Chapter 12

TRAINING RECURRENT NETWORKS FOR FILTERING AND CONTROL

Martin T. Hagan, Orlando De Jesús, Roger Schultz

School of Electrical and Computer Engineering
Oklahoma State University, Stillwater, Oklahoma

I. INTRODUCTION

Neural networks can be classified into recurrent and nonrecurrent categories. Nonrecurrent (feedforward) networks have no feedback elements; the output is calculated directly from the input through feedforward connections. In recurrent networks the output depends not only on the current input to the network, but also on the current or previous outputs or states of the network. For this reason, recurrent networks are more powerful than nonrecurrent networks and have important uses in control and signal processing applications.

This chapter introduces the Layered Digital Recurrent Network (LDRN), develops a general training algorithm for this network, and demonstrates the application of the LDRN to problems in controls and signal processing. In Section II we present the notation necessary to represent the LDRN. Section III contains a discussion of the dynamic backpropagation algorithms that are required to compute training gradients for recurrent networks. The concepts underlying the backpropagation-through-time and forward perturbation algorithms are presented in a unified framework and are demonstrated for a simple, single-loop recurrent network. In Section IV we describe a general forward perturbation algorithm for computing training gradients for the LDRN. Two application sections follow the discussion of dynamic backpropagation: neurocontrol and nonlinear filtering. These sections demonstrate the implementation of the general dynamic backpropagation algorithm. The control section (Section V) applies a neurocontrol architecture to the automatic equalization of an acoustic transmitter. The nonlinear filtering section (Section VI) demonstrates the application of a recurrent filtering network to a noise-cancellation application.

II. PRELIMINARIES

In this section we want to introduce the types of neural networks that are discussed in the remainder of this chapter. We also present the notation that we use to represent these networks. The networks we use are Layered Digital Recurrent Networks (LDRN). They are a generalization of the Layered Feedforward Network (LFFN), which has been modified to include feedback connections and delays. We begin here with a description of the LFFN and then show how it can be generalized to obtain the LDRN.

A. LAYERED FEEDFORWARD NETWORK

Figure 1 is an example of a layered feedforward network (two layers in this case). (See Demuth *et al.* [1998] for a full description of the notation used here.) The input vector to the network is represented by $\mathbf{p}^1$, which has R^1 elements. The superscript represents the input number, since it is possible to have more than one input vector. The input is connected to Layer 1 through the input weight $\mathbf{IW}^{1,1}$, where the first superscript represents the layer number and the second superscript represents the input number. The bias for the first layer is represented by $\mathbf{b}^1$. The net input to Layer 1 is denoted by $\mathbf{n}^1$, and is computed as

$$\mathbf{n}^1 = \mathbf{IW}^{1,1}\mathbf{p}^1 + \mathbf{b}^1 \tag{1}$$

The output of Layer 1, $\mathbf{a}^1$, is computed by passing the net input through a transfer function, according to $\mathbf{a}^1 = \mathbf{f}^1(\mathbf{n}^1)$. The output has S^1 elements. The output of the first layer is input to the second layer through the layer weight $\mathbf{LW}^{2,1}$. The overall output of the network is labeled $\mathbf{y}$. This is typically chosen to be the output of the last layer in the network, as it is in Figure 1, although it could be the output of any layer in the network.

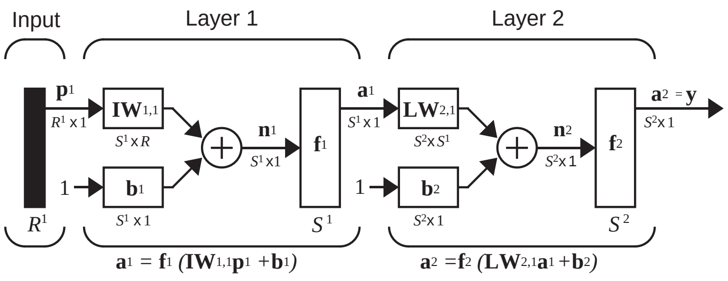


Figure 1. Example of a Layered Feedforward Network

Each layer in the LFFN is made up of 1) a set of weight matrices that come into that layer (which may connect from other layers or from external inputs), 2) a bias vector, 3) a summing junction, and 4) a transfer function. (In the LDRN, a set of tapped delay lines may also be included in a layer, as we will see later.) In the example given in Figure 1, there is only one weight matrix associated with each layer, but it is possible to have weight matrices that are connected from several different input vectors and layer outputs. This will become clear when we introduce the LDRN network. Also, the example in Figure 1 has only two layers; our general LFFN can have an arbitrary number of layers. The layers do not have to be connected in sequence from Layer 1 to Layer M. For example, Layer 1 could be connected to both Layer 3 and Layer 4, by weights $\mathbf{LW}^{3,1}$ and $\mathbf{LW}^{4,1}$, respectively. Although the layers do not have to be connected in a linear sequence by layer number, it must be possible to compute the output of the network by a simple sequence of calculations. There cannot be any feedback loops in the network. The order in which the individual layer outputs must be computed in order to obtain the correct network output is called the *simulation order*.

B. LAYERED DIGITAL RECURRENT NETWORK

We now introduce a class of recurrent networks that are based on the LFFN. The LFFN is a static network, in the sense that the network output can be computed directly from the network input, without the knowledge of initial network states. A Layered Digital Recurrent Network (LDRN) can contain feedback loops and time delays. The network response is a function of network inputs, as well as initial network states.

The components of the LDRN are the same as those of the LFFN, with the addition of the tapped delay line (TDL), which is shown in Figure 2. The output of the TDL is a vector containing current and previous values of the TDL input. In Figure 2 we show two abbreviated representations for the TDL. In the case on the left, the undelayed value of the input variable is included in the output vector. In the case on the right, only delayed values of the input are included in the output.

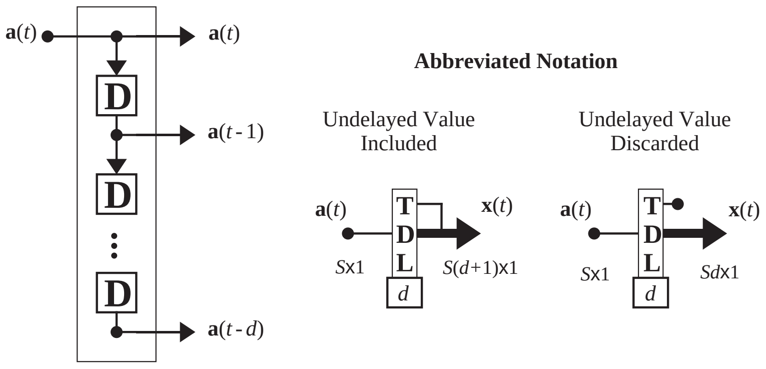


Figure 2. Tapped Delay Line

Figure 3 is an example of an LDRN. Like the LFFN, the LDRN is made up of layers. In addition to the weight matrices, bias, summing junction, and transfer function, which make up the layers of the LFFN, the layers of the LDRN also include any tapped delay lines that appear at the input of a weight matrix. (Any weight matrix in an LDRN can be proceeded by a tapped delay line.) For example, Layer 1 of Figure 3 contains the weight $\mathbf{LW}^{1,2}$ and the TDL at its input. Note that all of the layer outputs and net inputs in the LDRN are explicit functions of time.

The output of the TDL in Figure 3 is labeled $\grave{\mathbf{a}}^{2,2}(t)$. This indicates that it is a composite vector made up of delayed values of the output of Subnet 2 (indicated by the second superscript) and is an input to Subnet 2 (indicated by the first superscript). (A subnet is a series of layers which have no internal tapped delay lines. The number of the subnet is the same as the number of its output layer. These concepts are defined more carefully in a later section.) These TDL outputs are important variables in our training algorithm for the LDRN.

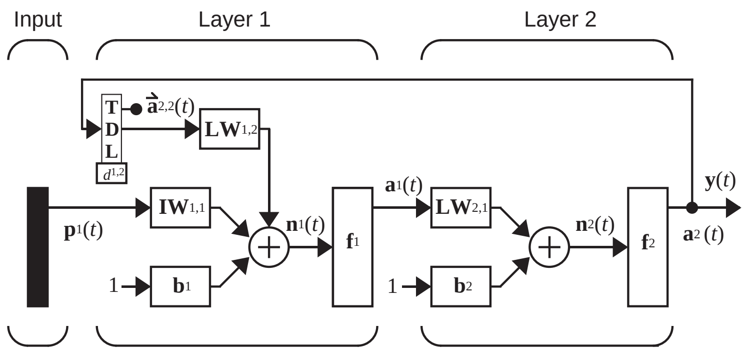


Figure 3. Layered Digital Recurrent Network Example

In the LDRN, feedback is added to an LFFN. Therefore, unlike the LFFN, the output of the network is a function not only of the weights, biases, and network input, but also of the outputs of some of the network layers at previous points in time. For this reason, it is not a simple matter to calculate the gradient of the network output with respect to the weights and biases (which is needed to train the network). This is because the weights and biases have two different effects on the network output. The first is the direct effect, which can be calculated using the standard backpropagation algorithm [Hagan *et al.*, 1996]. The second is an indirect effect, since some of the inputs to the network, such as $\vec{\mathbf{a}}^{1,2}(t)$, are also functions of the weights and biases. In the next section we briefly describe the gradient calculations for the LFFN and show how they must be modified for the LDRN. The main development of the next two sections is a general gradient calculation for arbitrary LDRN's.

III. PRINCIPLES OF DYNAMIC LEARNING

Consider again the multilayer network of Figure 1. The basic simulation equation of such a network is

$$\mathbf{a}^k = \mathbf{f}^k\left(\sum_i \mathbf{IW}^{k,i}\mathbf{p}^i + \sum_j \mathbf{LW}^{k,j}\mathbf{a}^j + \mathbf{b}^k\right), \tag{2}$$

where k is incremented through the simulation order.

The task of the network is to learn associations between a specified set of input/output pairs: $\{(\mathbf{p}_1, \mathbf{t}_1), (\mathbf{p}_2, \mathbf{t}_2), \ldots, (\mathbf{p}_Q, \mathbf{t}_Q)\}$. The performance index for the network is

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{y}_q)^T (\mathbf{t}_q - \mathbf{y}_q) = \sum_{q=1}^{Q} \mathbf{e}_q^T \mathbf{e}_q \tag{3}$$

where $\mathbf{y}_q$ is the output of the network when the q^{th} input, $\mathbf{p}_q$, is presented, and $\mathbf{x}$ is a vector containing all of the weights and biases in the network. (Later we use $\mathbf{x}^i$ to represent the weights and biases in Layer *i*.) The network should learn the $\mathbf{x}$ vector that minimizes F.

For the standard backpropagation algorithm [Hagan *et al.*, 1996] we use a steepest descent learning rule. The performance index is approximated by:

$$\hat{F} = \mathbf{e}_q^T \mathbf{e}_q, \tag{4}$$

where the total sum of squares is replaced by the squared errors for a single input/output pair. The approximate steepest (gradient) descent algorithm is then:

$$\Delta w_{i,j}^{k,l} = -\alpha \frac{\partial \hat{F}}{\partial w_{i,j}^{k,l}}, \quad \Delta b_i^k = -\alpha \frac{\partial \hat{F}}{\partial b_i^k} \tag{5}$$

where α is the learning rate. Define

$$s_i^k \equiv \frac{\partial \hat{F}}{\partial n_i^k} \tag{6}$$

as the sensitivity of the performance index to changes in the net input of unit *i* in layer *k*. Using the chain rule, we can show that

$$\frac{\partial \hat{F}}{\partial iw_{i,j}^{m,l}} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial iw_{i,j}^{m,l}} = s_i^m p_j^l, \quad \frac{\partial \hat{F}}{\partial lw_{i,j}^{m,l}} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial lw_{i,j}^{m,l}} = s_i^m a_j^l,$$

$$\frac{\partial \hat{F}}{\partial b_i^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial b_i^m} = s_i^m \tag{7}$$

It can also be shown that the sensitivities satisfy the following recurrence relation, in which *m* is incremented through the *backpropagation order*, which is the reverse of the simulation order:

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m) \sum_i (\mathbf{LW}^{i,m})^T \mathbf{s}^i \tag{8}$$

where

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \dots & 0 \\ 0 & \dot{f}^m(n_2^m) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \dot{f}^m(n_{S^m}^m) \end{bmatrix} \tag{9}$$

and

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m} \tag{10}$$

This recurrence relation is initialized at the output layer:

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}_q - \mathbf{y}_q) . \tag{11}$$

The overall learning algorithm now proceeds as follows: first, propagate the input forward using Eq. (2); next, propagate the sensitivities back using Eq. (11) and Eq. (8); and finally, update the weights and biases using Eq. (5) and Eq. (7).

Now consider an LDRN, such as the one shown in Figure 3. Suppose that we use the same gradient descent algorithm, Eq. (5), that is used in the standard backpropagation algorithm. The problem in this case is that when we try to find the equivalent of Eq. (7) we note that the weights and biases have two different effects on the network output. The first is the direct effect, which is accounted for by Eq. (7). The second is an indirect effect, since some of the inputs to the network, such as $\hat{\mathbf{a}}^{1,2}(t)$, are also functions of the weights and biases. To account for this indirect effect we must use dynamic backpropagation.

To illustrate dynamic backpropagation [Yang *et al.*, 1993, Yang, 1994], consider Figure 4, which is a simple recurrent network. It consists of an LFFN with a single feedback loop added from the output of the network, which is connected to the input of the network through a single delay. In this figure the vector $\mathbf{x}$ represents all of the network parameters (weights and biases) and the vector $\mathbf{a}(t)$ represents the output of the LFFN at time step *t*.

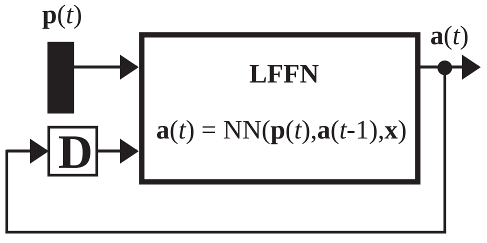


Figure 4. Simple Recurrent Network

Now suppose that we want to minimize

$$F(\mathbf{x}) = \sum_{t=1}^{Q} (\mathbf{t}(t) - \mathbf{a}(t))^T (\mathbf{t}(t) - \mathbf{a}(t)) \tag{12}$$

In order to use gradient descent, we need to find the gradient of *F* with respect to the network parameters. There are two different approaches to this problem. They both use the chain rule, but are implemented in different ways:

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \left[\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}}\right]^T \times \frac{\partial^e F}{\partial \mathbf{a}(t)} \tag{13}$$

or

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \left[\frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{x}}\right]^T \times \frac{\partial F}{\partial \mathbf{a}(t)} \tag{14}$$

where the superscript *e* indicates an explicit derivative, not accounting for indirect effects through time. The explicit derivatives can be obtained with the standard backpropagation algorithm, as in Eq. (8). To find the complete derivatives that are required in Eq. (13) and Eq. (14), we need the additional equations:

$$\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}} = \frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{a}(t-1)} \times \frac{\partial \mathbf{a}(t-1)}{\partial \mathbf{x}} \tag{15}$$

and

$$\frac{\partial F}{\partial \mathbf{a}(t)} = \frac{\partial^e F}{\partial \mathbf{a}(t)} + \frac{\partial^e \mathbf{a}(t+1)}{\partial \mathbf{a}(t)} \times \frac{\partial F}{\partial \mathbf{a}(t+1)} \tag{16}$$

Eq. (13) and Eq. (15) make up the forward perturbation (FP) algorithm. Note that the key term is

$$\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}} \tag{17}$$

which must be propagated forward through time.

Eq. (14) and Eq. (16) make up the backpropagation-through-time (BTT) algorithm. Here the key term is

$$\frac{\partial F}{\partial \mathbf{a}(t)} \tag{18}$$

which must be propagated backward through time.

In general, the FP algorithm requires somewhat more computation than the BTT algorithm. However, the BTT algorithm cannot be implemented in real time, since the outputs must be computed for all time steps, and then the derivatives must be backpropagated back to the initial time point. The FP algorithm is well suited for real time implementation, since the derivatives can be calculated at each time step.

IV. DYNAMIC BACKPROP FOR THE LDRN

In this section, we generalize the FP algorithm, so that it can be applied to arbitrary LDRN's. This is followed by applications of the LDRN and dynamic backpropagation to problems in filtering and control.

A. PRELIMINARIES

To explain this algorithm, we must create certain definitions related to the LDRN. We do that in the following paragraphs.

First, as we stated earlier, a *layer* consists of a set of *weights*, associated *tapped delay lines*, a *summing function*, and a *transfer function*. The network has *inputs* that are connected to special weights, called *input weights*, and denoted by $\mathbf{IW}^{i,j}$, where j denotes the number of the input vector that enters the weight, and i denotes the number of the layer to which the weight is connected. The weights connecting one layer to another are called *layer weights* and are denoted by $\mathbf{LW}^{i,j}$, where j denotes the number of the layer coming into the weight and i denotes the number of the layer at the output of weight. In order to calculate the network response in stages, layer by layer, we need to proceed in the proper layer order, so that the necessary inputs at each layer will be available. This ordering of layers is called the *simulation order*. In order to backpropagate the derivatives for the gradient calculations, we must proceed in the opposite order, which is called the *backpropagation order*.

In order to simplify the description of the training algorithm, the LDRN is divided into *subnets*. A subnet is a section of a network that has no tapped delay lines, except at the subnet input. Every LDRN can be organized as a collection of subnets. We define the subnets by proceeding backwards from the last subnet to the first subnet. To locate the last subnet, start at the first layer in the backpropagation order and proceed through the backpropagation order until you find a layer containing delays, which becomes the first layer in the last subnet. The last subnet is then defined as containing all of the layers beginning at the layer containing delays and continuing through the simulation order to the first layer in the backpropagation order (or the last layer in the simulation order). This defines the last subnet. To find the preceding subnet, start with the next layer in the backpropagation order and proceed in the same way until you find the next layer with delays. This process continues until you reach the last layer in the backpropagation order, at which time the first *subnet* is defined. As with the layer simulation order, we can also define a *subnet simulation order* that starts at the first subnet and continues until the last subnet.

For example, the LDRN shown in Figure 5 has thee layers and two subnets. To simplify the algorithm, the subnet is denoted by the number of its output layer. For this network the simulation order is 1-2-3, the backpropagation order is 3-2-1 and the subnet simulation order is 1-3.

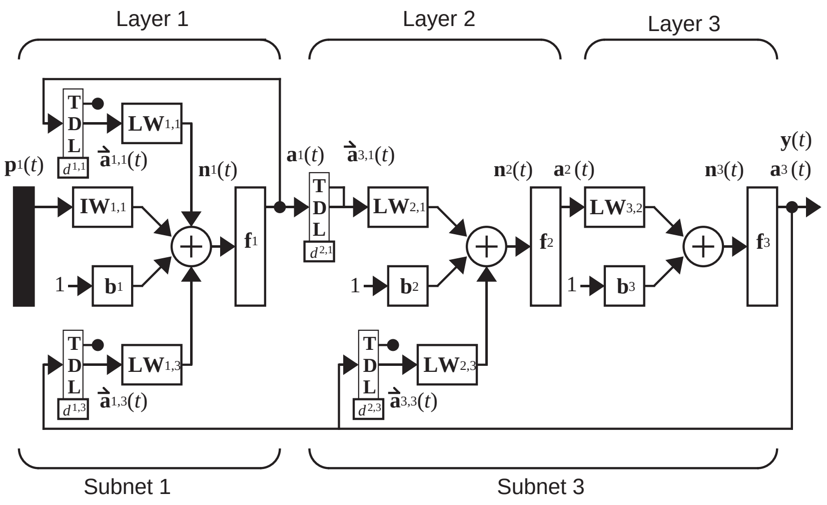


Figure 5. Three-layer LDRN with Two Subnets

B. EXPLICIT DERIVATIVES

We want to generalize the forward perturbation (FP) algorithm of Eq. (13) and Eq. (15) so that it can be applied to an arbitrary LDRN. Notice that we have two terms on the right-hand side of Eq. (15). We have an explicit derivative of the performance with respect to the weights, which accounts for the direct effect of the weights on the performance and can be computed through the standard backpropagation algorithm, as in Eq. (8). We also have a second term, which accounts for the fact that the weights have a secondary effect through the previous network output. In the general LDRN, we may have many different feedback loops, and therefore there could be many different terms on the right of Eq. (15), and each of those terms would have a separate equation like Eq. (15) to update the total derivative through time. For our development of the FP algorithm, we have one term (and an additional equation) for every place where one subnet is input to another subnet. Recall that the subnet boundaries are determined by the locations of the tapped delay lines. Within a subnet, standard backpropagation, as in Eq. (8), can be used to propagate the explicit derivatives, but at the subnet boundaries an equation like Eq. (15) must be used to calculate the total derivative, which includes both direct and indirect effects. In this subsection we describe the computation of the explicit derivatives, and then in the following subsection we explain the total derivative computation.

A backpropagation process to calculate the explicit derivatives is needed for each subnet. These equations involve calculating the derivative of the subnet output with respect to each layer output in the subnet. The basic equation is

$$\frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{a}^i(t)} = \sum_k \frac{\partial^e \mathbf{n}^k(t)}{\partial \mathbf{a}^i(t)} \times \frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{n}^k(t)} \tag{19}$$

where $\mathbf{a}^{jz}(t)$ represents a subnet output, $\mathbf{a}^i(t)$ is the output of a layer in subnet *jz*, $\mathbf{n}^k(t)$ is the net input of layer *k*, which has a connection from layer *i*. The index *i* is incremented through the backpropagation order. If we define

$$\mathbf{S}^{k,\,jz} \equiv \frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{n}^k(t)}, \text{ and note that } \frac{\partial^e \mathbf{n}^k(t)}{\partial \mathbf{a}^i(t)} = \mathbf{LW}^{k,\,i}, \tag{20}$$

then we can write Eq. (19) as

$$\frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{a}^i(t)} = \sum_k \mathbf{LW}^{k,\,i} \times \mathbf{S}^{k,\,jz}. \tag{21}$$

This recursion, where *i* is incremented along the backpropagation order, begins at the last layer in the subnet:

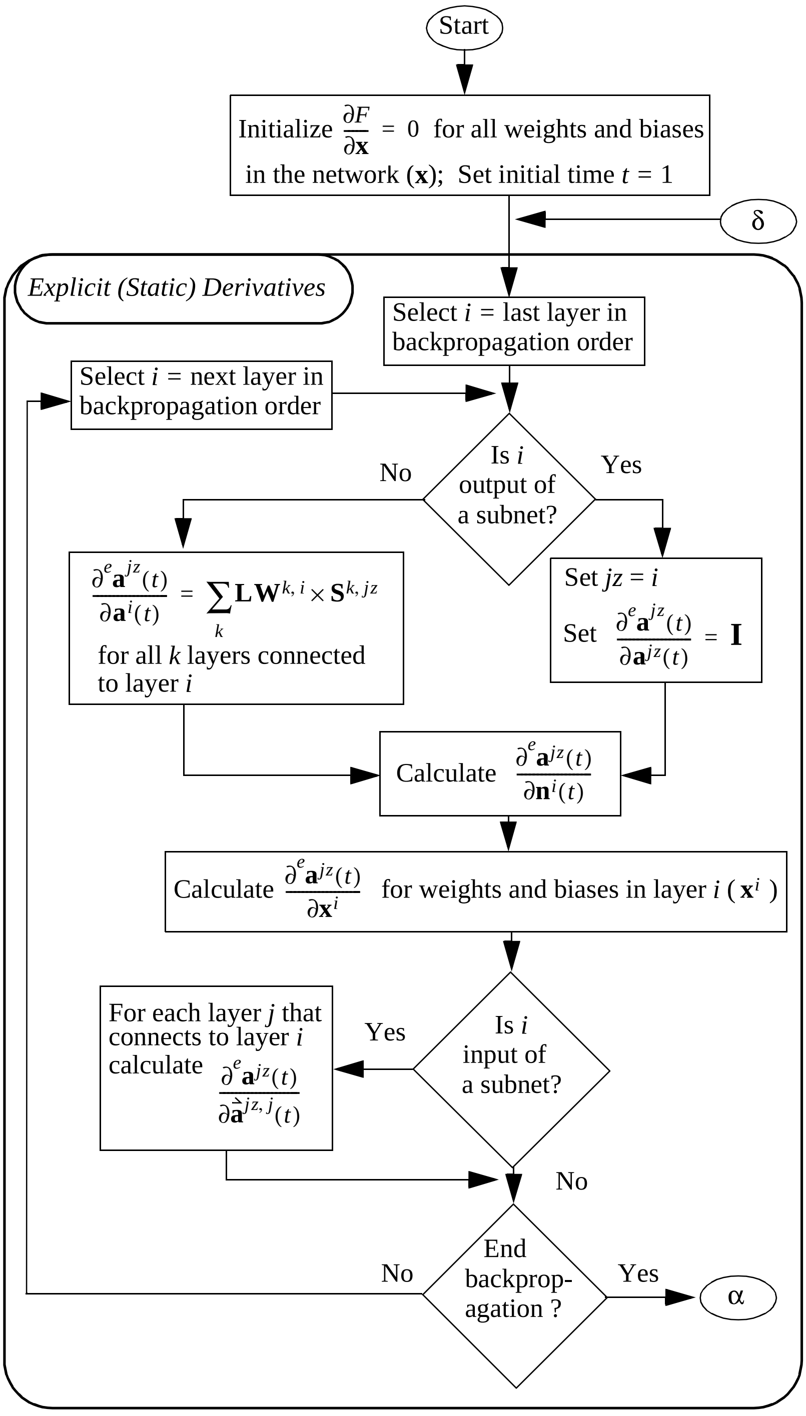
$$\frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{a}^{jz}(t)} = \mathbf{I}, \tag{22}$$

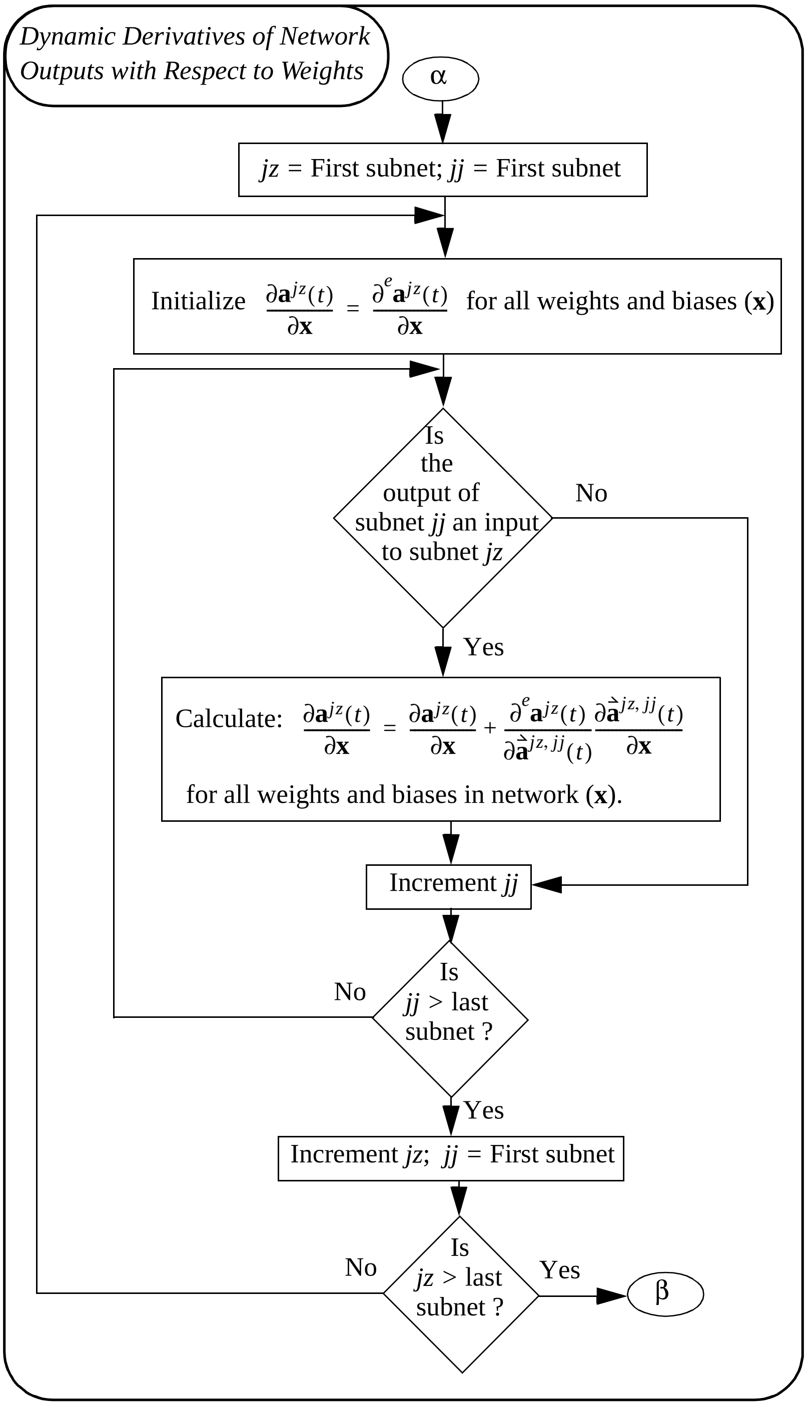
where $\mathbf{I}$ is an identity matrix whose dimension is the size of layer jz.

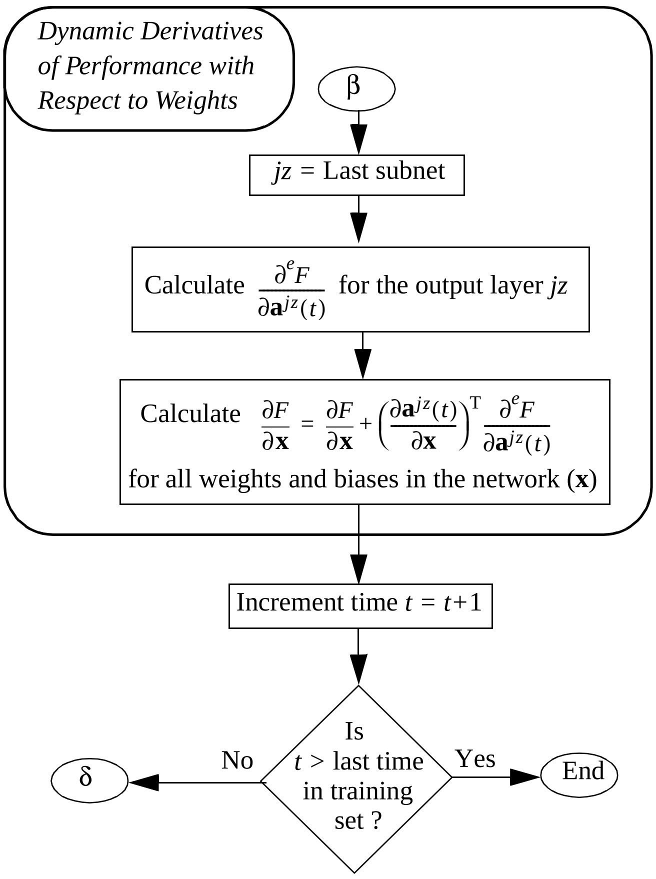
C. COMPLETE FP ALGORITHM FOR THE LDRN

We are now ready to describe a generalized FP algorithm for the arbitrary LDRN. There are two parts to the FP algorithm: Eq. (13) and Eq. (15). Eq. (13) remains the same for the LDRN as for the simple network in Figure 4. Eq. (15), however, must be modified. For the general case we have one equation like Eq. (15) for each subnet output. Each of these equations has a term for the explicit derivative and one additional term for each subnet output.

The complete FP algorithm for the LDRN network is given in the following flowchart. It contains three major sections. The first section computes the explicit (static) derivatives, as in Eq. (21), which are needed as part of the dynamic equations. The second section computes the dynamic derivatives of the subnet outputs with respect to the weights, as in Eq. (15). The final section computes the dynamic derivatives of performance with respect to the weights, as in Eq. (13).


Start
Initialize $\frac{\partial F}{\partial \mathbf{x}} = 0$ for all weights and biases in the network ($\mathbf{x}$); Set initial time $t = 1$
δ
Explicit (Static) Derivatives
Select i = last layer in backpropagation order
Select i = next layer in backpropagation order
Is i output of a subnet?
No
Yes
$\frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{a}^i(t)} = \sum_k \mathbf{LW}^{k,i} \times \mathbf{S}^{k,jz}$ for all k layers connected to layer i
Set $jz = i$ Set $\frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{a}^{jz}(t)} = \mathbf{I}$
Calculate $\frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{n}^i(t)}$
Calculate $\frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{x}^i}$ for weights and biases in layer i ($\mathbf{x}^i$)
Is i input of a subnet?
Yes
For each layer j that connects to layer i calculate $\frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \hat{\mathbf{a}}^{jz,j}(t)}$
No
End backprop-agation ?
No
Yes
α


Dynamic Derivatives of Network Outputs with Respect to Weights
α
jz = First subnet; jj = First subnet
Initialize $\frac{\partial \mathbf{a}^{jz}(t)}{\partial \mathbf{x}} = \frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{x}}$ for all weights and biases ($\mathbf{x}$)
Is the output of subnet jj an input to subnet jz
No
Yes
Calculate: $\frac{\partial \mathbf{a}^{jz}(t)}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^{jz}(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \vec{\mathbf{a}}^{jz,jj}(t)} \frac{\partial \vec{\mathbf{a}}^{jz,jj}(t)}{\partial \mathbf{x}}$ for all weights and biases in network ($\mathbf{x}$).
Increment jj
Is jj > last subnet ?
No
Yes
Increment jz; jj = First subnet
Is jz > last subnet ?
No
Yes
β



Dynamic Derivatives of Performance with Respect to Weights
β
jz = Last subnet
Calculate $\frac{\partial^e F}{\partial \mathbf{a}^{jz}(t)}$ for the output layer jz
Calculate $\frac{\partial F}{\partial \mathbf{x}} = \frac{\partial F}{\partial \mathbf{x}} + \left(\frac{\partial \mathbf{a}^{jz}(t)}{\partial \mathbf{x}}\right)^T \frac{\partial^e F}{\partial \mathbf{a}^{jz}(t)}$ for all weights and biases in the network ($\mathbf{x}$)
Increment time $t = t+1$
Is $t >$ last time in training set ?
No
Yes
δ
End

V. NEUROCONTROL APPLICATION

In this section we illustrate the application of the LDRN and dynamic backpropagation to a control problem. We wish to control the output of an acoustic transmitter, whose schematic diagram is shown in Figure 6.

Binary data is transmitted via acoustic signals from one pipeline location to another. Acoustic stress waves are imparted into the pipeline by the acoustic transmitter. The stress waves propagate through the pipeline to the acoustic receiver, which receives the transmitted signal. Tone bursts of different frequencies are used to represent either a 1 or a 0. The acoustic channel provided by the pipeline causes heavy distortion in the transmitted signal. There are often extraneous unwanted acoustic stress waves created by external sources, such as engines, geartrains, and pumps, that are imparted into the pipeline. The inherent channel distortion and the unwanted external noise can degrade the transmitted signal such that signal detection and interpretation at the receiver are unreliable or impossible. One method of enhancing communication performance is to equalize the effects of the transmission channel by adjusting the transmitter output so that the measured signal imparted to the pipeline at a short distance from the transmitter is the desired signal. Ideally, the feedback measurement of this signal is taken at the receiver. Of course, in practice, the feedback signal is measured near the transmitter. To alleviate the effects of unwanted disturbance noise, the transmitter can actively cancel the disturbance noise by way of destructive interference. The transmitter creates stress waves that are out of phase with, and of equal magnitude to, the undesired signals. In this example, a neural network controller is used as both a channel equalizer and an active noise canceller.

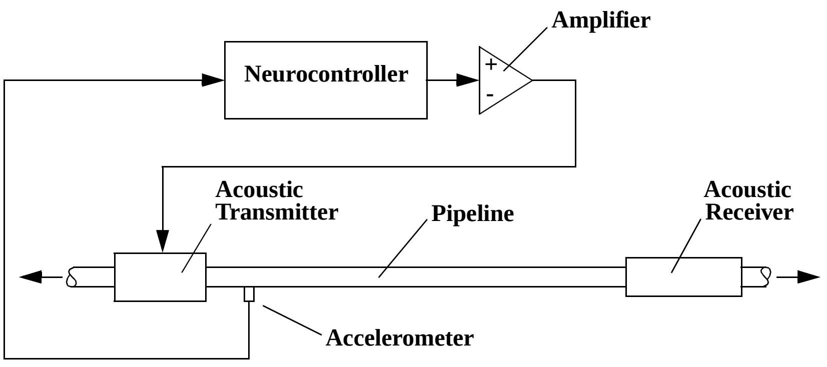


Figure 6. Schematic of Acoustic Transmitter/Receiver System.

In addition to illustrating dynamic backpropagation on the neurocontroller for the acoustic transmitter, this section also demonstrates the effect of training with approximations to true dynamic derivatives. The evaluation is based on squared error performance and floating point operations. When approximations are used, the computational burden can be reduced, but the errors generally increase.

Figure 7 is a schematic of the control system. In this system, model reference adaptive control (MRAC) [Narendra, 1990] is applied to the control of the acoustic transmitter. The plant model is used only as a backpropagation path for the derivatives needed to adjust the controller weights; the plant model weights are not adjusted. The plant model is a 2-layer LDRN, with 10 input taps, 50 feedback taps, and 15 hidden neurons. The controller weights are adjusted such that the error $\mathbf{e}_c(t)$, between a delayed reference input $\mathbf{r}(t)$ and the actual plant output $\mathbf{c}(t)$, is minimized. The controller structure consists of 40 input taps, 50 controller feedback taps, 75 plant output feedback taps, and 15 hidden neurons.

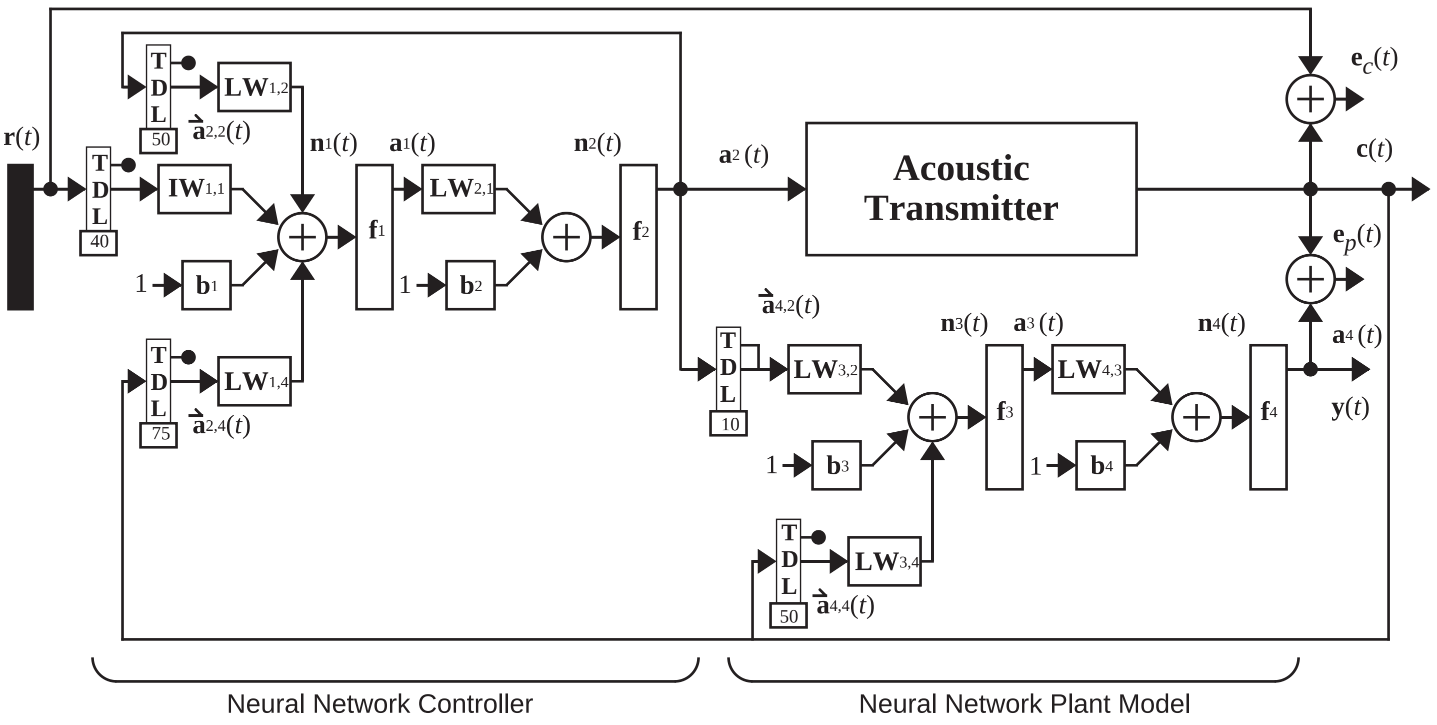


Figure 7. Self-Equalizing Acoustic Transmitter

If we apply the concepts described in the previous sections to the system shown in Figure 7, we notice that the total system is an LDRN that can be divided into two subnets. The last subnet (number 4) corresponds to the Neural Network Plant Model (with $\mathbf{a}^4(t)$ as the subnet output). The first subnet (number 2) corresponds to the Neural Controller (with $\mathbf{a}^2(t)$ as the subnet output). The subnets are fully connected, so we have two sets of training equations:

$$\frac{\partial \mathbf{a}^4(t)}{\partial \mathbf{x}} = \frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^4(t)}{\partial \hat{\mathbf{a}}^{4,2}(t)} \frac{\partial \hat{\mathbf{a}}^{4,2}(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^4(t)}{\partial \hat{\mathbf{a}}^{4,4}(t)} \frac{\partial \hat{\mathbf{a}}^{4,4}(t)}{\partial \mathbf{x}} \tag{23}$$

and

$$\frac{\partial \mathbf{a}^2(t)}{\partial \mathbf{x}} = \frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^2(t)}{\partial \hat{\mathbf{a}}^{2,4}(t)} \frac{\partial \hat{\mathbf{a}}^{2,4}(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^2(t)}{\partial \hat{\mathbf{a}}^{2,2}(t)} \frac{\partial \hat{\mathbf{a}}^{2,2}(t)}{\partial \mathbf{x}} \tag{24}$$

We now show how these equations can be developed using our general FP algorithm, which was described in the flowchart in the previous section. We start in the last layer in the backpropagation order (layer 4), obtaining:

$$\frac{\partial^e \mathbf{a}^4(t)}{\mathbf{a}^4(t)} = \mathbf{I}\ ;\ \mathbf{S}^{4,4} \equiv \frac{\partial^e \mathbf{a}^4(t)}{\mathbf{n}^4(t)} = \dot{\mathbf{F}}^4(\mathbf{n}^4)$$

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{b}^4} = \dot{\mathbf{F}}^4(\mathbf{n}^4)\ ;\ \frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{LW}^{4,3}} = \dot{\mathbf{F}}^4(\mathbf{n}^4) \cdot [\mathbf{a}^3(t)]^{\mathrm{T}}$$

Layer 3 is not the output of a subnet, so we apply:

$$\frac{\partial^e \mathbf{a}^4(t)}{\mathbf{a}^3(t)} = \mathbf{LW}^{4,3} \times \mathbf{S}^{4,4}$$

$$\mathbf{S}^{3,4} \equiv \frac{\partial^e \mathbf{a}^4(t)}{\mathbf{n}^3(t)} = \mathbf{LW}^{4,3} \times \mathbf{S}^{4,4} \times \dot{\mathbf{F}}^3(\mathbf{n}^3)$$

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{b}^3} = \mathbf{LW}^{4,3} \times \mathbf{S}^{4,4} \times \dot{\mathbf{F}}^3(\mathbf{n}^3)$$

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{LW}^{3,2}} = \mathbf{LW}^{4,3} \times \mathbf{S}^{4,4} \times \dot{\mathbf{F}}^3(\mathbf{n}^3) \times [\hat{\mathbf{a}}^{4,2}(t)]^{\mathrm{T}}$$

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{LW}^{3,4}} = \mathbf{LW}^{4,3} \times \mathbf{S}^{4,4} \times \dot{\mathbf{F}}^3(\mathbf{n}^3) \times [\bar{\mathbf{a}}^{4,4}(t)]^T$$

Layer 3 has two inputs with delays, therefore it is the beginning of the last subnet, and we calculate:

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \bar{\mathbf{a}}^{4,2}(t)} = \mathbf{LW}^{4,3} \times \mathbf{S}^{4,4} \times \dot{\mathbf{F}}^3(\mathbf{n}^3) \times \mathbf{LW}^{3,2}$$

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \bar{\mathbf{a}}^{4,4}(t)} = \mathbf{LW}^{4,3} \times \mathbf{S}^{4,4} \times \dot{\mathbf{F}}^3(\mathbf{n}^3) \times \mathbf{LW}^{3,4}$$

Layer 2 in the neural controller is the end of the first subnet. So we apply the equations:

$$\frac{\partial^e \mathbf{a}^2(t)}{\mathbf{a}^2(t)} = \mathbf{I}\,;\ \mathbf{S}^{2,2} \equiv \frac{\partial^e \mathbf{a}^2(t)}{\mathbf{n}^2(t)} = \dot{\mathbf{F}}^2(\mathbf{n}^2)$$

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{b}^2} = \dot{\mathbf{F}}^2(\mathbf{n}^2)\,;\ \frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{LW}^{2,1}} = \dot{\mathbf{F}}^2(\mathbf{n}^2) \cdot [\mathbf{a}^1(t)]^T$$

Layer 1 is not the output of a subnet, so we apply:

$$\frac{\partial^e \mathbf{a}^2(t)}{\mathbf{a}^1(t)} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2}$$

$$\mathbf{S}^{1,2} \equiv \frac{\partial^e \mathbf{a}^2(t)}{\mathbf{n}^1(t)} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1)$$

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{b}^1} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1)$$

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{LW}^{1,1}} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1) \times [\bar{\mathbf{a}}^{2,2}(t)]^T$$

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{LW}^{4,1}} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1) \times [\bar{\mathbf{a}}^{2,4}(t)]^T$$

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{IW}^{1,1}} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1) \times [\hat{\mathbf{r}}(t)]^{\mathrm{T}}$$

Layer 1 has two inputs with delays, so it is the end of the first subnet, and we calculate

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \hat{\mathbf{a}}^{2,2}(t)} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1) \times \mathbf{LW}^{1,2}$$

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \hat{\mathbf{a}}^{2,4}(t)} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1) \times \mathbf{LW}^{1,4}$$

Now that we have finished with the backpropagation step, we have the explicit derivatives for all of the weights and biases in the system. We are now ready to calculate the dynamic derivatives. We initialize

$$\frac{\partial \mathbf{a}^2(t)}{\partial \mathbf{x}} = \frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{x}}$$

and calculate

$$\frac{\partial \mathbf{a}^2(t)}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^2(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^2(t)}{\partial \hat{\mathbf{a}}^{2,2}(t)} \frac{\partial \hat{\mathbf{a}}^{2,2}(t)}{\partial \mathbf{x}}$$

and

$$\frac{\partial \mathbf{a}^2(t)}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^2(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^2(t)}{\partial \hat{\mathbf{a}}^{2,4}(t)} \frac{\partial \hat{\mathbf{a}}^{2,4}(t)}{\partial \mathbf{x}} .$$

This gives us Eq. (24) for all weights and biases. A similar process is performed for subnet 4 to obtain Eq. (23).

We have now computed all of the dynamic derivatives of the outputs of the subnets with respect to the weights. The next step is to compute the derivatives of the performance function with respect to the weights. We must first calculate

$$\frac{\partial^e F}{\partial \mathbf{a}^4(t)} = -2(\mathbf{r}(t) - \mathbf{a}^4(t)) ,$$

to obtain

$$\frac{\partial F}{\partial \mathbf{x}} = \frac{\partial F}{\partial \mathbf{x}} + \left(\frac{\partial \mathbf{a}^4(t)}{\partial \mathbf{x}}\right)^T \frac{\partial^e F}{\partial \mathbf{a}^4(t)}$$

for all the weights and biases in the neural controller. The process is repeated for each sample time in the training set.

The LDRN was trained using the preceding equations. Now we demonstrate the performance of the closed-loop acoustic transmitter system. The reference input used in the simulation consisted of a random sequence of tone burst pulses, as shown in Figure 8. The tone bursts are evenly spaced and appear randomly at one of two frequencies. In order to investigate the robustness of the controller, a periodic disturbance noise was added to the plant input, representing extraneous acoustic stress waves created by external sources. The open-loop plant response, with no controller, is shown in Figure 9. The closed-loop response, after the controller was trained to convergence, is shown in Figure 10.

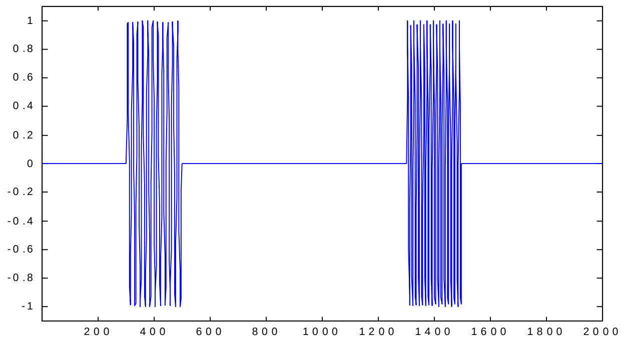


Figure 8. Tone Burst Reference Input

In the next test, dynamic backpropagation was used in plant model, but was not used in backpropagating derivatives in the controller. Only static backpropagation was used in the controller. (In Eq. (24) only the explicit derivative terms are calculated.) This procedure requires less computation than the full dynamic backpropagation but may not be as accurate. The results are shown in Figure 11.

For the last test, 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 (static) derivatives. The controller weights never converged in this case, so a plot of the results is not shown. The results are summarized in Table 1.

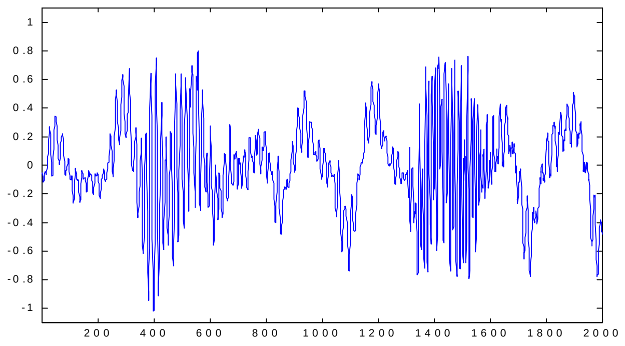


Figure 9. Open-loop Plant Response

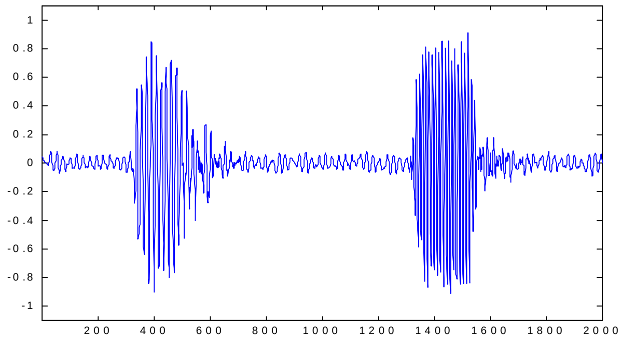


Figure 10. Closed-Loop System Response (Full Dynamic Training)

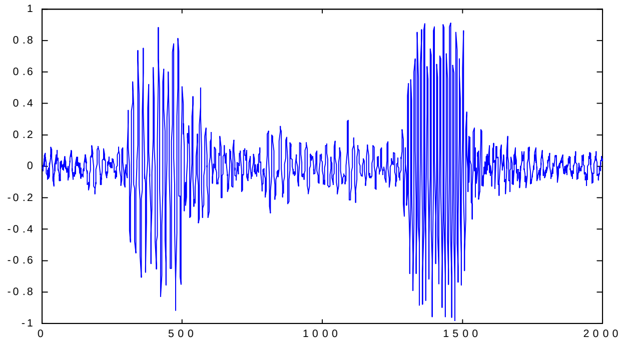


Figure 11. Response without Dynamic Controller Training

Table 1 provides a summary of the performance results for all three tests. These results highlight the increased computational burden when calculating dynamic derivatives instead of simply static derivatives. 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 for the Neural Controller

VI. RECURRENT FILTER

This section provides a second example of the application of the LDRN and dynamic backpropagation. We use a multi-loop recurrent network to predict an experimental acoustic signal. The prediction of acoustic signals is crucial in active sound cancellation systems. Acoustic environments are often very complex, due to the complexity of typical sound sources and the presence of reflected sound waves. The dynamic nature of acoustical systems makes the use of recurrent filter structures of great interest for prediction and control of this type of system.

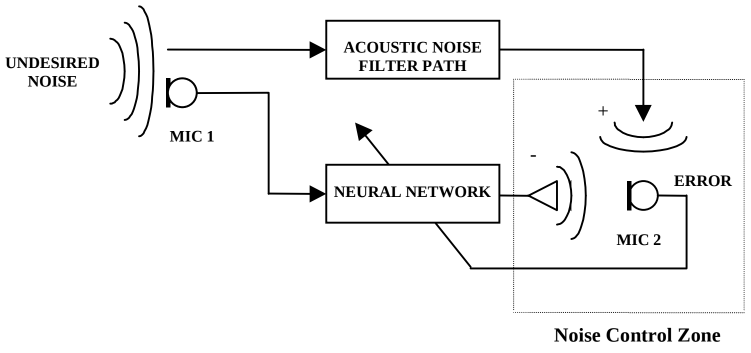


Figure 12. Active Noise Control System

Figure 12 depicts a typical Active Noise Control (ANC) system. An acoustic noise source creates undesirable noise in a surrounding area. The goal of the active noise suppression system is to reduce the undesirable noise at a particular location by using a loudspeaker to produce “anti-noise” that attenuates the unwanted noise by destructive interference. In order for a system of this type to work effectively, it is critical that the ANC system be able to predict (and then cancel) unwanted sound in the noise control zone.

In the first part of this section we develop the dynamic training equations for the ANC system. Then we present experimental results showing the prediction performance.

Figure 13 shows the structure of the LDRN used for predicting the acoustic data. In this network there are 3 cascaded recurrent structures. If we follow the methods described in Section IV, we see that the system is composed of three subnets. Therefore, we have three sets of training equations:

$$\frac{\partial \mathbf{a}^6(t)}{\partial \mathbf{x}} = \frac{\partial^e \mathbf{a}^6(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^6(t)}{\partial \vec{\mathbf{a}}^{6,4}(t)} \frac{\partial \vec{\mathbf{a}}^{6,4}(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^6(t)}{\partial \vec{\mathbf{a}}^{6,6}(t)} \frac{\partial \vec{\mathbf{a}}^{6,6}(t)}{\partial \mathbf{x}} \tag{25}$$

$$\frac{\partial \mathbf{a}^4(t)}{\partial \mathbf{x}} = \frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^4(t)}{\partial \vec{\mathbf{a}}^{4,2}(t)} \frac{\partial \vec{\mathbf{a}}^{4,2}(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^4(t)}{\partial \vec{\mathbf{a}}^{4,4}(t)} \frac{\partial \vec{\mathbf{a}}^{4,4}(t)}{\partial \mathbf{x}} \tag{26}$$

$$\frac{\partial \mathbf{a}^2(t)}{\partial \mathbf{x}} = \frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{x}} + \frac{\partial^e \mathbf{a}^2(t)}{\partial \vec{\mathbf{a}}^{2,2}(t)} \frac{\partial \vec{\mathbf{a}}^{2,2}(t)}{\partial \mathbf{x}} \tag{27}$$

Notice that in Eq. (27) there is only one dynamic term. This is because there is only one tapped-delay input that comes from a subnet.

We now show how these equations can be developed using our general FP procedure, which was described in the flowchart of Section IV. We start in the last layer in the backpropagation order (Layer 6) to get the following equations:

$$\frac{\partial^e \mathbf{a}^6(t)}{\partial \mathbf{a}^6(t)} = \mathbf{I}\,,\ \mathbf{S}^{6,6} \equiv \frac{\partial^e \mathbf{a}^6(t)}{\partial \mathbf{n}^6(t)} = \dot{\mathbf{F}}^6(\mathbf{n}^6)$$

$$\frac{\partial^e \mathbf{a}^6(t)}{\partial \mathbf{b}^6} = \dot{\mathbf{F}}^6(\mathbf{n}^6)\,,\ \frac{\partial^e \mathbf{a}^6(t)}{\partial \mathbf{LW}^{6,5}} = \dot{\mathbf{F}}^6(\mathbf{n}^6) \cdot [\mathbf{a}^5(t)]^T$$

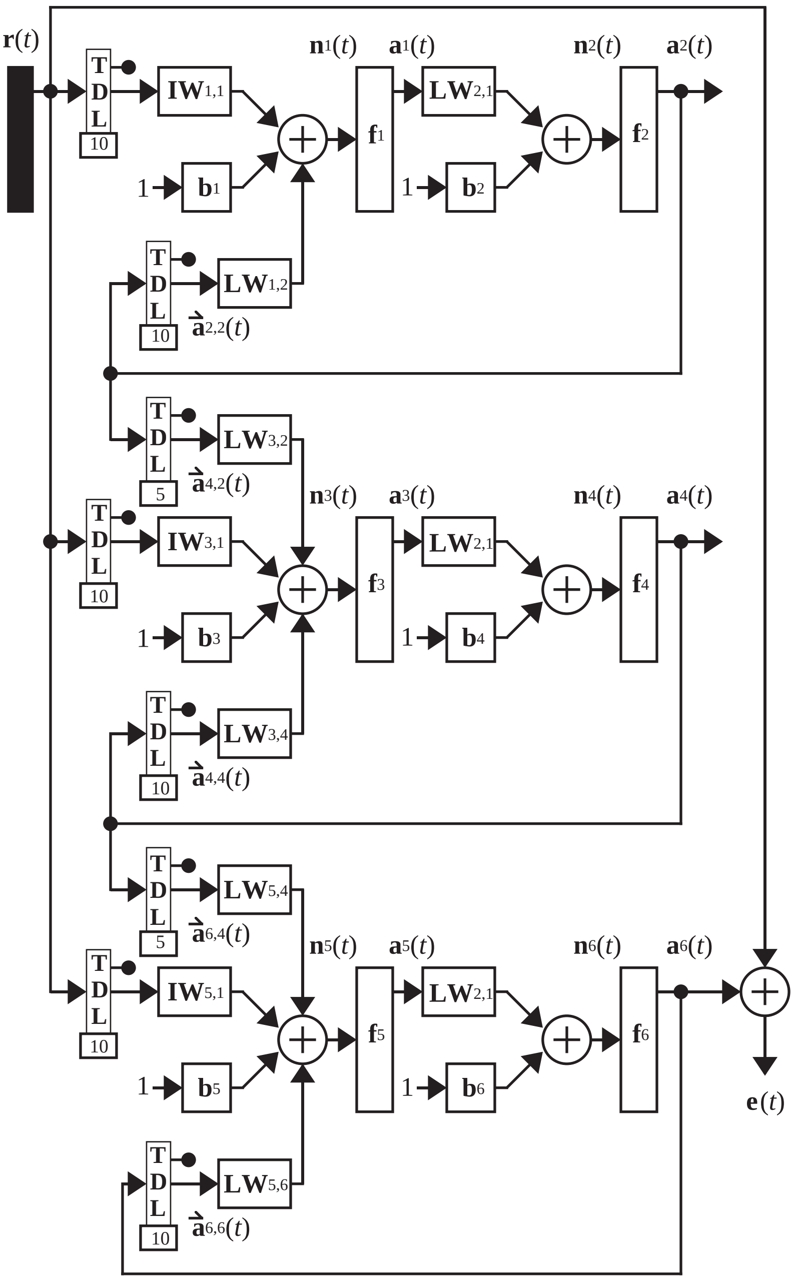


Figure 13. Cascaded Recurrent Neural Network

Layer 5 is not the output of a subnet, so the resulting equations are:

$$\frac{\partial^e \mathbf{a}^6(t)}{\partial \mathbf{a}^5(t)} = \mathbf{LW}^{6,5} \times \mathbf{S}^{6,6}$$

$$\mathbf{S}^{5,6} \equiv \frac{\partial^e \mathbf{a}^6(t)}{\partial \mathbf{n}^5(t)} = \mathbf{LW}^{6,5} \times \mathbf{S}^{6,6} \times \dot{\mathbf{F}}^5(\mathbf{n}^5)$$

$$\frac{\partial^e \mathbf{a}^6(t)}{\partial \mathbf{b}^5} = \mathbf{LW}^{6,5} \times \mathbf{S}^{6,6} \times \dot{\mathbf{F}}^5(\mathbf{n}^5)$$

$$\frac{\partial^e \mathbf{a}^6(t)}{\partial \mathbf{LW}^{5,4}} = \mathbf{LW}^{6,5} \times \mathbf{S}^{6,6} \times \dot{\mathbf{F}}^5(\mathbf{n}^5) \times [\hat{\mathbf{a}}^{6,4}(t)]^{\mathrm{T}}$$

$$\frac{\partial^e \mathbf{a}^6(t)}{\partial \mathbf{IW}^{5,1}} = \mathbf{LW}^{6,5} \times \mathbf{S}^{6,6} \times \dot{\mathbf{F}}^5(\mathbf{n}^5) \times [\hat{\mathbf{r}}(t)]^{\mathrm{T}}$$

$$\frac{\partial^e \mathbf{a}^6(t)}{\partial \mathbf{LW}^{5,6}} = \mathbf{LW}^{6,5} \times \mathbf{S}^{6,6} \times \dot{\mathbf{F}}^5(\mathbf{n}^5) \times [\hat{\mathbf{a}}^{6,6}(t)]^{\mathrm{T}}$$

Layer 5 has two tapped-delay inputs from subnets, so we must calculate the explicit derivatives of the subnet output with respect to these inputs to yield:

$$\frac{\partial^e \mathbf{a}^6(t)}{\partial \hat{\mathbf{a}}^{6,4}(t)} = \mathbf{LW}^{6,5} \times \mathbf{S}^{6,6} \times \dot{\mathbf{F}}^5(\mathbf{n}^5) \times \mathbf{LW}^{5,4}$$

$$\frac{\partial^e \mathbf{a}^6(t)}{\partial \hat{\mathbf{a}}^{6,6}(t)} = \mathbf{LW}^{6,5} \times \mathbf{S}^{6,6} \times \dot{\mathbf{F}}^5(\mathbf{n}^5) \times \mathbf{LW}^{5,6}$$

Layer 4 is the end of the second subnet, so we now calculate the explicit derivatives with respect to the second subnet output.

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{a}^4(t)} = \mathbf{I}\ ;\ \mathbf{S}^{4,4} \equiv \frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{n}^4(t)} = \dot{\mathbf{F}}^4(\mathbf{n}^4)$$

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{b}^4} = \dot{\mathbf{F}}^4(\mathbf{n}^4)\,;\ \frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{LW}^{4,3}} = \dot{\mathbf{F}}^4(\mathbf{n}^4)\cdot[\mathbf{a}^3(t)]^T$$

Layer 3 is not the output of a subnet, so the resulting equations are:

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{a}^3(t)} = \mathbf{LW}^{4,3}\times\mathbf{S}^{4,4}$$

$$\mathbf{S}^{3,4} \equiv \frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{n}^3(t)} = \mathbf{LW}^{4,3}\times\mathbf{S}^{4,4}\times\dot{\mathbf{F}}^3(\mathbf{n}^3)$$

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{b}^3} = \mathbf{LW}^{4,3}\times\mathbf{S}^{4,4}\times\dot{\mathbf{F}}^3(\mathbf{n}^3)$$

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{LW}^{3,2}} = \mathbf{LW}^{4,3}\times\mathbf{S}^{4,4}\times\dot{\mathbf{F}}^3(\mathbf{n}^3)\times[\bar{\mathbf{a}}^{4,2}(t)]^T$$

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{IW}^{3,1}} = \mathbf{LW}^{4,3}\times\mathbf{S}^{4,4}\times\dot{\mathbf{F}}^3(\mathbf{n}^3)\times[\bar{\mathbf{r}}(t)]^T$$

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \mathbf{LW}^{3,4}} = \mathbf{LW}^{4,3}\times\mathbf{S}^{4,4}\times\dot{\mathbf{F}}^3(\mathbf{n}^3)\times[\bar{\mathbf{a}}^{4,4}(t)]^T$$

Layer 3 has two delayed inputs from other subnets, so we must compute the following explicit derivatives:

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \bar{\mathbf{a}}^{4,2}(t)} = \mathbf{LW}^{4,3}\times\mathbf{S}^{4,4}\times\dot{\mathbf{F}}^3(\mathbf{n}^3)\times\mathbf{LW}^{3,2}$$

$$\frac{\partial^e \mathbf{a}^4(t)}{\partial \bar{\mathbf{a}}^{4,4}(t)} = \mathbf{LW}^{4,3}\times\mathbf{S}^{4,4}\times\dot{\mathbf{F}}^3(\mathbf{n}^3)\times\mathbf{LW}^{3,4}$$

Layer 2 is the end of the first subnet, so we apply the equations

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{a}^2(t)} = \mathbf{I}\,;\ \mathbf{S}^{2,2} \equiv \frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{n}^2(t)} = \dot{\mathbf{F}}^2(\mathbf{n}^2)$$

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{b}^2} = \dot{\mathbf{F}}^2(\mathbf{n}^2) \; ; \; \frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{LW}^{2,1}} = \dot{\mathbf{F}}^2(\mathbf{n}^2) \cdot [\mathbf{a}^1(t)]^{\mathrm{T}}$$

Layer 1 is not the output of a subnet, so we apply

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{a}^1(t)} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2}$$

$$\mathbf{S}^{1,2} \equiv \frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{n}^1(t)} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1)$$

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{b}^1} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1)$$

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{IW}^{1,1}} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1) \times [\grave{\mathbf{r}}(t)]^{\mathrm{T}}$$

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \mathbf{LW}^{1,2}} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1) \times [\grave{\mathbf{a}}^{2,2}(t)]^{\mathrm{T}}$$

Layer 1 has one delayed input from another subnet, so we calculate

$$\frac{\partial^e \mathbf{a}^2(t)}{\partial \grave{\mathbf{a}}^{2,2}(t)} = \mathbf{LW}^{2,1} \times \mathbf{S}^{2,2} \times \dot{\mathbf{F}}^1(\mathbf{n}^1) \times \mathbf{LW}^{1,2}$$

At this point, we have the explicit derivatives for all the weights and biases in the system. These explicit derivatives are used with Eq. (25) – Eq. (27) to compute the dynamic derivatives we need for training the network. Notice that the solution to Eq. (27) is an input to Eq. (26) and Eq. (27) on the following time step. The solution to Eq. (26) is an input to Eq. (25) and Eq. (26) on the following time step. Finally, the solution to Eq. (25) is an input to Eq. (25) on the following time step.

After all the dynamic derivatives of the output of the system with respect to the weights and biases have been computed, we must calculate

$$\frac{\partial^e F}{\partial \mathbf{a}^6(t)} = -2(\mathbf{r}(t) - \mathbf{a}^6(t))$$

We can then compute the derivative of the cost function with respect to all the weights and biases using

$$\frac{\partial F}{\partial \mathbf{x}} = \frac{\partial F}{\partial \mathbf{x}} + \left(\frac{\partial \mathbf{a}^6(t)}{\partial \mathbf{x}}\right)^{\mathrm{T}} \frac{\partial^e F}{\partial \mathbf{a}^6(t)}$$

The process is repeated for each sample time in the training set.

After the network was trained, it was used to predict experimentally recorded noise. The result is shown in Figure 14. The data was collected in an acoustically "live" environment that was conducive to sound reflection. The prediction results for the LDRN, trained with full dynamic backpropagation, is compared to two other systems. The first comparison system is an LDRN that is trained only with static derivatives. The second comparison system is a non-recurrent LFFN system with a tapped-delay line at the input. Figure 14 shows the actual and predicted signals when full dynamic backpropagation is used to train the LDRN. Figure 15 is a plot of the errors between actual and predicted signals.

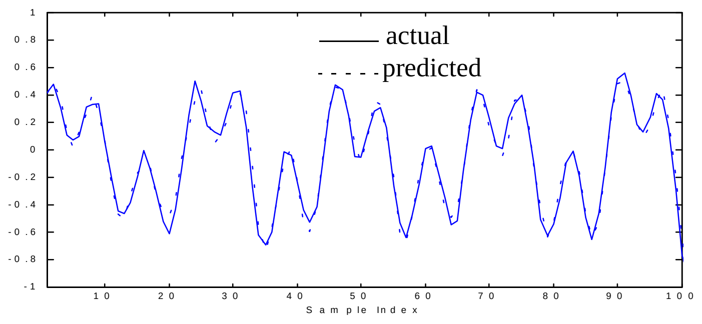


Figure 14. Prediction Results for LDRN with Full Dynamic Training

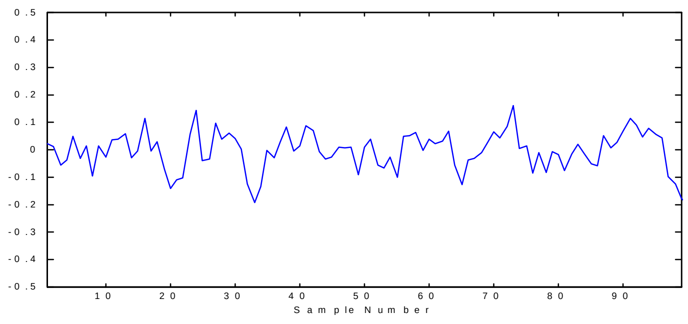


Figure 15. Errors for LDRN with Full Dynamic Training

The next experiment uses the same data, but only explicit (static) derivatives were used. The errors between actual and predicted signals are shown in Figure 16.

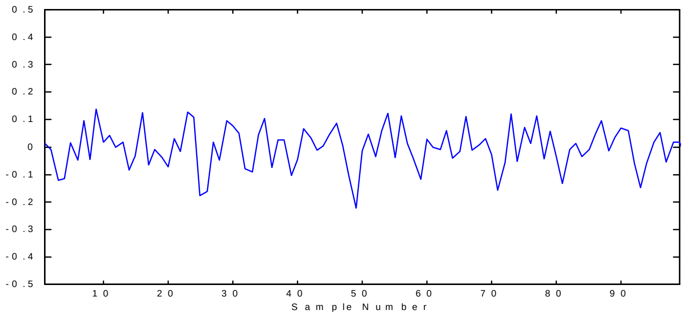


Figure 16. Prediction Errors for Static Training

The results shown in Figure 16 are reasonably good. We can see some degradation in performance, which might be critical in certain situations. In sound cancellation applications, for example, the differences would certainly be detectable by the human ear.

As a final experiment, the data was processed using an LFFN, with TDL input. The network size was adjusted so that the number of weights was comparable to the LDRN used in the previous experiment. The prediction errors for the LFFN are shown in Figure 17.

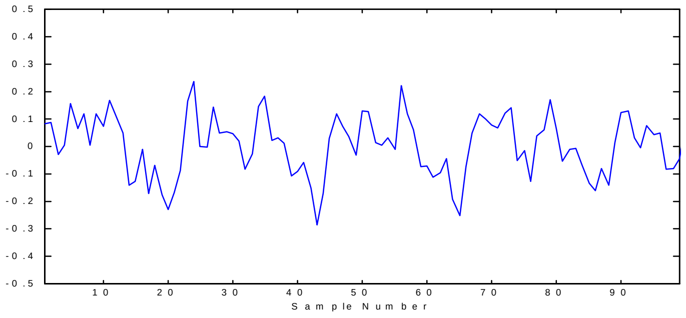


Figure 17. LFFN Prediction Results

The LFFN prediction performance is not too bad, but is significantly worse than the LDRN performance, when trained with full dynamic backpropagation. A summary of the simulation results is provided in Table 2. Notice the dramatic increase in floating point operations required to process each sample when dynamic training is used.

Prediction Method	Flops/Sample	Mean Squared Prediction Error
LDRN Full Dynamic Training	4.32×10^4	.0050
LDRN Static Training	5.19×10^3	.0087
LFFN	1.85×10^3	.0120

Table 2. Simulation Results

VII. SUMMARY

This chapter has discussed the training of recurrent neural networks for control and signal processing. When computing training gradients in recurrent networks, there are two different effects that we must account for. The first is a direct effect, which explains the immediate impact of a change in the weights on the output of the network at the current time. The second is an indirect effect, which accounts for the fact that some of the inputs to the network are previous network outputs, which are also functions of the weights. To account for this indirect effect we must use dynamic backpropagation.

This chapter has introduced the Layered Digital Recurrent Network (LDRN), which is a general class of recurrent network. A universal dynamic training algorithm for the LDRN was also developed. The LDRN was then applied to problems in control and signal processing. A number of practical issues must be addressed when applying dynamic training. Computational requirements for dynamic training can be much higher than those for static training, but static training is not as accurate and may not converge. The appropriate form of training to use (dynamic, static, or some combination) varies from problem to problem.

REFERENCES

Demuth, H. B. and Beale, M., *Users' Guide for the Neural Network Toolbox for MATLAB*, The Mathworks, Natick, MA, 1998.

Hagan, M.T., Demuth, H. B., Beale, M., *Neural Network Design*, PWS Publishing Company, Boston, 1996.

Narendra, K. S. and Parthasrathy, A. M., Identification and control for dynamic systems using neural networks, *IEEE Transactions on Neural Networks*, 1(1), 4, 1990.

Yang, W., *Neurocontrol Using Dynamic Learning*, Doctoral Thesis, Oklahoma State University, Stillwater, 1994.

Yang, W. and Hagan, M.T., Training recurrent networks, *Proceedings of the 7th Oklahoma Symposium on Artificial Intelligence*, Stillwater, 226, 1993.