both the computational burdens and storage requirements related to network training. This is evident in Eq. (6) from the simple example of section 2.2. To further illustrate the effect of tapped-delay inputs, and to show the impact of multiple loops on derivative calculations let us now consider some additional examples.

2.4 Cascaded Recurrent Network Example

A cascaded recurrent neural network structure is shown in Figure 3. In Figure 3 each of the subnets NN_1, NN_2 and NN_3 contain weights $\underline{w}_1$, $\underline{w}_2$ and $\underline{w}_3$ respectively. For this example we will develop only the dynamic gradient equations needed for updating the weights in NN_1. The number of weights will be denoted N_1. Since we are using the forward perturbation method we will start our development by applying Eq. (3) to the NN_3 subnet. Doing this yields:

$$\underbrace{\frac{\partial \underline{a}_i}{\partial \underline{w}_1}}_{N_1 x 1} = \underbrace{\frac{\partial^e \underline{a}_i}{\partial \underline{w}_1}}_{N_1 x 1} + \underbrace{\frac{\partial \underline{q}_3^T}{\partial \underline{w}_1}}_{N_1 x 2} \underbrace{\frac{\partial^e \underline{a}_i}{\partial \underline{q}_3}}_{2 x 1} + \underbrace{\frac{\partial \underline{z}_3^T}{\partial \underline{w}_1}}_{N_1 x 2} \underbrace{\frac{\partial^e \underline{a}_i}{\partial \underline{z}_3}}_{2 x 1} \tag{7}$$

The explicit derivatives in Eq. (7) may be computed using standard backpropagation. In this example the explicit derivative of the output $\underline{a}_i$ with respect to the NN_1 weights is zero because there is no direct backpropagation path from the output to the weights, only tapped delay lines. The derivative of the inputs $\underline{q}_3$ with respect to the weights $\underline{w}_1$ is an N_1x2 matrix of past derivative values computed using Eq. (7). This matrix is updated after each recursion of Eq. (7). The derivative of the inputs $\underline{z}_3$ with respect to the weights $\underline{w}_1$ is by coincidence also an N_1x2 matrix.

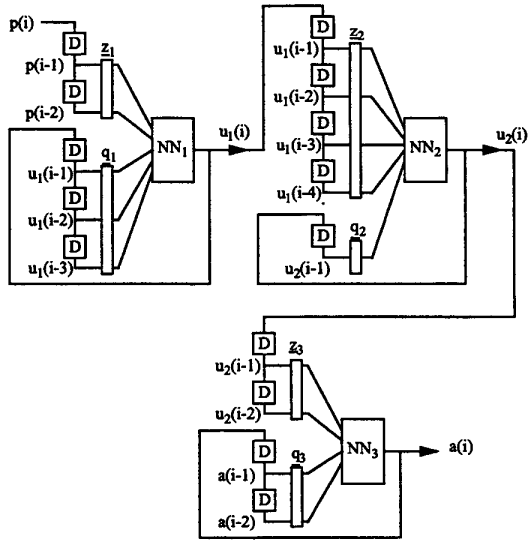

Figure 3 Cascaded Recurrent Neural Network Structure

However, the updates for this matrix are computed using another dynamic derivative equation which is:

$$\underbrace{\frac{\partial \underline{u}_{2_i}}{\partial \underline{w}_1}}_{N_1 x 1} = \underbrace{\frac{\partial^e \underline{u}_{2_i}}{\partial \underline{w}_1}}_{N_1 x 1} + \underbrace{\frac{\partial \underline{q}_2^T}{\partial \underline{w}_1}}_{N_1 x 1} \underbrace{\frac{\partial^e \underline{u}_{2_i}}{\partial \underline{q}_2}}_{1 x 1} + \underbrace{\frac{\partial \underline{z}_2^T}{\partial \underline{w}_1}}_{N_1 x 4} \underbrace{\frac{\partial^e \underline{u}_{2_i}}{\partial \underline{z}_2}}_{4 x 1} \tag{8}$$

This derivative must be computed dynamically because of the presence of the feedback loop around subnet NN_2. Eq. (8) is used to update some of the elements of Eq. (7) at each recursion. The explicit derivatives in Eq. (8) may be computed using standard backpropagation. The derivative of the inputs $\underline{q}_2$ with respect to the weights $\underline{w}_1$ is an N_1x1 matrix of past derivative values computed using Eq. (8). This matrix is updated after each recursion of Eq. (8). The derivative of the inputs $\underline{z}_2$ with respect to the weights $\underline{w}_1$ is an N_1x4 matrix. The updates for this matrix are computed using yet another dynamic derivative equation which is:

$$\underbrace{\frac{\partial \underline{u}_{1_i}}{\partial \underline{w}_1}}_{N_1 x 1} = \underbrace{\frac{\partial^e \underline{u}_{1_i}}{\partial \underline{w}_1}}_{N_1 x 1} + \underbrace{\frac{\partial \underline{q}_1^T}{\partial \underline{w}_1}}_{N_1 x 3} \underbrace{\frac{\partial^e \underline{u}_{1_i}}{\partial \underline{q}_1}}_{3 x 1} \tag{9}$$

This derivative must be computed dynamically because of the presence of the feedback loop around subnet NN_1. Eq. (9) is used to update some of the elements of Eq. (8) at each recursion. Notice that Eq. (9) has one less term than Eq. (7) and Eq. (8). This is because the derivative of the inputs $\underline{z}_1$ with respect to the NN_1 weights $\underline{w}_1$ are zeros, so this term is always zero in terms of the nonrecurrent inputs. The explicit derivatives in Eq. (9) may be computed using standard backpropagation. The derivative of the inputs $\underline{q}_1$ with

respect to the weights $\underline{w}_1$ is an N_1x3 matrix of past derivative values computed using Eq. (9). This matrix is updated after each recursion of Eq. (9). Similar equations for the weights contained in subnets NN_2 and NN_3 can be developed. This example shows us that for each cascaded recurrent network we add to the combined subnet structure we need one more nested dynamic derivative equation. Again it is important to note that the size of the non-explicit derivative matrices appearing in the dynamic backpropagation equations is directly related to the number of tapped-delay inputs used for each subnet.

2.5 Nested Loop Recurrent Network Example

A nested loop recurrent neural network structure is shown in Figure 4.

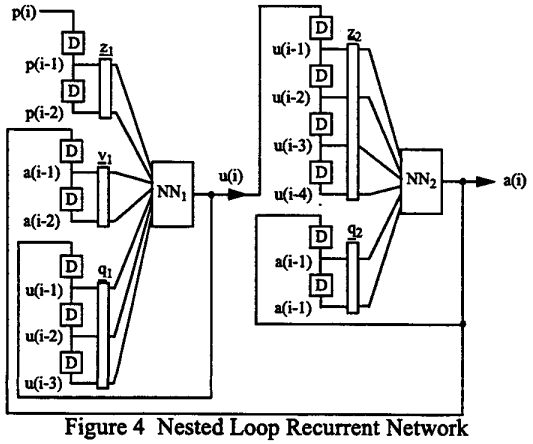

Figure 4 Nested Loop Recurrent Network

In Figure 4 the subnets NN_1, NN_2 contain weights $\underline{w}_1$ and $\underline{w}_2$, respectively. For this example we will develop only the dynamic gradient equations needed for updating the weights in NN_1. The number of weights will be denoted N_1. Since we are using the forward perturbation method we will start our development by applying Eq. (3) to the NN_2 subnet. Doing this yields:

$$\underbrace{\frac{\partial \underline{a}_i}{\partial \underline{w}_1}}_{N_1x1} = \underbrace{\frac{\partial^e \underline{a}_i}{\partial \underline{w}_1}}_{N_1x1} + \underbrace{\frac{\partial \underline{q}_2^T}{\partial \underline{w}_1}}_{N_1x2} \underbrace{\frac{\partial^e \underline{a}_i}{\partial \underline{q}_2}}_{2x1} + \underbrace{\frac{\partial \underline{z}_2^T}{\partial \underline{w}_1}}_{N_1x4} \underbrace{\frac{\partial^e \underline{a}_i}{\partial \underline{z}_2}}_{4x1} \tag{10}$$

The explicit derivatives in Eq. (10) may be computed using standard backpropagation. In this example the explicit derivative of the output $\underline{a}_i$ with respect to the NN_1 weights is zero because there is no direct backpropagation path from the output to the weights, only tapped delay lines. The derivative of the inputs $\underline{q}_2$ with respect to the weights $\underline{w}_1$ is an N_1x2 matrix of past derivative values computed using Eq. (10). This matrix is updated after each recursion of Eq. (10). The derivative of the inputs $\underline{z}_2$ with respect to the weights $\underline{w}_1$ is an N_1x4 matrix. However, the updates for this matrix are computed using another dynamic derivative equation which is:

$$\underbrace{\frac{\partial \underline{u}_i}{\partial \underline{w}_1}}_{N_1x1} = \underbrace{\frac{\partial^e \underline{u}_i}{\partial \underline{w}_1}}_{N_1x1} + \underbrace{\frac{\partial \underline{q}_1^T}{\partial \underline{w}_1}}_{N_1x3} \underbrace{\frac{\partial^e \underline{u}_i}{\partial \underline{q}_1}}_{3x1} + \underbrace{\frac{\partial \underline{v}_1^T}{\partial \underline{w}_1}}_{N_1x2} \underbrace{\frac{\partial^e \underline{u}_i}{\partial \underline{v}_1}}_{2x1} \tag{11}$$

This derivative includes dynamic terms from both the inner loop around NN_1, and the outer loop which goes around subnets NN_1, and NN_2. Notice that in this example the presence of the outer feedback loop did not create the need for another dynamic derivative equation as in the cascaded recurrent example, but added a term to the last dynamic derivative equation. Eq. (11) is used to update some of the elements of Eq. (10) at each recursion. The explicit derivatives in Eq. (11) may be computed using standard backpropagation. The derivative of the inputs $\underline{q}_1$ with respect to the weights $\underline{w}_1$ is an N_1x3 matrix of past derivative values computed using Eq. (11). This matrix is updated after each recursion of Eq. (11). The derivative of the inputs $\underline{v}_1$ with respect to the weights $\underline{w}_1$ is an N_1x2 matrix. The updates for this matrix are computed using Eq. (10). Similar equations for the weights contained in subnet NN_2 can be developed. In this example the dynamic training equation are not nested, but instead they are coupled. Each is used to compute matrix updates for the other. This coupling occurs in network structures which contain nested loops. In examining this example and the cascaded recurrent network example in the previous section we can draw some conclusions. We can see that cascaded recurrent networks produce nested dynamic derivative equations, and that nested feedback loops produce coupled dynamic derivative equations.

3. Computational Considerations

The examples in the previous section illustrate how feedback loops, and even more importantly, how tapped-delay input structures increase the computational burden and storage requirements in training neural networks. In this section we will quantify the costs associated with dynamic training equations and tapped-delay lines. Consider the simple subnetwork structure of Figure 5.

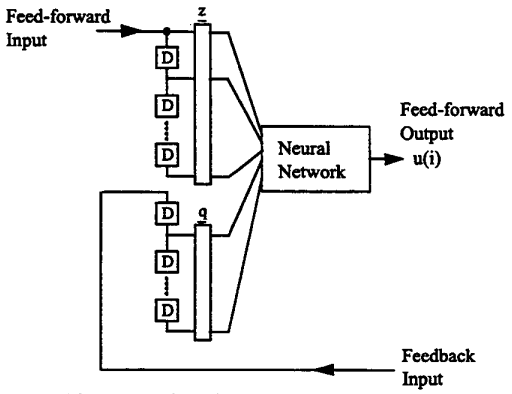

Figure 5 Time-lagged Recurrent Subnet

Z and Q will be defined as the number of taps in the feed-forward and feedback input lines, respectively, for an individual subnet. We will assume that the output of the subnet is eventually connected to the feedback in some manner such that the feedback is affected by the subnet output. We will also assume that the feed-forward input to the subnet is coming from at least one other subnet in which there are weights which must be trained. The number of weights which are to be adjusted in a preceding subnet will be defined as N. The generic dynamic gradient equation, which must be used in backpropagating derivatives through the single output structure shown in Figure 5, can be written as:

$$\underbrace{\frac{\partial \underline{u}_i}{\partial \underline{w}}}_{Nx1} = \underbrace{\frac{\partial^e \underline{u}_i}{\partial \underline{w}}}_{Nx1} + \underbrace{\frac{\partial \underline{q}^T}{\partial \underline{w}}}_{NxQ} \underbrace{\frac{\partial^e \underline{u}_i}{\partial \underline{q}}}_{Qx1} + \underbrace{\frac{\partial \underline{z}^T}{\partial \underline{w}}}_{NxZ} \underbrace{\frac{\partial^e \underline{u}_i}{\partial \underline{z}}}_{Zx1} \tag{12}$$

If more inputs either from additional feedback loops, or additional feed-forward paths are connected to a subnet, then Eq. (12) will have additional matrix-vector product terms like the ones associated with each of the two inputs in Figure 5. Now let's examine the nature of the basic matrix-vector product terms which appear in Eq. (12). One of these matrix-vector product terms is needed for each tapped-delay line. The number of multiplications and additions needed to compute one of these terms can be expressed as:

$$Multiplications = NumTaps \times NumWeights \tag{13}$$

$$Additions = (NumTaps - 1) \times NumWeights \tag{14}$$

Eq. (13) means that for every additional tap in any delay line which is used as input to a subnet through which derivatives are backpropagated, there will be additional required multiplications equal to the number of weights to be adjusted. Eq. (14) indicates there be increased summing operations required which equal the number of weights to be adjusted. If many taps are used, the increased burden of computing the dynamic derivatives can be very substantial. Returning now to Eq. (12), we can see that a matrix of derivatives must be stored from one time step to the next. Often the situation arises where elements of the same derivative matrix are used in computing dynamic derivatives for two different subnets. This means that only one derivative matrix must be stored from one time step to the next. This can occur when a particular subnet output is used as both feed-forward and feedback inputs. This situation can also occur if a subnet output is a feed-forward input to two different subnets, or as feedback input to two different subnets. In any of these cases the storage requirement will be dependent on the longest tapped-delay line in which the input is used. The number of different derivative matrices which must be saved is equal to the number of unique outputs within the structure of connected subnets which are passed through tapped-delay lines. The number of elements which must be saved can be expressed as:

$$MemoryLocs = NumWeights \times MaxDelays \tag{15}$$

For real-time applications the computational burden associated with tapped-delay lines is of a bigger concern than the issue of storage. The preceding discussion highlights the impact of computing full dynamic derivatives for recurrent networks. The increase in computational burden associated with computing full dynamic derivatives makes it tempting to use gradient approximations instead of true dynamic derivatives in training neural networks. One such approximation is to ignore the dynamic terms completely, using only explicit derivatives in training. If this technique is used where tapped-delay lines are present between subnets, then it may be necessary to approximate the explicit derivative from one subnet to the next using the most current path in the tapped-delay line. Another approach to approximating dynamic derivative terms is to use only the first dynamic derivative term contribution in the derivative matrix-vector product. In other words only the contribution from the first delay tap is considered in the dynamic derivative formulation. In the next section we will examine the result of using the various approximations.

4. Simulation Results

In this section the results for a simulated neural network controls problem will be presented. The controller performance when using true dynamic derivatives for the neuro-controller training will be compared to performance when using two derivative approximations. The evaluation will be based on squared error performance, and required floating point operations. Figure 6 is a schematic of the

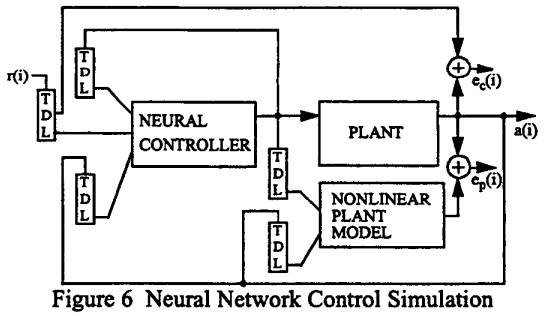

Figure 6 Neural Network Control Simulation

simulated system. This system is very similar to the model reference adaptive control (MRAC) problem described in [4]. The blocks marked TDL represent tapped-delay lines. The plant model is used only as a backpropagation path for the derivatives needed to adjust the controller weights. The plant model used was a fully recurrent 2-layer nonlinear multi-layer perceptron network with 10 input taps, 50 feedback taps and 15 hidden neurons. The weights in the plant model are not adjusted during controller training. The controller weights are adjusted such that the error $e_c(i)$, between a delayed reference input r(i), and the actual plant output a(i), is minimized. The controller structure consisted of 40 input taps, 50 controller feedback taps, 75 plant output feedback taps and 15 hidden neurons. The reference input used in the simulation consisted of a ran-

dom sequence of tone burst pulses as shown in Figure 7. The tone bursts are evenly spaced and appear randomly at one of two frequencies. Periodic disturbance noise was added to the plant input. The open-loop plant response with no controller is shown in Figure 8. The simulation was first run using full dynamic backpropagation during controller training. The plant response after controller convergence is shown in Figure 9. In the next part of the simulation dynamic backpropagation was used in the plant model, but was not used in backpropagating derivatives in the controller. The results are shown in Figure 10.

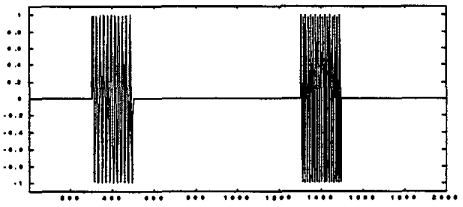

Figure 7 Tone Burst Controller Input

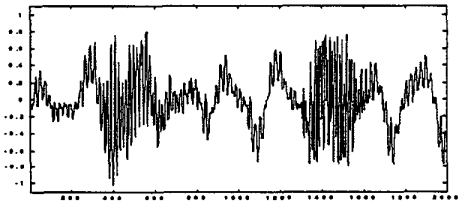

Figure 8 Open-loop Plant Response

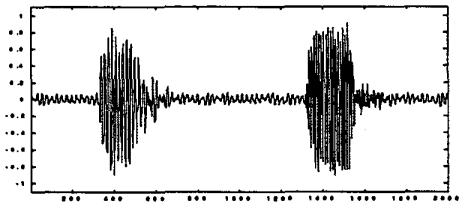

Figure 9 Response with Full Dynamic Training

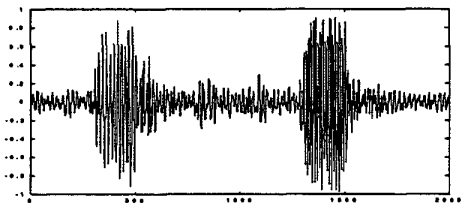

Figure 10 Response without Dynamic Controller Training

For the last part of the simulation, dynamic backpropagation was only used to compute a dynamic derivative across the first delay in the tapped-delay line between the plant model and the controller. All other derivatives were computed using only explicit derivatives. It was impossible to get the controller weights to converge, so a plot of the results is not shown. Table 1. provides a summary of the performance results for the simulations. These results highlight how much more computational burden there is when calculating dynamic derivatives than when using static backpropagation alone. In this example reasonable performance was possible even when dynamic derivatives were used only in the plant model. This derivative approximation decreased the computational burden by approximately 65%. Using essentially no dynamic derivatives in training reduced the computational burden by approximately 98%. However, performance in this case was unacceptable.

Derivative Method	Flops/Sample	Sum Squared Error
Full Dynamic	9.83×10^5	43.44
Plant Only Dynamic	3.48×10^5	55.53
No Dynamic	1.85×10^4	127.88

Table 1. Simulation Results

5. Conclusions

In this paper we described time-lagged recurrent networks and dynamic backpropagation. More specifically, we examined the impact of the presence of multiple feedback loops and tapped-delay lines on the equations used to compute the dynamic derivatives needed for network training. Several examples were presented and the derivative equations developed for each. The effects of feedback loops and tapped-delay lines on computational burden and storage requirements were discussed. Simulation results for a nonlinear adaptive controls problem were presented. The effects on computational burden and squared-error performance when using two derivative approximations have been compared to results obtained when using full dynamic backpropagation.

6. References

[1] P.J. Werbos, "Backpropagation Through Time": What It Is and How To Do It", Proceedings of the IEEE, Vol 78, 1990, pp. 1550-1560

[2] Wei-Chung Yang, "Neurocontrol Using Dynamic Learning", *Doctoral Thesis*, Oklahoma State University, 1994

[3] M. T. Hagan, H. B. Demuth and M. Beale, *Neural Network Design*, Boston: PWS Publishing Co., 1996.

[4] K.S. Narendra, A.M. Parthasrathy, "Identification and Control for Dynamic Systems Using Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 1, 1990, pp. 4-27