Backpropagation Through Time for a General Class of Recurrent Network

Orlando De Jesús and Martin T. Hagan
School of Electrical and Computer Engineering
Oklahoma State University
Stillwater, OK
email:dorland@okstate.com, mhagan@okstate.edu

Abstract

This paper introduces a general class of dynamic network - the Layered Digital Dynamic Network. It then derives the backpropagation-through-time algorithm for computing the gradient of the network error with respect to the weights of the network.

1 Introduction

Neural networks can be classified into dynamic and static categories. Static (feedforward) networks have no feedback elements and contain no delays; the output is calculated directly from the input through feedforward connections. In dynamic networks the output depends not only on the current input to the network, but also on the current or previous inputs, outputs or states of the network. There are two general approaches to gradient calculations in dynamic networks: backpropagation-through-time (BTT) and forward perturbation. In this paper we will discuss BTT. The BTT algorithm has been described as a basic concept [6], and it has been developed in detail for specific network architectures. In this paper, we develop it for a general class of network structure.

This paper introduces the Layered Digital Dynamic Network (LDDN) and develops a general training algorithm for this network. Section 2 contains a discussion of the backpropagation algorithms that are required to compute training gradients for dynamic networks. The concepts underlying the backpropagation-through-time (BTT) algorithm are demonstrated for a simple, single-loop dynamic network. In Section 3 we present the notation necessary to represent the LDDN. In Section 4 we describe a general backpropagation-through-time algorithm for computing training gradients for the LDDN.

2 Example of Dynamic Gradient Calculations

To illustrate dynamic backpropagation ([7], [8]), consider Figure 1, which is a simple dynamic 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.

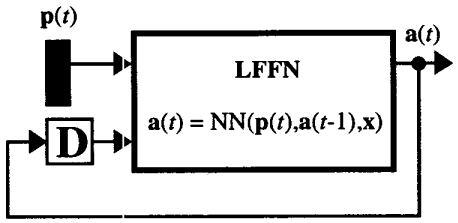

Figure 1: Simple Dynamic Network

Now suppose that we want to minimize

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

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. One is called backpropagation through time (BTT) and the other is called forward perturbation (FP). They both use the chain rule, but are implemented in different ways. (See [1] for a discussion of the forward perturbation method.) In this paper we concentrate on the BTT algorithm, which begins with the following equation

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \left[\frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{x}}\right]^T \times \frac{\partial F}{\partial \mathbf{a}(t)} \tag{2}$$

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

0-7803-7044-9/01/$10.00 ©2001 IEEE

algorithm, as in [3]. To find the complete derivatives that are required in Eq. (2), we need the additional equation:

$$\frac{\partial F}{\partial \mathbf{a}(t)} = \frac{\partial^e F}{\partial \mathbf{a}(t)} + \frac{\partial^e \mathbf{a}(t+1)}{\partial \mathbf{a}(t)} \times \frac{\partial F}{\partial \mathbf{a}(t+1)} \tag{3}$$

Eq. (2) and Eq. (3) make up the backpropagation-through-time (BTT) algorithm. Here the key term is

$$\frac{\partial F}{\partial \mathbf{a}(t)} \tag{4}$$

which must be propagated backward through time.

In the next sections we generalize the BTT algorithm, so that it can be applied to arbitrary LDDN's.

3 Definition of the Layered Digital Dynamic Network

To explain the general BTT algorithm, we must first define the LDDN. We do that in this section.

First, as we stated earlier, a *layer* consists of a set of *weights*, associated *tapped delay lines*, a *summing junction*, 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, some layers of the LDDN will be assigned as network outputs, and some will be assigned as network inputs. A layer is an *input layer* if it has an input weight, or if it contains any delays with any of its weight matrices. A layer is an *output layer* if its output will be compared to a target during training, or if it is connected to an input layer through a matrix which has any delays associated with it.

For example, the LDDN shown in Figure 2 has two output layers (1 and 3) and two input layers (1 and 2). For this network the simulation order is 1-2-3, and the backpropagation order is 3-2-1. As an aid in later derivations, we will define U as the set of all output layer numbers and X as the set of all input layer numbers. For the LDDN in Figure 2, $U=\{1,3\}$ and $X=\{1,2\}$.

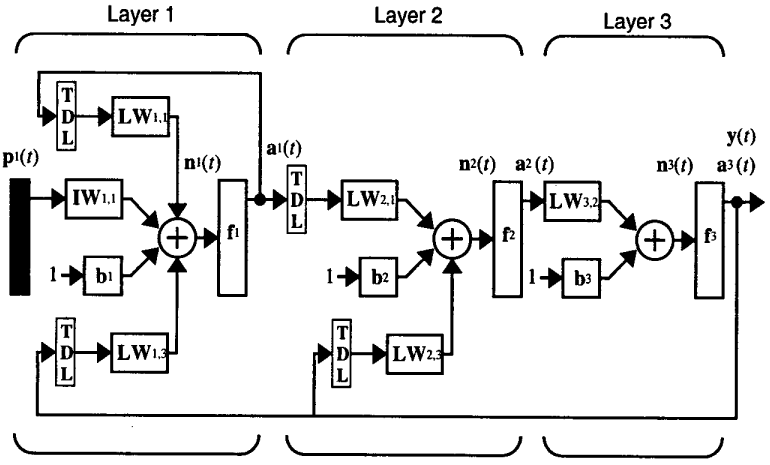

Figure 2: Three-layer LDDN with two output layers (1 and 3) and two input layers (1 and 2)

The general equations for simulating an arbitrary LDDN network are given below. The net input at layer m can be computed as

$$\mathbf{n}^m(t) = \sum_{l \in L_m^f} \sum_{d \in DL_{m,l}} \mathbf{LW}^{m,l}(d)\mathbf{a}^l(t-d) + \sum_{l \in I_m} \sum_{d \in DI_{m,l}} \mathbf{IW}^{m,l}(d)\mathbf{p}^l(t-d) + \mathbf{b}^m \tag{5}$$

where $DL_{m,l}$ is the set of all delays in the tapped delay line between Layer l and Layer m, $DI_{m,l}$ is the set of all delays in the tapped delay line between Input l and Layer m, I_m is the set of indices of input vectors that connect forward to layer m, and L_m^f is the set of indices of layers that directly connect *forward* to layer m. The output of layer m is then computed as

$$\mathbf{a}^m(t) = \mathbf{f}^m(\mathbf{n}^m(t)). \tag{6}$$

At each time point, Eq. (5) and Eq. (6) are iterated forward through the layers, as m is incremented through the simulation order. Time is then incremented from t=1 to t=Q.

4 Backpropagation-Through-Time

In this section we will generalize the Backpropagation-Through-Time (BTT) algorithm, given in Eq. (2) and Eq. (3), for LDDN networks.

4.1 Eq. (2)

The first step is to generalize Eq. (2). For the general LDDN network, we can calculate the terms of the gradient by using the chain rule, as in

$$\frac{\partial F}{\partial lw_{i,j}^{m,l}(d)} = \sum_{t=1}^{Q}\left\{\sum_{u\in U}\sum_{k=1}^{S_u}\frac{\partial F}{\partial a_k^u(t)}\times\frac{\partial^e a_k^u(t)}{\partial n_i^m(t)}\right\}\frac{\partial^e n_i^m(t)}{\partial lw_{i,j}^{m,l}(d)} \tag{7}$$

$$\frac{\partial F}{\partial iw_{i,j}^{m,l}(d)} = \sum_{t=1}^{Q}\left\{\sum_{u\in U}\sum_{k=1}^{S_u}\frac{\partial F}{\partial a_k^u(t)}\times\frac{\partial^e a_k^u(t)}{\partial n_i^m(t)}\right\}\frac{\partial^e n_i^m(t)}{\partial iw_{i,j}^{m,l}(d)} \tag{8}$$

$$\frac{\partial F}{\partial b_i^m} = \sum_{t=1}^{Q}\left\{\sum_{u\in U}\sum_{k=1}^{S_u}\frac{\partial F}{\partial a_k^u(t)}\times\frac{\partial^e a_k^u(t)}{\partial n_i^m(t)}\right\}\frac{\partial^e n_i^m(t)}{\partial b_i^m} \tag{9}$$

where u is an output layer, U is the set of all output layers, and S_u is the number of neurons in layer u.

From Eq. (5), the elements of the net input can be computed as

$$n_i^m(t) = \sum_{l\in L_m^f}\sum_{d\in DL_{m,l}}\left(\sum_{j=1}^{S_l} lw_{i,j}^{m,l}(d)a_j^l(t-d)\right) + \sum_{l\in L_m^f}\sum_{d\in DI_{m,l}}\left(\sum_{j=1}^{S_l} iw_{i,j}^{m,l}(d)p_j^l(t-d)\right) + b_i^m \tag{10}$$

Therefore,

$$\frac{\partial^e n_i^m(t)}{\partial lw_{i,j}^{m,l}(d)} = a_j^l(t-d), \frac{\partial^e n_i^m(t)}{\partial iw_{i,j}^{m,l}(d)} = p_j^l(t-d), \frac{\partial^e n_i^m(t)}{\partial b_i^m} = 1. \tag{11}$$

We will also define

$$d_i^m(t) = \sum_{u\in U}\sum_{k=1}^{S_u}\frac{\partial F}{\partial a_k^u(t)}\times\frac{\partial^e a_k^u(t)}{\partial n_i^m(t)}. \tag{12}$$

The terms of the gradient can then be written

$$\frac{\partial F}{\partial lw_{i,j}^{m,l}(d)} = \sum_{t=1}^{Q} d_i^m(t)a_j^l(t-d), \tag{13}$$

$$\frac{\partial F}{\partial iw_{i,j}^{m,l}(d)} = \sum_{t=1}^{Q} d_i^m(t)p_j^l(t-d), \tag{14}$$

$$\frac{\partial F}{\partial b_i^m} = \sum_{t=1}^{Q} d_i^m(t). \tag{15}$$

If we define the following sensitivity term

$$s_{k,i}^{u,m}(t) \equiv \frac{\partial^e a_k^u(t)}{\partial n_i^m(t)}, \tag{16}$$

which can be used to make up the following matrix

$$\mathbf{S}^{u,m}(t) = \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^m(t)^T} = \begin{bmatrix} s_{1,1}^{u,m}(t) & s_{1,2}^{u,m}(t) & \dots & s_{1,S_m}^{u,m}(t) \\ s_{2,1}^{u,m}(t) & s_{2,2}^{u,m}(t) & \dots & s_{1,S_m}^{u,m}(t) \\ \vdots & & & \\ s_{S_u,1}^{u,m}(t) & s_{S_u,2}^{u,m}(t) & \dots & s_{S_u,S_m}^{u,m}(t) \end{bmatrix} = \left[\mathbf{s}_1^{u,m}(t)\ \mathbf{s}_2^{u,m}(t)\ \dots\ \mathbf{s}_{S_m}^{u,m}(t)\right] = \begin{bmatrix} {}_1\mathbf{s}^{u,m}(t)^T \\ {}_2\mathbf{s}^{u,m}(t)^T \\ \vdots \\ {}_{S_u}\mathbf{s}^{u,m}(t)^T \end{bmatrix} \tag{17}$$

then the elements $d_i^m(t)$ can be written

$$d_i^m(t) = \sum_{u\in U}\sum_{k=1}^{S_u}\frac{\partial F}{\partial a_k^u(t)}\times s_{k,i}^{u,m}(t). \tag{18}$$

In matrix form this becomes

$$\mathbf{d}^m(t) = \sum_{u\in U}[\mathbf{S}^{u,m}(t)]^T\times\frac{\partial F}{\partial \mathbf{a}^u(t)} \tag{19}$$

where

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \begin{bmatrix} \frac{\partial F}{\partial a_1^u(t)} \\ \frac{\partial F}{\partial a_2^u(t)} \\ \vdots \\ \frac{\partial F}{\partial a_{S_u}^u(t)} \end{bmatrix} \tag{20}$$

Now the gradient can be written in matrix form.

$$\frac{\partial F}{\partial \mathbf{LW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^m(t) \times [\mathbf{a}^l(t-d)]^T, \tag{21}$$

$$\frac{\partial F}{\partial \mathbf{IW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^m(t) \times [\mathbf{p}^l(t-d)]^T, \tag{22}$$

$$\frac{\partial F}{\partial \mathbf{b}^m} = \sum_{t=1}^{Q} \mathbf{d}^m(t). \tag{23}$$

Eq. (21) through Eq. (23) make up the generalization of Eq. (2) for the LDDN network. It remains to compute the explicit derivatives of the sensitivity matrix $\mathbf{S}^{u,m}(t)$.

4.2 Explicit Derivatives

In order to compute the elements of the sensitivity matrix, we use a form of backpropagation. The sensitivities at the outputs of the network can be computed as

$$s_{k,i}^{u,u}(t) = \frac{\partial^e a_k^u(t)}{\partial n_i^u(t)} = \begin{cases} \dot{f}^u(n_i^u(t)) & \text{for } i = k \\ 0 & \text{for } i \neq k \end{cases}, \quad u \in U, \tag{24}$$

or, in matrix form,

$$\mathbf{S}^{u,u}(t) = \dot{\mathbf{F}}^u(\mathbf{n}^u(t)), \tag{25}$$

where $\dot{\mathbf{F}}^u(\mathbf{n}^u(t))$ is defined as

$$\dot{\mathbf{F}}^u(\mathbf{n}^u(t)) = \begin{bmatrix} \dot{f}^u(n_1^u(t)) & 0 & \dots & 0 \\ 0 & \dot{f}^u(n_2^u(t)) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \dot{f}^u(n_{S^u}^u(t)) \end{bmatrix} \tag{26}$$

The matrices $\mathbf{S}^{u,m}(t)$ can be computed by backpropagating through the network, from each network output, using

$$\mathbf{S}^{u,m}(t) = \left\{ \sum_{l \in L_m^b} \mathbf{S}^{u,l}(t) \mathbf{LW}^{l,m}(0) \right\} \dot{\mathbf{F}}^m(\mathbf{n}^m(t)), \quad u \in U, \tag{27}$$

where m is decremented from u through the backpropagation order and L_m^b is the set of indices *of* layers that are directly connected backwards to layer m (or to which layer m connects forward) and that contain no delays.

4.2.1 Eq. (3). The next step of the development of the BTT algorithm is the generalization of Eq. (3). Again, we use the chain rule:

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \frac{\partial^e F}{\partial \mathbf{a}^u(t)} + \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u}} \left[\frac{\partial^e \mathbf{a}^{u'}(t+d)}{\partial \mathbf{n}^x(t+d)^T} \times \frac{\partial^e \mathbf{n}^x(t+d)}{\partial \mathbf{a}^u(t)^T} \right]^T \times \frac{\partial F}{\partial \mathbf{a}^{u'}(t+d)} \tag{28}$$

(Many of the terms in these summations will be zero. We will provide a more efficient representation later in this section.) In Eq. (3) we only had one delay in the system. Now we need to account for each network output, how that network output is connected back through a network input, and also for the number of times each network output is delayed before it is applied to a network input. That is the reason for the three summations in Eq. (28). This equation must be updated backward in time, as t is varied from Q to 1. The terms

$$\frac{\partial F}{\partial \mathbf{a}^{u'}(t)} \tag{29}$$

are generally set to zero for $t > Q$.

If we consider one element of the matrix in the brackets on the right side of Eq. (28), we can write

$$\left[\frac{\partial^e \mathbf{a}^{u'}(t+d)}{\partial \mathbf{n}^x(t+d)^T} \times \frac{\partial^e \mathbf{n}^x(t+d)}{\partial \mathbf{a}^u(t)^T} \right]_{i,j} = \sum_{k=1}^{S_x} \frac{\partial^e a_i^{u'}(t+d)}{\partial n_k^x(t+d)} \times \frac{\partial^e n_k^x(t+d)}{\partial a_j^u(t)} \tag{30}$$

The first term on the right hand side of this equation is just our previously defined sensitivity

$$\frac{\partial^e a_i^{u'}(t+d)}{\partial n_k^x(t+d)} = s_{i,k}^{u',x}(t+d), \tag{31}$$

which can be computed from Eq. (27). To find the second term on the right hand side of Eq. (30), we can write

$$n_k^x(t+d) = \sum_{l \in L_x^f} \sum_{d' \in DL_{x,l}} \left(\sum_{i=1}^{S_l} lw_{k,i}^{x,l}(d') a_i^l(t+d-d') \right) + \sum_{l \in L_x^f} \sum_{d' \in DI_{x,l}} \left(\sum_{i=1}^{S_l} iw_{k,i}^{x,l}(d') p_i^l(t+d-d') \right) + b_k^x \quad (32)$$

We can now write

$$\frac{\partial^e n_k^x(t+d)}{\partial a_j^u(t)} = lw_{k,j}^{x,u}(d) \quad (33)$$

Therefore, Eq. (30) can be written

$$\left[\frac{\partial^e \mathbf{a}^{u'}(t+d)}{\partial \mathbf{n}^x(t+d)}^T \times \frac{\partial^e \mathbf{n}^x(t+d)}{\partial \mathbf{a}^u(t)}^T \right]_{i,j} = \sum_{k=1}^{S_x} s_{i,k}^{u',x}(t+d) \times lw_{k,j}^{x,u}(d) \quad (34)$$

or in matrix form

$$\frac{\partial^e \mathbf{a}^{u'}(t+d)}{\partial \mathbf{n}^x(t+d)}^T \times \frac{\partial^e \mathbf{n}^x(t+d)}{\partial \mathbf{a}^u(t)}^T = \mathbf{S}^{u',x}(t+d) \times \mathbf{LW}^{x,u}(d). \quad (35)$$

This allows us to write Eq. (28) as

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \frac{\partial^e F}{\partial \mathbf{a}^u(t)} + \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u}} [\mathbf{S}^{u',x}(t+d) \times \mathbf{LW}^{x,u}(d)]^T \times \frac{\partial F}{\partial \mathbf{a}^{u'}(t+d)} \quad (36)$$

Many of the terms in the summation on the right hand side of Eq. (36) will be zero and will not have to be computed. In order to provide a more efficient implementation of Eq. (36), we define the following sets:

$$E_{LW}^X(u) = \{x \in X \ni \exists(\mathbf{LW}^{x,u} \neq 0)\} \quad (37)$$

$$E_{LW}^U(x) = \{u \in U \ni \exists(\mathbf{LW}^{x,u} \neq 0)\} \quad (38)$$

$$E_S^U(x) = \{u \in U \ni \exists(\mathbf{S}^{u,x} \neq 0)\} \quad (39)$$

$$E_S^X(u) = \{x \in X \ni \exists(\mathbf{S}^{u,x} \neq 0)\} \quad (40)$$

$$E_S(u) = \{x \ni \exists(\mathbf{S}^{u,x} \neq 0)\} \quad (41)$$

We can now rearrange the order of the summation in Eq. (36):

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \frac{\partial^e F}{\partial \mathbf{a}^u(t)} + \sum_{x \in X} \sum_{d \in DL_{x,u}} \sum_{u' \in U} \mathbf{LW}^{x,u}(d)^T \times \mathbf{S}^{u',x}(t+d)^T \times \frac{\partial F}{\partial \mathbf{a}^{u'}(t+d)} \quad (42)$$

and sum only over the existing terms:

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \frac{\partial^e F}{\partial \mathbf{a}^u(t)} + \sum_{x \in E_{LW}^X(u)} \sum_{d \in DL_{x,u}} \mathbf{LW}^{x,u}(d)^T \sum_{u' \in E_S^U(x)} \mathbf{S}^{u',x}(t+d)^T \times \frac{\partial F}{\partial \mathbf{a}^{u'}(t+d)} \quad (43)$$

4.3 Summary

The total BTT algorithm is summarized in Figure 3

5 Summary

This paper has introduced the Layered Digital Dynamic Network (LDDN), which is a general class of dynamic network. A universal dynamic training algorithm for the LDDN was also developed, based on the backpropagation-through-time algorithm.

6 References

[1] De Jesús, O. and Hagan, M., "Forward perturbation for a general class of dynamic network," International Joint Conference on Neural Networks, Washington, DC, July, 2001.

[2] Demuth, H. B. and Beale, M., *Users' Guide for the Neural Network Toolbox for MATLAB, ver. 4.0*, The Mathworks, Natick, MA, 2000.

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

[4] Magnus, J.R., and Neudecker, H., *Matrix Differential Calculus*, John Wiley & Sons, Ltd., Chichester, 1999.

[5] Narendra, K. S. and Parthasrathy, A. M., Identification and control for dynamic systems using neural networks, *IEEE Transactions on Neural Networks*, Vol. 1, 1990, pp. 4-27.

[6] Werbos, P. J., "Backpropagation through time: What it is and how to do it," *Proceedings of the IEEE*, Vol. 78, October 1990, pp. 1550–1560.

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

[8] Yang, W. and Hagan, M.T., "Training recurrent networks," Proceedings of the 7th Oklahoma Symposium on Artificial Intelligence, Stillwater, 1993, pp. 226-233.

Backpropagation-Through-Time

Initialize:

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \mathbf{0}, t > Q, \text{ for all } u \in U,$$

For $t = Q$ to 1,

$U' = \varnothing$, and $E_S(u) = \varnothing$ for all $u \in U$.

For m decremented through the BP order

For all $u \in U'$

$$\mathbf{S}^{u,m}(t) = \left\{ \sum_{l \in E_S(u) \cap L_m^b} \mathbf{S}^{u,l}(t) \mathbf{LW}^{l,m}(0) \right\} \dot{\mathbf{F}}^m(\mathbf{n}^m(t))$$

add m to the set $E_S(u)$

EndFor u

If $m \in U$

$$\mathbf{S}^{m,m}(t) = \dot{\mathbf{F}}^m(\mathbf{n}^m(t))$$

add m to the sets U' and $E_S(m)$

EndIf m

EndFor m

For $u \in U$ decremented through the BP order

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \frac{\partial^e F}{\partial \mathbf{a}^u(t)} + \sum_{x \in E_{LW}^x(u)} \sum_{d \in DL_{x,u}} \mathbf{LW}^{x,u}(d)^T \sum_{u' \in E_S^U(x)} \mathbf{S}^{u',x}(t+d)^T \times \frac{\partial F}{\partial \mathbf{a}^{u'}(t+d)}$$

EndFor u

For all layers m

$$\mathbf{d}^m(t) = \sum_{u \in E_S^U(m)} [\mathbf{S}^{u,m}(t)]^T \times \frac{\partial F}{\partial \mathbf{a}^u(t)}$$

EndFor m

EndFor t

Compute Gradients

$$\frac{\partial F}{\partial \mathbf{LW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^m(t) \times [\mathbf{a}^l(t-d)]^T$$

$$\frac{\partial F}{\partial \mathbf{IW}^{m,l}(d)} = \sum_{t=1}^{Q} \mathbf{d}^m(t) \times [\mathbf{p}^l(t-d)]^T$$

$$\frac{\partial F}{\partial \mathbf{b}^m} = \sum_{t=1}^{Q} \mathbf{d}^m(t)$$

Figure 3: Pseudo Code for the Backpropagation-Through-Time Algorithm