Νευρο-Ασαφής Υπολογιστική Neuro-Fuzzy Computing

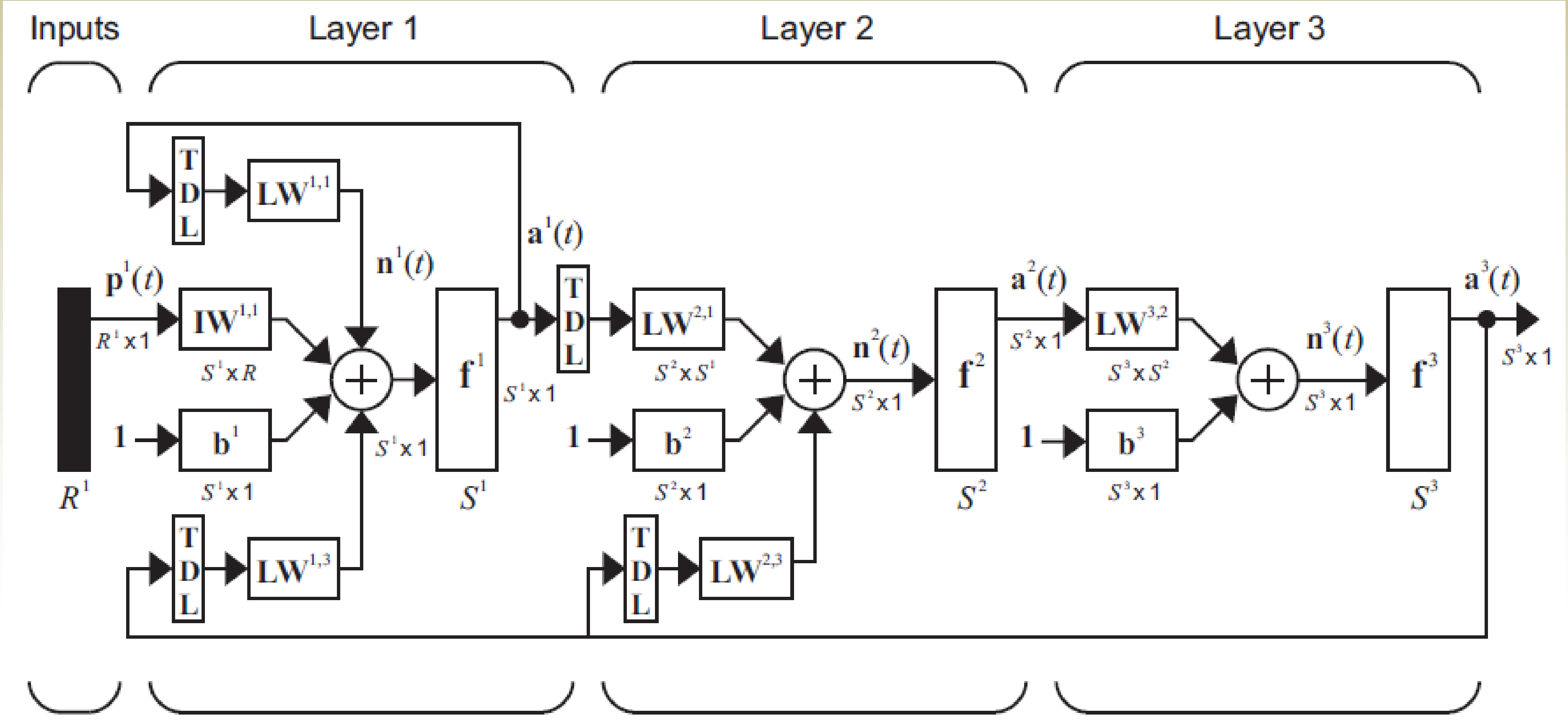
Διδάσκων –
Δημήτριος Κατσαρός

@ Τμ. ΗΜΜΥ
Πανεπιστήμιο Θεσσαλίας

Διάλεξη 22η

Recall LDDNs

- We introduced a framework for representing general dynamic networks, and called it **Layered Digital Dynamic Networks (LDDN)**



Recall LDDNs

- The general equations for the computation of the net input $\mathbf{n}^m(t)$ for layer m of an LDDN are:

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

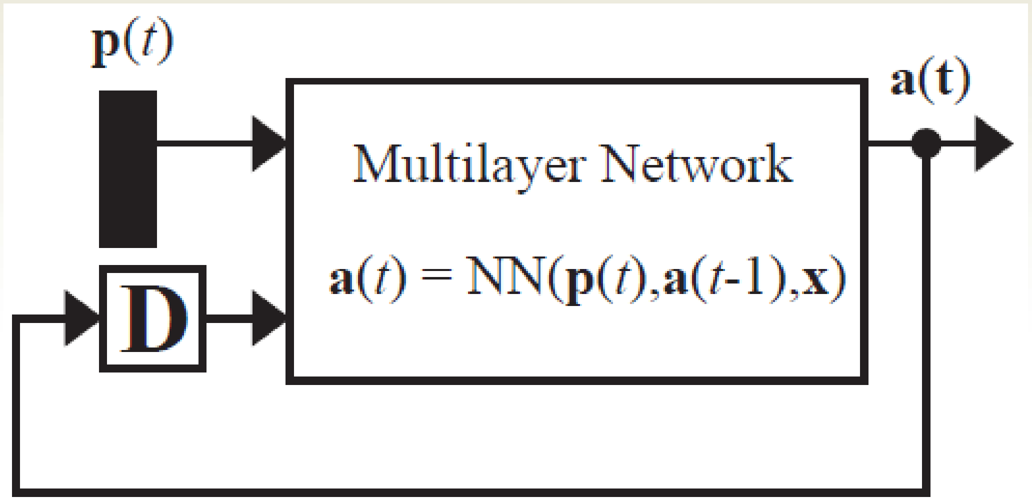
where $\mathbf{p}^l(t)$ is the l-th input vector at time t, $\mathbf{IW}^{m,l}$ is the *input weight* between input l and layer m, $\mathbf{LW}^{m,l}$ is the *layer weight* between layer l and layer m, $\mathbf{b}^m$ is the bias vector for layer m, $DL_{m,l}$ is the set of all delays in the tapped delay 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 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:

$$\mathbf{a}^m(\mathrm{t}) = \mathbf{f}^m(\mathbf{n}^m(t))$$

Principles of Dynamic Learning

- Having made this initial investigation of RNNs in the previous lecture, let's consider the slightly more complex dynamic network (than a two-layer RNN):
  - It consists of a static multilayer network with a single feedback loop added from the output of the network to the input of the network through a single delay
  - The vector **x** represents all of the network parameters (weights and biases), and the vector **a**(*t*) represents the output of the multilayer network at time step *t*



Principles of Dynamic Learning

- As with a standard multilayer network, we want to adjust the weights and biases of the network to minimize the performance index, F(**x**), which is normally chosen to be the mean squared error
- With dynamic networks, we need to modify the standard backpropagation algorithm
- There are two different approaches to this problem. They both use the chain rule, but are implemented in different ways:

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

Equation A1

or

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

Equation B1

where the superscript ^{e} indicates an explicit derivative, not accounting for indirect effects through time

Principles of Dynamic Learning

- The explicit derivatives can be obtained with the standard backpropagation algorithm
- To find the complete derivatives that are required in the previous slide's equations, we need the additional equations:

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

Equation A2

and

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

Equation B2

Principles of Dynamic Learning

- Equations A1 and A2 make up the real-time recurrent learning (RTRL) algorithm
  - Note that the key term is:

$$\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}^T}$$

  which must be propagated <u>forward</u> through time

- Equations B1 and B2 make up the backpropagation-through-time (BPTT) algorithm
  - Note that the key term is:

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

  which must be propagated <u>backward</u> through time

Principles of Dynamic Learning

- In general, the RTRL algorithm requires somewhat more computation than the BPTT algorithm to compute the gradient
- However, the BPTT algorithm cannot be conveniently 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 RTRL algorithm is well suited for real time implementation, since the derivatives can be calculated at each time step

Dynamic Backpropagation

- Now we will develop general RTRL and BPTT algorithms for dynamic networks represented in the LDDN framework: this development will involve generalizing Equations A1 & A2, and B1 & B2
- 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 that has any delays associated with it

Dynamic Backpropagation

- For example, the LDDN shown in Slide-2's figure 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 mentioned, $U=\{1,3\}$ and $X=\{1,2\}$
- The general equations for simulating an arbitrary LDDN network are given in the Slide-3 equations. At each time point, these equations are iterated forward through the layers, as m is incremented through the simulation order. Time is then incremented from $t=1$ to Q

Real-Time Recurrent Learning for LDDNs

Real Time Recurrent Learning: Eq. A1

- We will generalize the RTRL algorithm, given in Equations A1 and A2, for LDDN networks. This development will follow in many respects the development of the backpropagation algorithm for static multilayer networks
- The first step in developing the RTRL algorithm is to generalize Equation A1. For the general LDDN network, we can calculate the terms of the gradient by using the chain rule, as in:

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

- If we compare this equation with Equation A1, we notice that in addition to each time step, we also have a term in the sum for each output layer. However, if the performance index F($\mathbf{x}$) is not explicitly a function of a specific output $\mathbf{a}^u(t)$, then that explicit derivative will be zero

Real Time Recurrent Learning: Eq. A2

- The next step of the development of the RTRL algorithm is the generalization of Equation A2. Again, we use the chain rule:

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

- In Equation A2 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 first two summations in the above equation. These equations must be updated forward in time, as t is varied from 1 to Q.

- The terms $\frac{\partial \mathbf{a}^u(t)}{\partial \mathbf{x}^T}$ are generally set to zero for $t \leq 0$

Real Time Recurrent Learning: Eq. A2

- To implement the previous slide's equation, we need to compute the terms:

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

Equation C

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

$$n_k^x(t) = \sum_{l \in L_x^f} \sum_{d' \in DL_{x,l}} \Big[\sum_{i=1}^{S^l} lw_{k,i}^{x,l}(d') a_i^l(t-d') +$$

$$+ \sum_{l \in I_x} \sum_{d' \in DI_{x,l}} \Big[\sum_{i=1}^{R^l} iw_{k,i}^{x,l}(d') p_i^l(t-d') \Big] + b_k^x$$

Real Time Recurrent Learning: Eq. A2

- We can now write:

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

- If we define the following sensitivity term:

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

which can be used to make up the following matrix:

$$\mathbf{S}^{u,m}(t) = \frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{n}^m(t)^T} = \begin{bmatrix} S_{1,1}^{u,m}(t) & S_{1,2}^{u,m}(t) & \dots & S_{1,S_m}^{u,m}(t) \\ S_{2,1}^{u,m}(t) & S_{2,2}^{u,m}(t) & \dots & S_{2,S_m}^{u,m}(t) \\ \vdots & \vdots & & \vdots \\ S_{S_u,1}^{u,m}(t) & S_{S_u,2}^{u,m}(t) & \dots & S_{S_u,S_m}^{u,m}(t) \end{bmatrix}$$

Real Time Recurrent Learning: Eq. A2

- Then we can write Equation C as follows:

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

- or in matrix form

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

- Therefore, Slide-13's equation can be written:

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

Real Time Recurrent Learning: Eq. A2

- Many of the terms in the summation on the right hand side of the previous slide’s equation will be zero and will not have to be computed
- To take advantage of these efficiencies, we introduce some indicator sets. They are sets that tell us for which layers the weights and the sensitivities are nonzero

Real Time Recurrent Learning: Eq. A2

- The first type of indicator set contains all of the output layers that connect to a specified layer x (which will always be an input layer) with at least some nonzero delay:

$$E_{LW}^{U}(x) = \{u \in U \text{ such that } \exists(\mathbf{LW}^{x,u}(d) \neq 0, d \neq 0)\}$$

- The second type of indicator set contains the input layers that have a nonzero sensitivity with a specified layer *u*:

$$E_{S}^{X}(u) = \{x \in X \text{ such that } \exists(\mathbf{S}^{u,x} \neq 0)\}$$

  - When $\mathbf{S}^{u,x}$ is nonzero, there is a static connection from layer *x* to output layer *u*

- The third type of indicator set contains the layers that have a nonzero sensitivity with a specified layer u:

$$E_{S}(u) = \{x \text{ such that } \exists(\mathbf{S}^{u,x} \neq 0)\}$$

- The difference between $E^{X}{}_{S}(u)$ and $E_{S}(u)$ is that $E^{X}{}_{S}(u)$ contains only input layers. $E_{S}(u)$ will not be needed in the simplification of the 1st equation of Slide-16, but it will be used for the calculation of sensitivities later

Real Time Recurrent Learning: Eq. A2

- Using the first two equations in the previous slide, we can rearrange the order of the summations in the last equation of Slide16 and sum only over nonzero terms:

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

- The above equation makes up the generalization of Equation A2 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 next

Real Time Recurrent Learning: Sensitivities

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

$$s_{k,i}^{u,u}(t) = \frac{\partial^e a_k^u(t)}{\partial n_i^u(t)} = \begin{cases} f'^{u}(n_i^u(t))) & \text{for } i = k, u \in U \\ 0 & \text{for } i \neq k \end{cases}$$

- or, in matrix form:

$$\mathbf{S}^{u,m}(t) = \mathbf{F'}^{\mathrm{u}}(\mathbf{n}^{\mathrm{u}}(t))$$

Real Time Recurrent Learning: Sensitivities

where $\mathbf{F}'^{u}(\mathbf{n}^{u}(t))$ is defined as follows:

$$\mathbf{F}'^{\mathbf{u}}(\mathbf{n}^{u})(t) = \begin{bmatrix} f'^{u}(n_1^u(t)) & 0 & \dots & 0 \\ 0 & f'^{u}(n_2^u(t)) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & f'^{u}(n_{S^u}^u(t)) \end{bmatrix}$$

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

$$\mathbf{S}^{u,m}(t) = \left[\sum_{l \in E_S(u) \cap L_m^b} \mathbf{S}^{u,l}(t)\mathbf{LW}^{l,m}(0) \right] \mathbf{F}'^{m}(\mathbf{n}^{m}(t)), \; u \in U$$

where m is decremented from u through the backpropagation order, and L^b_m 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 in the connection

Real Time Recurrent Learning: Explicit Derivatives

- We also need to compute the explicit derivatives:

$$\frac{\partial^e \mathbf{a}^u(t)}{\partial \mathbf{x}^T}$$

- Using the chain rule of calculus, we can derive the following expansion of the previous equation for input weights:

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

- In vector form, we can write:

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

Real Time Recurrent Learning: Explicit Derivatives

- In matrix form, we have:

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

- and in a similar way we can derive the derivatives for layer weights and biases:

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

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

where the *vec* operator transforms a matrix into a vector by stacking the columns of the matrix one underneath the other, and **A** ⊗ **B** is the Kronecker product of **A** and **B**

Backpropagation Through Time for LDDNs

Recall: Principles of Dynamic Learning

- As with a standard multilayer network, we want to adjust the weights and biases of the network to minimize the performance index, F(**x**), which is normally chosen to be the mean squared error
- With dynamic networks, we need to modify the standard backpropagation algorithm
- There are two different approaches to this problem. They both use the chain rule, but are implemented in different ways:

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

Equation A1

or

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

Equation B1

where the superscript ^{e} indicates an explicit derivative, not accounting for indirect effects through time

Recall: Principles of Dynamic Learning

- The explicit derivatives can be obtained with the standard backpropagation algorithm
- To find the complete derivatives that are required in the previous slide's equations, we need the additional equations:

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

Equation A2

and

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

Equation B2

Recall: Principles of Dynamic Learning

- Equations A1 and A2 make up the real-time recurrent learning (RTRL) algorithm
  - Note that the key term is:

$$\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}^T}$$

  which must be propagated forward through time

- Equations B1 and B2 make up the backpropagation-through-time (BPTT) algorithm
  - Note that the key term is:

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

  which must be propagated backward through time

Backpropagation-Through-Time: Eq.B1

- Now, we will generalize the Backpropagation-Through-Time (BPTT) algorithm, given in Eq. B1 and B2, for LDDN networks
- The first step is to generalize Eq. B1. For the general LDDN network, we can calculate the terms of the gradient by using the chain rule, as in:

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

(for the layer weights), 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*

Backpropagation-Through-Time: Eq.B1

- From equation in Slide14

$$n_k^x(t) = \sum_{l \in L_x^f} \sum_{d' \in DL_{x,l}} \Big[\sum_{i=1}^{S^l} lw_{k,i}^{x,l}(d')a_i^l(t-d') + \sum_{l \in I_x} \sum_{d' \in DI_{x,l}} \Big[\sum_{i=1}^{R^l} iw_{k,i}^{x,l}(d')p_i^l(t-d') \Big] + b_k^x$$

- we can write: $\dfrac{\partial^e n_i^m(t)}{\partial lw_{i,j}^{m,l}(d)} = a_j^l(t-d)$
- We will also define: $d_i^m(t) = \displaystyle\sum_{u \in U} \sum_{k=1}^{S^u} \frac{\partial F}{\partial a_k^u(t)} \times \frac{\partial^e a_k^u(t)}{\partial n_i^m(t)}$
- The terms of the gradient for the layer weights can then be written:

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

- If we use the sensitivity term defined in previous lecture:

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

Backpropagation-Through-Time: Eq.B1

- then the elements $d^m{}_i(t)$ can be written:

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

- In matrix form this becomes:

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

where

$$\frac{\partial F}{\partial \mathbf{a}^u(t)} = \left[\frac{\partial F}{\partial a_1^u(t)} \frac{\partial F}{\partial a_2^u(t)} \cdots \frac{\partial F}{\partial a_{S^u}^u(t)} \right]^T$$

Backpropagation-Through-Time: Eq.B1

- Now the gradient can be written in matrix form:

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

and by similar steps we can find the derivatives for the biases and input weights:

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

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

- This Slide's equations make up the generalization of Equation B1 for the LDDN network.

Backpropagation-Through-Time: Eq.B2

- The next step in the development of the BPTT algorithm is the generalization of Equation B2. Again, we use the chain rule:

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

- (Many of the terms in these summations will be zero. We will provide a more efficient representation later in this lecture.)
- In Equation B2 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 this slide's equation. This equation must be updated backward in time, as t is varied from Q to 1

Backpropagation-Through-Time: Eq.B2

- The terms $\frac{\partial F}{\partial \mathbf{a}^{u'}(t)}$ are generally set to zero for t > Q
- If we consider the matrix in the brackets on the right side of last slide's equation, from equation in Slide-16

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

- we can write:

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

- This allows us to write last slide's equation as follows:

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

- Many of the terms in the summation on the right hand side of the above equation will be zero and will not have to be computed

Backpropagation-Through-Time: Eq.B2

- In order to provide a more efficient implementation of previous slide’s last equation, we define the following indicator sets:

$$E_{LW}^{X}(u) = \{x \in X \text{ such that } \exists(\mathbf{LW}^{x,u}(d) \neq 0, d \neq 0)\}$$

$$E_{S}^{U}(x) = \{u \in U \text{ such that } \exists(\mathbf{S}^{u,x} \neq 0)\}$$

- The first set contains all of the input layers that have a connection from output layer u with at least some nonzero delay
- The second set contains output layers that have a nonzero sensitivity with input layer x. When the sensitivity $S^{u,x}$ is nonzero, there is a static connection from input layer x to output layer u

Backpropagation-Through-Time: Eq.B2

- We can now rearrange the order of the summation in Slide-33's equation and sum only over the existing terms:

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