Forward Perturbation Algorithm 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 forward perturbation 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 the BTT algorithm, the network response is computed for all time points, and then the gradient is computed by starting at the last time point and working backwards in time. In the FP algorithm, the gradient can be computed at the same time as the network response, since it is computed by starting at the first time point, and then working forward through time. In this paper we will discuss FP. The FP algorithm has been discussed in a number of papers ([6], [8]), but generally in the context of specific network architectures. In this paper, we develop FP 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 describes the general concept of the forward perturbation algorithm through the use of a simple example. In Section 3 we define LDDN's. In Section 4 we describe a general forward perturbation algorithm for computing training gradients for the LDDN.

A companion paper [1] follows a similar script for the backpropagation through time algorithm.

2 Example of Dynamic Gradient Calculations

To illustrate dynamic backpropagation ([3], [9], [10]), 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.

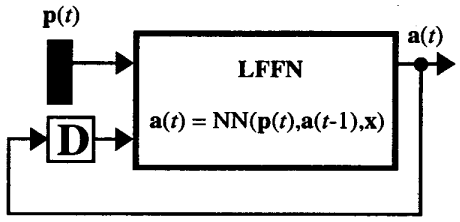

Figure 1: Simple Dynamic Network

Now suppose that we want to minimize

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

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 backpropagation through time method.) In this paper we concentrate on the FP algorithm, which begins with the following equation

0-7803-7044-9/01/$10.00 ©2001 IEEE

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

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 algorithm, as in [4]. To find the complete derivatives that are required in Eq. (2), we need the additional equation:

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

Eq. (2) and Eq. (3) make up the forward perturbation (FP) algorithm. Here the key term is

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

which must be propagated forward through time.

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

3 Definition of the Layered Digital Dynamic Network

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

First, 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 $X=\{1,2\}$.

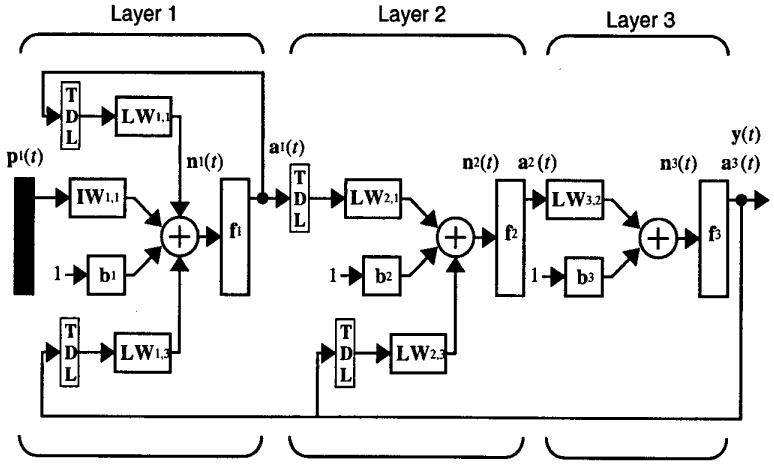

Figure 2: Three-layer LDDN with two output layers (1 and 3) and two input layers (1 and 2)

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 Forward Perturbation

In this section we will generalize the Forward Perturbation (FP) 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

$$\frac{\partial F}{\partial w} = \sum_{t=1}^{Q} \sum_{u \in U} \left\{ \left[\frac{\partial \mathbf{a}^u(t)}{\partial w} \right]^T \times \frac{\partial^e F}{\partial \mathbf{a}^u(t)} \right\}, \tag{7}$$

where w represents $lw_{i,j}^{m,l}(d)$, $iw_{i,j}^{m,l}(d)$ and b_i^m. (The equation is the same for all network parameters.)

4.2 Eq. (3)

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

$$\frac{\partial \mathbf{a}^u(t)}{\partial w} = \frac{\partial^e \mathbf{a}^u(t)}{\partial w} + \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u'}} \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^x(t)^T} \times \frac{\partial^e \mathbf{n}^x(t)}{\partial \mathbf{a}^{u'}(t-d)^T} \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial w} \tag{8}$$

In Eq. (3) we only had one delay in the system. Now we need to account for each output and also for the number of times each output is delayed before it is input to another layer. That is the reason for the summations in Eq. (8). These equations must be updated forward in time, as t is varied from 1 to Q. The terms

$$\frac{\partial \mathbf{a}^u(t)}{\partial w} \tag{9}$$

are generally set to zero for $t \leq 0$.

To implement Eq. (8) we need to compute the terms

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^x(t)^T} \times \frac{\partial^e \mathbf{n}^x(t)}{\partial \mathbf{a}^{u'}(t-d)^T} \tag{10}$$

To find the second term on the right, we can use

$$n_k^x(t) = \sum_{l \in L_x^f} \sum_{d' \in DL_{x,l}} \left(\sum_{i=1}^{S_l} lw_{k,i}^{x,l}(d') a_i^l(t-d') \right) + \sum_{l \in L_x^f} \sum_{d' \in DI_{x,l}} \left(\sum_{i=1}^{S_l} iw_{k,i}^{x,l}(d') p_i^l(t-d') \right) + b_k^x \tag{11}$$

we can now write

$$\frac{\partial^e n_k^x(t)}{\partial a_j^{u'}(t-d)} = lw_{k,j}^{x,u'}(d) \tag{12}$$

If we define the following sensitivity term

$$s_{k,i}^{u,m}(t) \equiv \frac{\partial^e a_k^u(t)}{\partial n_i^m(t)}, \tag{13}$$

which can be used to make up the following matrix

$$\mathbf{S}^{u,m}(t) = \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^m(t)^T} = \begin{bmatrix} s_{1,1}^{u,m}(t) & s_{1,2}^{u,m}(t) & \cdots & s_{1,S_m}^{u,m}(t) \\ s_{2,1}^{u,m}(t) & s_{2,2}^{u,m}(t) & \cdots & s_{1,S_m}^{u,m}(t) \\ & & & \\ s_{S_u,1}^{u,m}(t) & s_{S_u,2}^{u,m}(t) & \cdots & s_{S_u,S_m}^{u,m}(t) \end{bmatrix} = \left[\mathbf{s}_1^{u,m}(t)\ \mathbf{s}_2^{u,m}(t)\ \cdots\ \mathbf{s}_{S_m}^{u,m}(t) \right] = \begin{bmatrix} {}_1\mathbf{s}^{u,m}(t)^T \\ {}_2\mathbf{s}^{u,m}(t)^T \\ \\ {}_{S_u}\mathbf{s}^{u,m}(t)^T \end{bmatrix} \tag{14}$$

then we can write Eq. (10) as

$$\left[\frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^x(t)}^T \times \frac{\partial^e \mathbf{n}^x(t)}{\partial \mathbf{a}^{u'}(t-d)}^T\right]_{i,j} = \sum_{k=1}^{S_x} s_{i,k}^{u,x}(t+d) \times lw_{k,j}^{x,u'}(d) \quad (15)$$

or in matrix form

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^x(t)}^T \times \frac{\partial^e \mathbf{n}^x(t)}{\partial \mathbf{a}^{u'}(t-d)}^T = \mathbf{S}^{u,x}(t) \times \mathbf{LW}^{x,u'}(d). \quad (16)$$

and therefore Eq. (8) can be written

$$\frac{\partial \mathbf{a}^u(t)}{\partial w} = \frac{\partial^e \mathbf{a}^u(t)}{\partial w} + \sum_{u' \in U} \sum_{x \in X} \sum_{d \in DL_{x,u'}} \mathbf{S}^{u,x}(t) \times \mathbf{LW}^{x,u'}(d) \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial w} \quad (17)$$

Many of the terms in the summation on the right hand side of Eq. (17) will be zero and will not have to be computed. To take advantage of these efficiencies, we introduce the following definitions.

$$E_{LW}^U(x) = \{u \in U \ni \exists(\mathbf{LW}^{x,u} \neq 0)\} \quad (18)$$

$$E_S^X(u) = \{x \in X \ni \exists(\mathbf{S}^{u,x} \neq 0)\} \quad (19)$$

$$E_S(u) = \{x \ni \exists(\mathbf{S}^{u,x} \neq 0)\} \quad (20)$$

Using Eq. (18) and Eq. (19), we can rearrange the order of the summations in Eq. (17) and sum only over existing terms

$$\frac{\partial \mathbf{a}^u(t)}{\partial w} = \frac{\partial^e \mathbf{a}^u(t)}{\partial w} + \sum_{x \in E_S^X(u)} \mathbf{S}^{u,x}(t) \sum_{u' \in E_{LW}^U(x)} \sum_{d \in DL_{x,u'}} \mathbf{LW}^{x,u'}(d) \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial w} \quad (21)$$

This can be written for the individual weight matrices

$$\frac{\partial \mathbf{a}^u(t)}{\partial vec(\mathbf{LW}^{m,l}(d))}^T = \frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{LW}^{m,l}(d))}^T + \sum_{x \in E_S^X(u)} \mathbf{S}^{u,x}(t) \sum_{u' \in E_{LW}^U(x)} \sum_{d \in DL_{x,u'}} \mathbf{LW}^{x,u'}(d) \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial vec(\mathbf{LW}^{m,l}(d))}^T \quad (22)$$

$$\frac{\partial \mathbf{a}^u(t)}{\partial vec(\mathbf{IW}^{m,l}(d))}^T = \frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{IW}^{m,l}(d))}^T + \sum_{x \in E_S^X(u)} \mathbf{S}^{u,x}(t) \sum_{u' \in E_{LW}^U(x)} \sum_{d \in DL_{x,u'}} \mathbf{LW}^{x,u'}(d) \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial vec(\mathbf{IW}^{m,l}(d))}^T \quad (23)$$

$$\frac{\partial \mathbf{a}^u(t)}{\partial (\mathbf{b}^m)}^T = \frac{\partial^e \mathbf{a}^u(t)}{\partial (\mathbf{b}^m)}^T + \sum_{x \in E_S^X(u)} \mathbf{S}^{u,x}(t) \sum_{u' \in E_{LW}^U(x)} \sum_{d \in DL_{x,u'}} \mathbf{LW}^{x,u'}(d) \times \frac{\partial \mathbf{a}^{u'}(t-d)}{\partial (\mathbf{b}^m)}^T \quad (24)$$

where the *vec* operator transforms a matrix into a vector by stacking the columns of the matrix one underneath the other [5].

Eq. (21) through Eq. (24) make up the generalization of Eq. (3) for the LDDN network. It remains to compute the sensitivity matrices $\mathbf{S}^{u,m}(t)$ and the explicit derivatives $\partial^e \mathbf{a}^u(t)/\partial w$, which are described in the next two subsections.

4.3 Sensitivities

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

$$s_{k,i}^{u,u}(t) = \frac{\partial^e a_k^u(t)}{\partial n_i^u(t)} = \begin{cases} \dot{f}^u(n_i^u(t)) & \text{for } i = k \\ 0 & \text{for } i \neq k \end{cases}, \quad u \in U, \quad (25)$$

or, in matrix form,

$$\mathbf{S}^{u,u}(t) = \dot{\mathbf{F}}^u(\mathbf{n}^u(t)), \quad (26)$$

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

$$\dot{\mathbf{F}}^u(\mathbf{n}^u(t)) = \begin{bmatrix} \dot{f}^u(n_1^u(t)) & 0 & \dots & 0 \\ 0 & \dot{f}^u(n_2^u(t)) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \dot{f}^u(n_{S^u}^u(t)) \end{bmatrix} \quad (27)$$

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

$$\mathbf{S}^{u,m}(t) = \left\{ \sum_{l \in L_m^b} \mathbf{S}^{u,l}(t)\mathbf{LW}^{l,m}(0) \right\} \dot{\mathbf{F}}^m(\mathbf{n}^m(t)), \; u \in U, \quad (28)$$

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.

4.4 Explicit Derivatives

We also need to compute the explicit derivatives

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial w}. \quad (29)$$

We can derive the following three expansions of Eq. (29):

$$\frac{\partial^e a_k^u(t)}{\partial iw_{i,j}^{m,l}(d)} = \frac{\partial^e a_k^u(t)}{\partial n_i^m(t)} \times \frac{\partial^e n_i^m(t)}{\partial iw_{i,j}^{m,l}(d)} = s_{i,k}^{u,m}(t) \times p_j^l(t-d), \quad (30)$$

$$\frac{\partial^e a_k^u(t)}{\partial lw_{i,j}^{m,l}(d)} = \frac{\partial^e a_k^u(t)}{\partial n_i^m(t)} \times \frac{\partial^e n_i^m(t)}{\partial lw_{i,j}^{m,l}(d)} = s_{i,k}^{u,m}(t) \times a_j^l(t-d), \quad (31)$$

$$\frac{\partial^e a_k^u(t)}{\partial b_i^m} = \frac{\partial^e a_k^u(t)}{\partial n_i^m(t)} \times \frac{\partial^e n_i^m(t)}{\partial b_i^m} = s_{i,k}^{u,m}(t). \quad (32)$$

In vector form we can write

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial iw_{i,j}^{m,l}(d)} = \mathbf{s}_i^{u,m}(t) \times p_j^l(t-d), \quad (33)$$

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial lw_{i,j}^{m,l}(d)} = \mathbf{s}_i^{u,m}(t) \times a_j^l(t-d), \quad (34)$$

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial b_i^m} = \mathbf{s}_i^{u,m}(t). \quad (35)$$

In matrix form we have

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{IW}^{m,l}(d))^T} = [\mathbf{p}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t), \quad (36)$$

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{LW}^{m,l}(d))^T} = [\mathbf{a}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t), \quad (37)$$

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial (\mathbf{b}^m)^T} = \mathbf{S}^{u,m}(t), \quad (38)$$

where $\mathbf{A} \otimes \mathbf{B}$ is the Kronecker product of $\mathbf{A}$ and $\mathbf{B}$ [5].

4.5 Summary

The total FP algorithm for the LDDN network 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 forward perturbation algorithm.

6 References

[1] De Jesús, O. and Hagan, M., "Backpropagation through time 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., De Jesús, O., and Schultz, R., "Training Recurrent Networks for Filtering and Control," in *Recurrent Neural Networks: Design and Applications*, L.R. Medsker and L.C. Jain, Eds., CRC Press, 2000, pp. 325-354.

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

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

[6] 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.

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

[8] Williams, R. J. and Zipser, D., "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, pp. 270–280, 1989.

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

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

Initialize:

Forward Perturbation

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

For $t = 1$ to Q,

$U' = \varnothing$, $E_S(u) = \varnothing$ and $E_S^X(u) = \varnothing$ 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 $E_S(u)$

if $m \in X$, add m to the set $E_S^X(u)$

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 $E_S(m)$

if $m \in X$, add m to the set $E_S^X(m)$

EndIf m

EndFor m

For $u \in U$ incremented through the simulation order

For all weights and biases ($\mathbf{w}$ is a vector containing all weights and biases)

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{IW}^{m,l}(d))^T} = [\mathbf{p}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t)$$

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial vec(\mathbf{LW}^{m,l}(d))^T} = [\mathbf{a}^l(t-d)]^T \otimes \mathbf{S}^{u,m}(t)$$

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial (\mathbf{b}^m)^T} = \mathbf{S}^{u,m}(t)$$

EndFor weights and biases

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

EndFor u

EndFor t

Compute Gradients

$$\frac{\partial F}{\partial \mathbf{w}^T} = \sum_{t=1}^{Q} \sum_{u \in U} \left\{ \left[\frac{\partial^e F}{\partial \mathbf{a}^u(t)} \right]^T \times \frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{w}^T} \right\}$$

Figure 3: Pseudo Code for the Forward Perturbation Method