Νευρο-Ασαφής Υπολογιστική
Neuro-Fuzzy Computing

Διδάσκων –
Δημήτριος Κατσαρός

@ Τμ. ΗΜΜΥ
Πανεπιστήμιο Θεσσαλίας

Διάλεξη 21η

Dynamic networks

Introduction

- Neural networks can be classified into static and dynamic
- The multilayer network (MLP) that we have discussed is a static network
  - This means that the output can be 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
  - For example, the adaptive filter networks (ADALINE with delay elements) we discussed are dynamic networks
    - since the output is computed from a tapped delay line of previous inputs
  - The Hamming network we discussed is also a dynamic network
    - It has recurrent (feedback) connections, which means that the current output is a function of outputs at previous times

Introduction

- Dynamic networks are networks that contain delays and that operate on a sequence of inputs
  - (In other words, the ordering of the inputs is important to the operation of the network.)
- These dynamic networks can have purely feedforward connections, or they can also have some feedback (recurrent) connections
- Dynamic networks have memory. Their response at any given time will depend not only on the current input, but on the history of the input sequence

Introduction

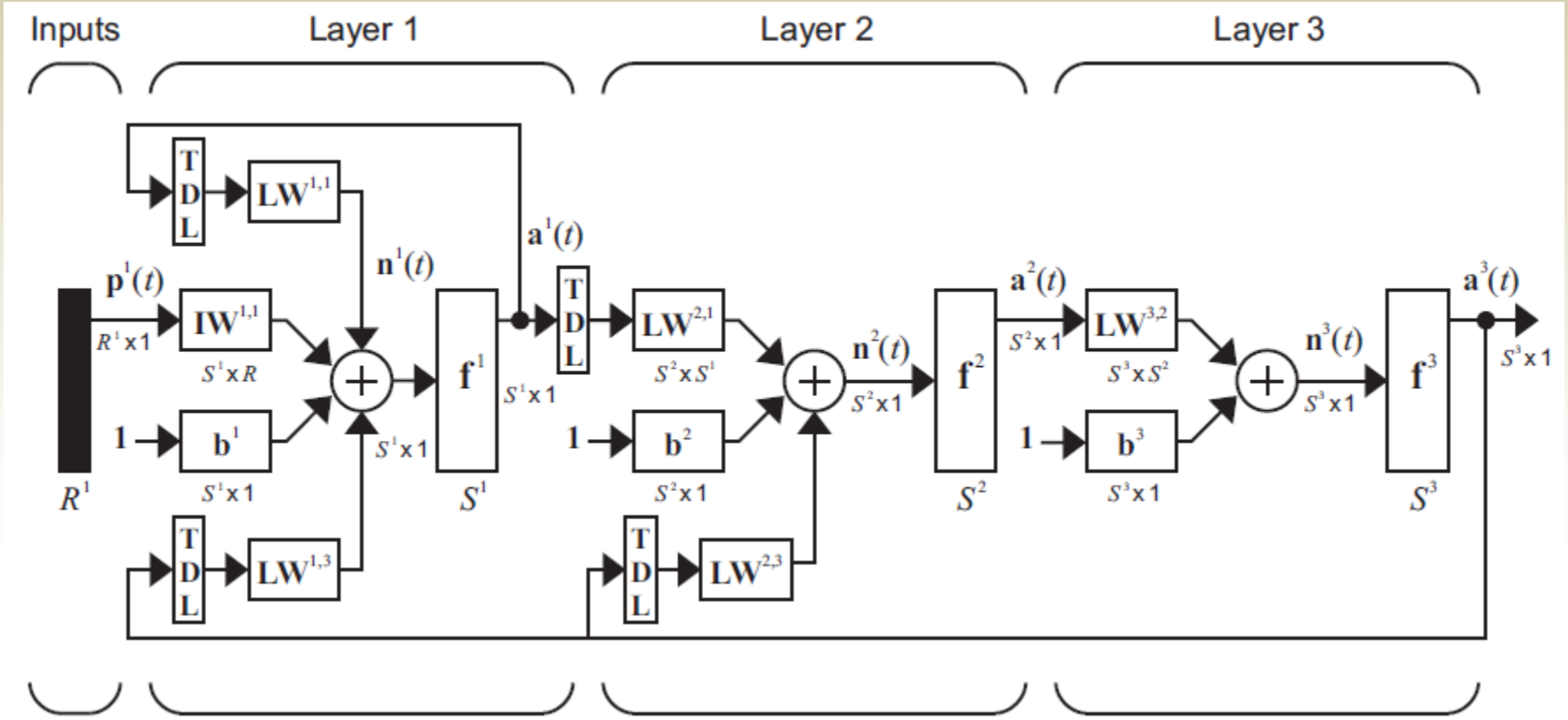
- Because dynamic networks have memory, they can be trained to learn sequential or time-varying patterns.
  - Instead of approximating functions, like the static multilayer perceptron network, a dynamic network can approximate a dynamic system
    - This has applications in such diverse areas as control of dynamic systems, prediction in financial markets, channel equalization in communication systems, phase detection in power systems, sorting, fault detection, speech recognition, learning of grammars in natural languages, and even the prediction of protein structure in genetics
- Dynamic networks can be trained using the standard optimization methods
  - However, the gradients and Jacobians that are required for these methods cannot be computed using the standard backpropagation
  - We will present the dynamic backpropagation algorithms that are required for computing the gradients for dynamic networks

Introduction

- There are two general approaches (with many variations) to gradient and Jacobian calculations in dynamic networks:
  - backpropagation-through time (BPTT), and
  - real-time recurrent learning (RTRL)
- In the BPTT 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 backward in time
  - This algorithm is efficient for the gradient calculation, but it is difficult to implement on-line, because the algorithm works backward in time from the last time step
- In the RTRL 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
  - RTRL requires more calculations than BPTT for calculating the gradient, but RTRL allows a convenient framework for on-line implementation. For Jacobian calculations, the RTRL algorithm is generally more efficient than the BPTT algorithm

Layered Digital Dynamic Networks (LDDN)

- We will introduce a framework for representing general dynamic networks: we call it **Layered Digital Dynamic Networks (LDDN)**
  - With this new notation, we can represent networks with multiple recurrent (feedback) connections and tapped delay lines
- Consider the example dynamic network given in figure below



Layered Digital Dynamic Networks (LDDN)

- 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:

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

Layered Digital Dynamic Networks (LDDN)

- LDDN networks can have several layers connecting to layer m
- Some of the connections can be recurrent through tapped delay lines
- An LDDN can also have multiple input vectors, and the input vectors can be connected to any layer in the network
  - for static multilayer networks, we assumed that the single input vector connected only to Layer 1
- With static multilayer networks, the layers were connected to each other in numerical order
  - In other words, Layer 1 was connected to Layer 2, which was connected to Layer 3, etc.
- Within the LDDN framework, any layer can connect to any other layer, even to itself

Layered Digital Dynamic Networks (LDDN)

- However, in order to use he equation that calculates the net input, we need to compute the layer outputs in a specific order
- The order in which the layer outputs must be computed to obtain the correct network output is called the *simulation order*
  - (This order need not be unique; there may be several valid simulation orders.)
- In order to backpropagate the derivatives for the gradient calculations, we must proceed in the opposite order, which is called the *backpropagation order*

Layered Digital Dynamic Networks (LDDN)

- As with the multilayer network, the fundamental unit of the LDDN is the layer. Each layer in the LDDN is made up of five components:

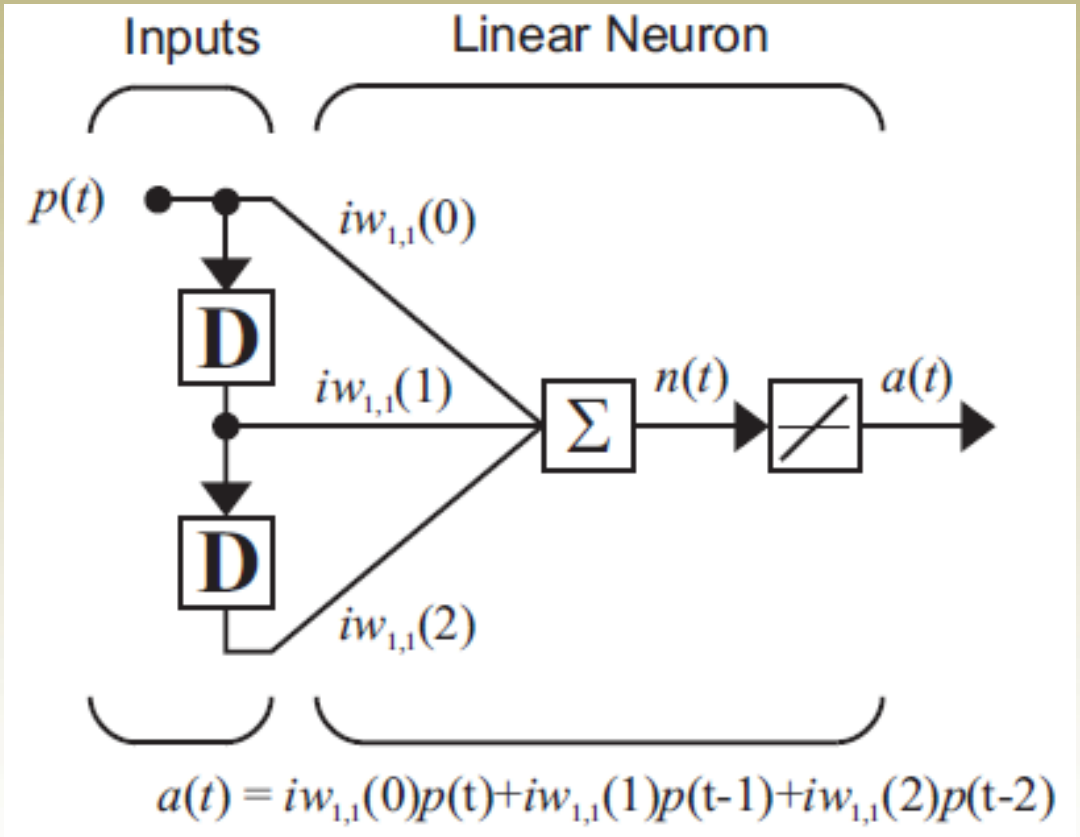
1. a set of weight matrices that come into that layer
   - which may connect from other layers or from external inputs
2. any tapped delay lines (represented by $DL_{m,l}$ or $DI_{m,l}$) that appear at the input of a set of weight matrices
   - Any set of weight matrices can be preceded by a TDL
     - For example, Layer 1 of the figure contains the weights $\mathbf{LW}^{1,3}$(d) and the corresponding TDL
3. a bias vector
4. a summing function
5. a transfer function

Layered Digital Dynamic Networks (LDDN)

- The output of the LDDN is a function not only of the weights, biases, and current network inputs, but also of some layer outputs 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
- 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
  - The second is an indirect effect, since some of the inputs to the network are previous outputs, which are also functions of the weights and biases

Dynamic networks: Example-1

- Consider the feedforward dynamic network shown below



- This is an ADALINE filter and we are representing it in the LDDN framework

Dynamic networks: Example-1

- The network has a TDL on the input, with $DI_{1,1}=\{0,1,2\}$
- To demonstrate the operation of this network, we will apply a square wave as input, and we will set all of the weight values equal to 1/3:

$$iw_{1,1}(0)=1/3, \quad iw_{1,1}(1)=1/3, \quad iw_{1,1}(2)=1/3$$

- The network response is calculated from:

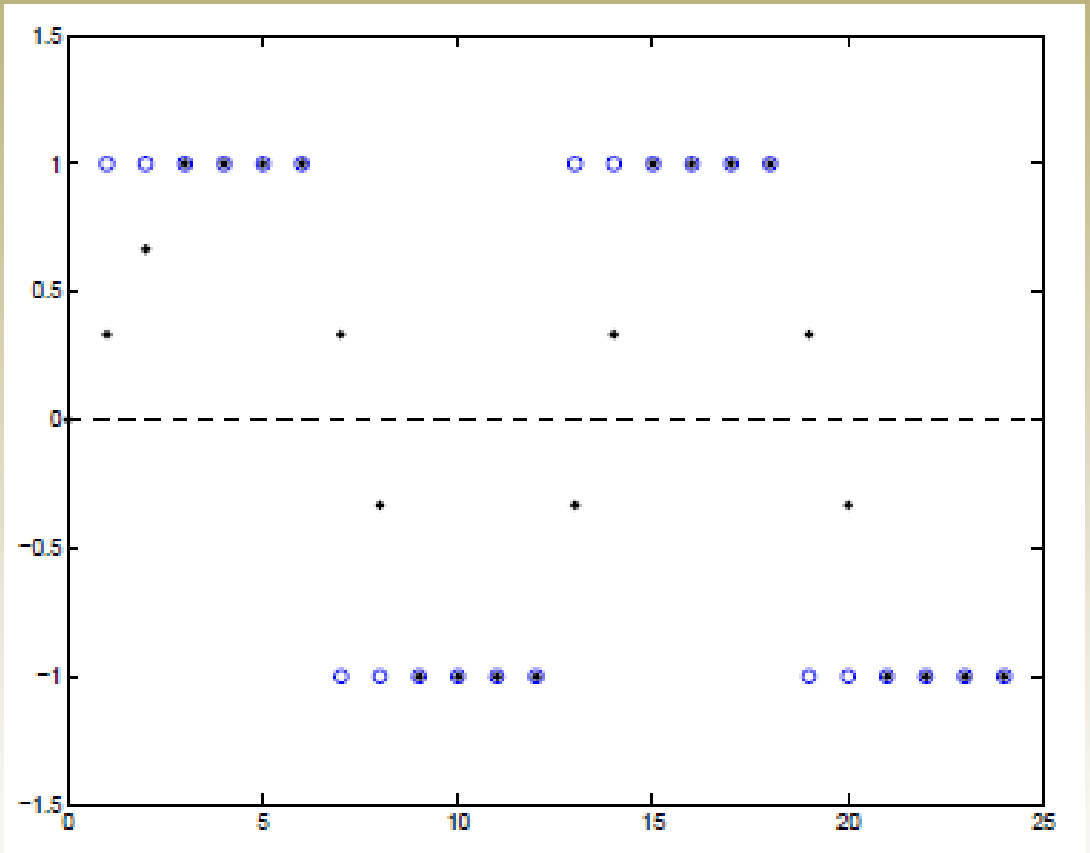
$$\mathbf{a}(t) = \mathbf{n}(t) = \sum_{d=0}^{2} \mathbf{IW}(d)\mathbf{p}(t-d)$$

$$= n_1(t) = iw_{1,1}(0)p(t) + iw_{1,1}(1)p(t\text{-}1) + iw_{1,1}(2)p(t\text{-}2)$$

where we have left off the superscripts on the weight and the input, since there is only one input and only one layer

Dynamic networks: Example-1

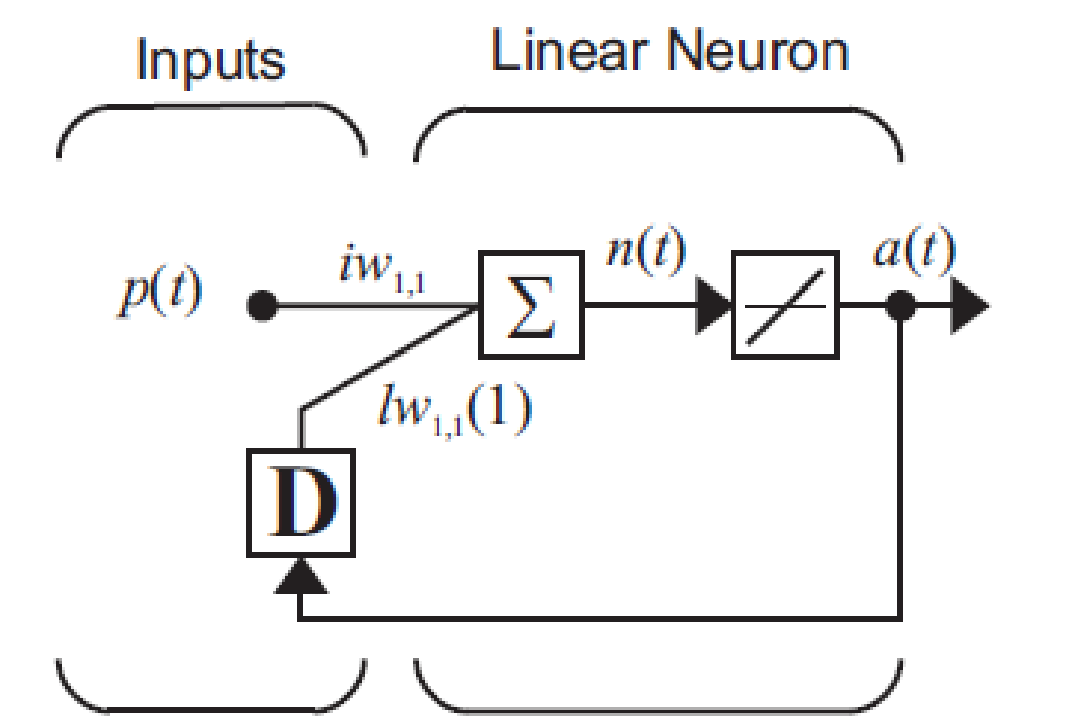
- The response of the network is shown in figure
  - The open circles represent the square-wave input signal $p(t)$. The dots represent the network response $a(t)$
- For this dynamic network, the response at any time point depends on the previous three input values
  - If the input is constant, the output will become constant after 3 time steps
- This type of linear network is called a Finite Impulse Response (FIR) filter



This dynamic network has memory. Its response at any given time will depend not only on the current input, but on the history of the input sequence. If the network does not have any feedback connections, then only a finite amount of history will affect the response

Dynamic networks: Example-2

- Now consider another simple linear dynamic network, but one that has a recurrent connection. The network in figure is a recurrent dynamic network.



Dynamic networks: Example-2

- The equation of operation of the network is:

$$\mathbf{a}^1(t) = \mathbf{n}^1(t) = \mathbf{LW}^{1,1}(1)\,\mathbf{a}^1(t\text{-}1) + \mathbf{IW}^{1,1}(0)\mathbf{p}^1(t)$$
$$= lw_{1,1}(1)a(t\text{-}1) + iw_{1,1}p(t)$$

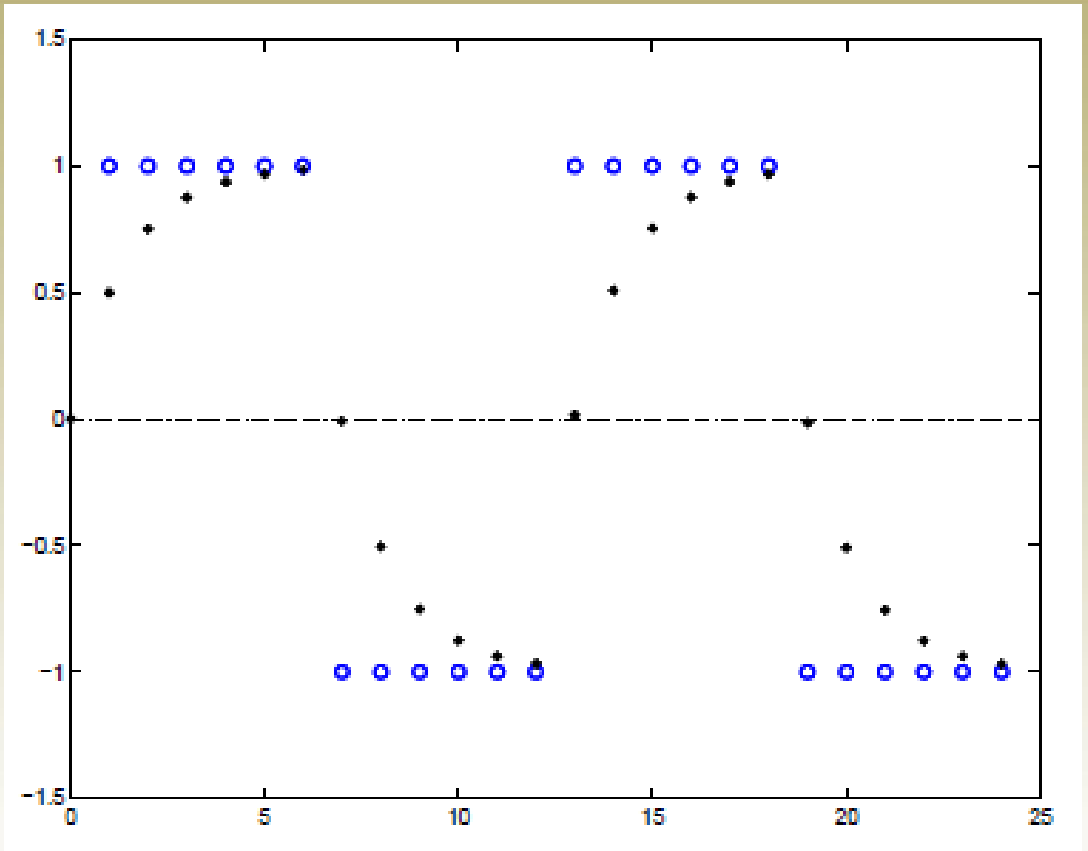
  where, in the last line, we have left off the superscripts, since there is only one neuron and one layer in the network

- To demonstrate the operation of this network, we will set the weight values to:

$$lw_{1,1}(1)=1/2 \text{ and } iw_{1,1}(1)=1/2$$

Dynamic networks: Example-2

- The response of this network to the square wave input is shown in figure
- The network responds exponentially to a change in the input sequence
- Unlike the FIR filter network shown before, the exact response of the network at any given time is a function of the infinite history of inputs to the network



Dynamic networks: Some facts

- Static networks can be trained to approximate static functions, like $sin(p)$, where the output can be computed directly from the current input
- Dynamic networks, on the other hand, can be trained to approximate dynamic systems, such as robot arms, aircraft, biological processes and economic systems, where the current system output depends on a history of previous inputs and outputs
- Because dynamic systems are more complex than static functions, we expect that the training process for dynamic networks will be more challenging than static network training

Principles of Dynamic Learning

- Before we get into the details of training dynamic networks, let's first investigate a simple example.
- Consider again the recurrent network of Example-2
- Suppose that we want to train the network using steepest descent
- The first step is to compute the gradient of the performance function
- For this example we will use sum squared error:

$$F(\mathbf{x}) = \sum_{t=1}^{Q} e^2(t) = \sum_{t=1}^{Q} (t(t) - a(t))^2$$

Principles of Dynamic Learning

- The two elements of the gradient will be:

$$\frac{\partial F(\mathbf{x})}{\partial lw_{1,1}(1)} = \sum_{t=1}^{Q} \frac{\partial e^2(t)}{\partial lw_{1,1}(1)} = -2\sum_{t=1}^{Q} e(t)\frac{\partial a(t)}{\partial lw_{1,1}(1)}$$

$$\frac{\partial F(\mathbf{x})}{\partial iw_{1,1}} = \sum_{t=1}^{Q} \frac{\partial e^2(t)}{\partial iw_{1,1}} = -2\sum_{t=1}^{Q} e(t)\frac{\partial a(t)}{\partial iw_{1,1}}$$

- The key terms in these equations are the derivatives of the network output with respect to the weights:

$$\frac{\partial a(t)}{\partial lw_{1,1}(1)} \quad \text{and} \quad \frac{\partial a(t)}{\partial iw_{1,1}}$$

Principles of Dynamic Learning

- If we had a static network, then these terms would be very easy to compute
  - They would correspond to $\alpha(t\text{-}1)$ and $p(t)$, respectively
- However, for recurrent networks, the weights have two effects on the network output
  - The first is the direct effect, which is also seen in the corresponding static network.
  - The second is an indirect effect, caused by the fact that one of the network inputs is a previous network output
- Let's compute the derivatives of the network output, in order to demonstrate these two effects

Principles of Dynamic Learning

- The equation of operation of the network is:

$$a(t) = lw_{1,1}(1)a(t–1) + iw_{1,1}p(t)$$

- We can compute the terms in the equations at the botton of Slide 21 by taking the derivatives of the previous equation:

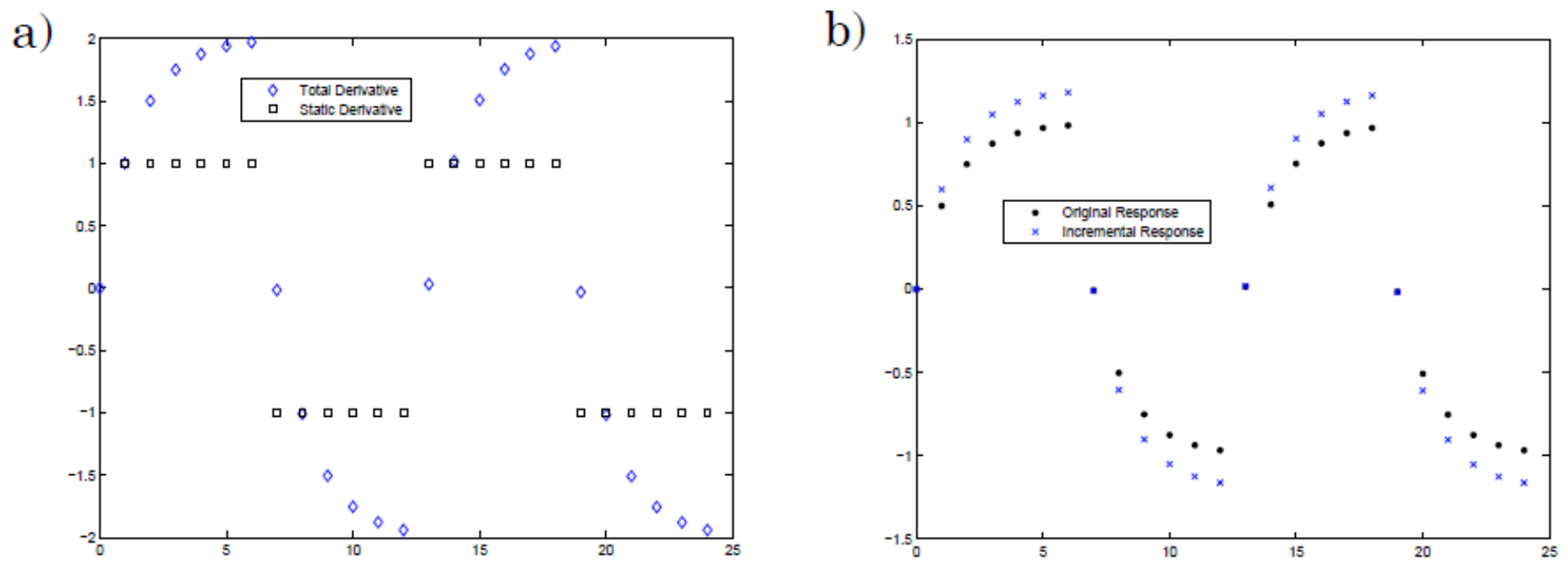
$$\frac{\partial a(t)}{\partial lw_{1,1}(1)} = a(t-1) + lw_{1,1}(1)\frac{\partial a(t-1)}{\partial lw_{1,1}(1)}$$

$$\frac{\partial a(t)}{\partial iw_{1,1}} = p(t) + lw_{1,1}(1)\frac{\partial a(t-1)}{\partial iw_{1,1}}$$

- The first term in each of these equations represents the direct effect that each weight has on the network output
- The second term represents the indirect effect
- Note that unlike the gradient computation for static networks, the derivative at each time point depends on the derivative at previous time points (or at future time points)

Principles of Dynamic Learning

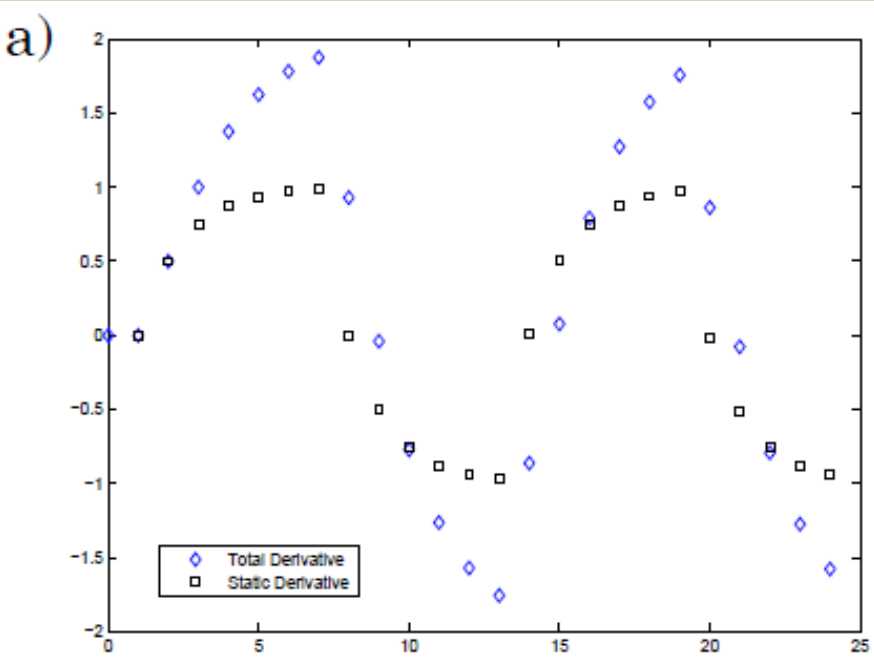
- The following figures illustrate the dynamic derivatives

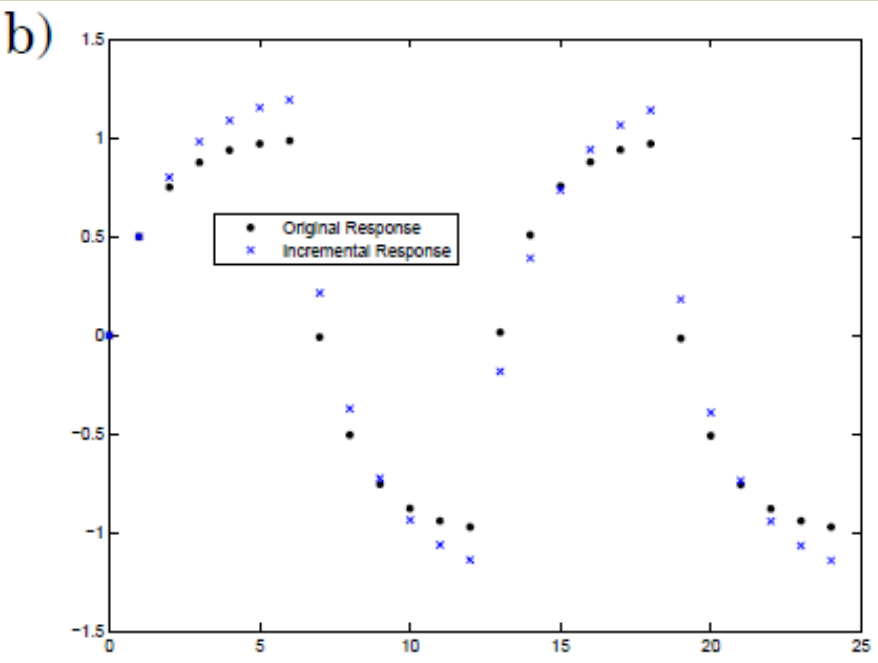


- In the left figure we see the total derivatives $\partial a(t)/\partial iw_{1,1}$ and also the static portions of the derivatives. Note that if we consider only the static portion, we will underestimate the effect of a change in the weight. In the right figure we see the original response of the network and a new response, in which $iw_{1,1}$ is increased from 0.5 to 0.6. By comparing the above two figures, we can see how the derivative indicates the effect on the network response of a change in the weight $iw_{1,1}$.

Principles of Dynamic Learning

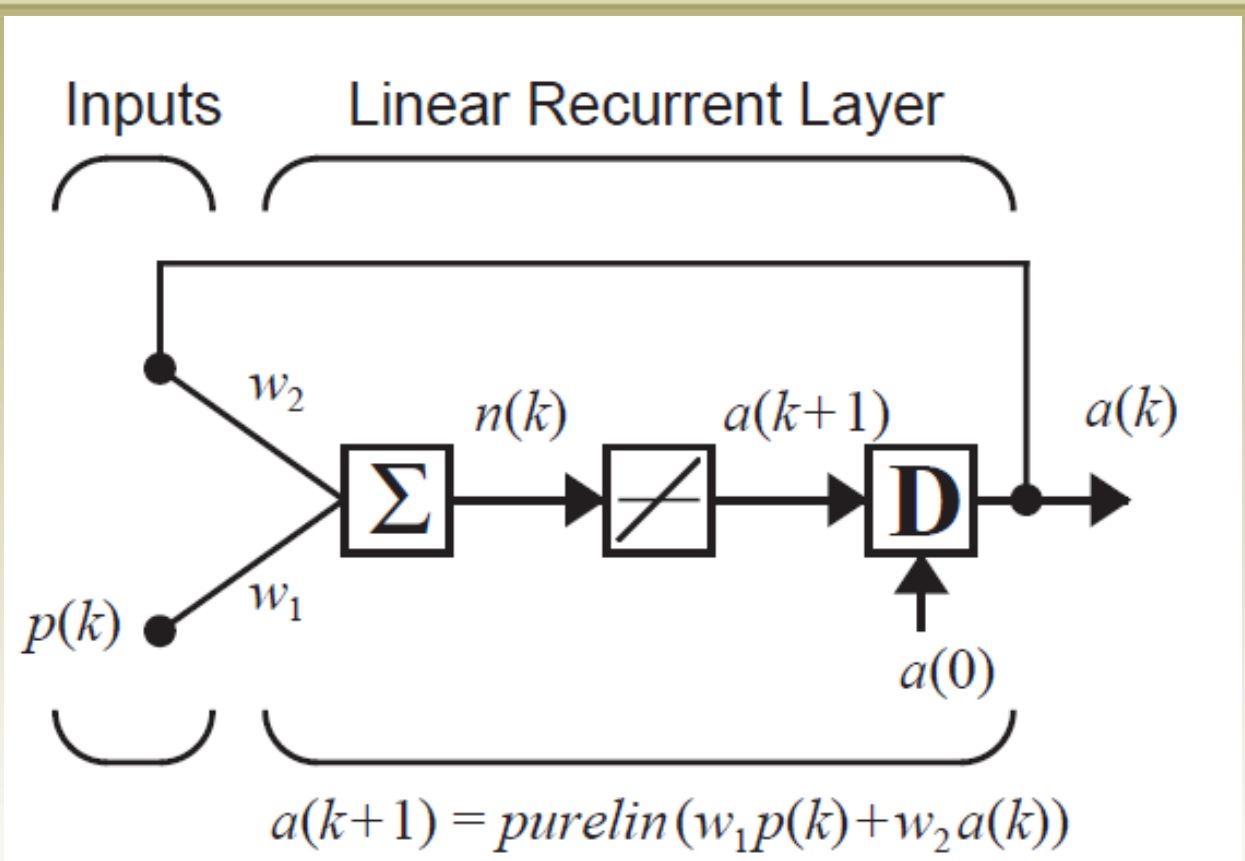
- In the figures below we see similar results for $lw_{1,1}(1)$
- The key ideas to get from this example are: 1) the derivatives have static and dynamic components, and 2) the dynamic component depends on other time points





Derivation of BackProp equations for a two-layer Recurrent Neural Network

BackProp equations for a two-layer RNN



- The first step is to define our performance index. As with the multilayer networks, we will use squared error:

$$F(\mathbf{x}) = (t(k) - a(k))^2 = (e(k))^2$$

BackProp equations for a two-layer RNN

- For our weight updates we will use the steepest descent algorithm:

$$\Delta w_i = -\alpha \frac{\partial}{\partial w_i} \hat{F}(\mathbf{x})$$

- These derivatives can be computed as follows:

$$\frac{\partial}{\partial w_i} \hat{F}(\mathbf{x}) = \frac{\partial}{\partial w_i} (t(k) - a(k))^2 = 2(t(k) - a(k)) \left\{ - \frac{\partial a(k)}{\partial w_i} \right\}$$

- Therefore, the key terms we need to compute are:

$$\frac{\partial a(k)}{\partial w_i}$$

- To compute these terms we first need to write out the network equation:

$$a(k+1) = purelin(w_1 p(\mathrm{k}) + w_2 a(\mathrm{k})) = w_1 p(\mathrm{k}) + w_2 a(\mathrm{k})$$

BackProp equations for a two-layer RNN

- Next we take the derivative of both sides of this equation with respect to the network weights:

$$\frac{\partial a(k+1)}{\partial w_1} = p(k) + w_2 \frac{\partial a(k)}{\partial w_1}$$

$$\frac{\partial a(k+1)}{\partial w_2} = a(k) + w_2 \frac{\partial a(k)}{\partial w_2}$$

- (Note that we had to take account of the fact that a(k) is itself a function of w_1 and w_2.) These two recursive equations are then used to compute the derivatives needed for the steepest descent weight update. The equations are initialized with (since the initial condition is not a function of the weight):

$$\frac{\partial a(0)}{\partial w_1} = 0, \quad \frac{\partial a(0)}{\partial w_2} = 0$$

BackProp equations for a two-layer RNN

- To illustrate the process, let's say that a(0)=0. The first network update would be

$$a(1) = w_1 p(0) + w_2 a(0) = w_1 p(0)$$

- The first derivatives would be computed:

$$\frac{\partial a(1)}{\partial w_1} = p(0) + w_2 \frac{\partial a(0)}{\partial w_1} = p(0), \quad \frac{\partial a(1)}{\partial w_2} = a(0) + w_2 \frac{\partial a(0)}{\partial w_2} = 0$$

- The first weight updates would be:

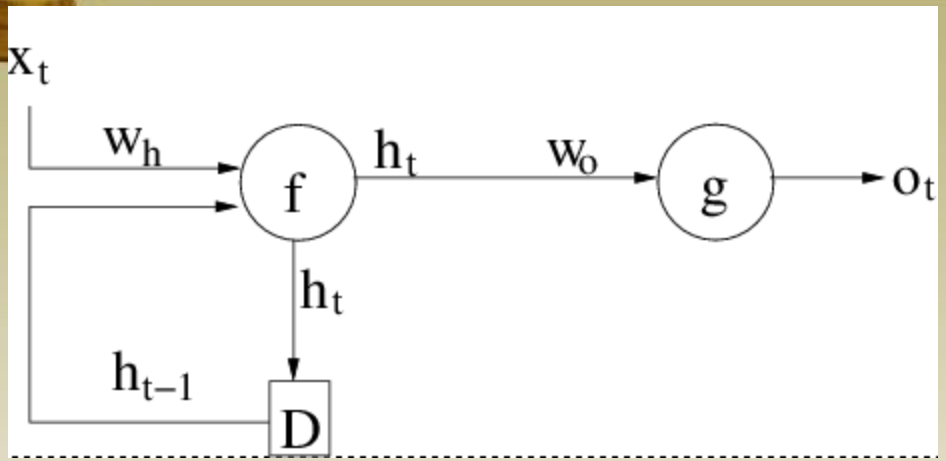
$$\Delta w_i = -\alpha \frac{\partial}{\partial w_i} \hat{F}(\mathbf{x}) = -\alpha \left[2(t(1) - a(1)) \left\{ - \frac{\partial a(1)}{\partial w_i} \right\} \right]$$

$$\Delta w_1 = -2\alpha(t(1) - a(1))\{-p(0)\}$$

$$\Delta w_2 = -2\alpha(t(1) - a(1))\{0\} = 0.$$

Derivation of BackProp equations for a three-layer Recurrent Neural Network: The generic case

BPTT in a three-layer RNN: General idea



$$h_t = f(x_t, h_{t-1}, w_h)$$

$$o_t = g(h_t, w_o)$$

$$L = \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t)$$

$$\frac{\partial L}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_h)}{\partial h_t}\frac{\partial h_t}{\partial w_h}$$

h_t depends on both h_{t-1} and w_h, where the computation of h_{t-1} depends on w_h

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial w_h}$$

BPTT in a three-layer RNN: General idea

BACKGROUND: Assume three sequences $\{a_t\}$, $\{b_t\}$, $\{c_t\}$ with $a_0=0$, and $\mathbf{a_t = b_t + c_t a_{t-1}}$ for t=1,2,... Then, for $t \geq 1$, it can be shown that:

$$a_t = b_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} c_j \right) b_i$$

If we substitute:

$$a_t = \frac{\partial h_t}{\partial w_h}$$

$$b_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h}$$

$$c_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}}$$

BPTT in a three-layer RNN: General idea

Then, we get:

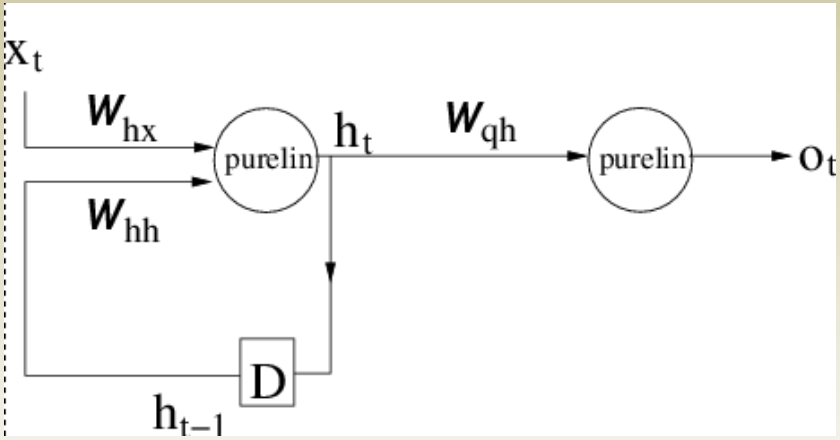
$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}} \right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}$$

Derivation of BackProp equations for a three-layer Recurrent Neural Network: The detailed case

Detailed BPTT in a three-layer RNN

Now, we show how to compute the gradients of the objective function with respect to all the decomposed model parameters.

We consider an RNN without bias parameters, whose activation function in the hidden and output layers uses the identity mapping (f(x)=x).



$$\mathbf{h}_t = \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1}$$

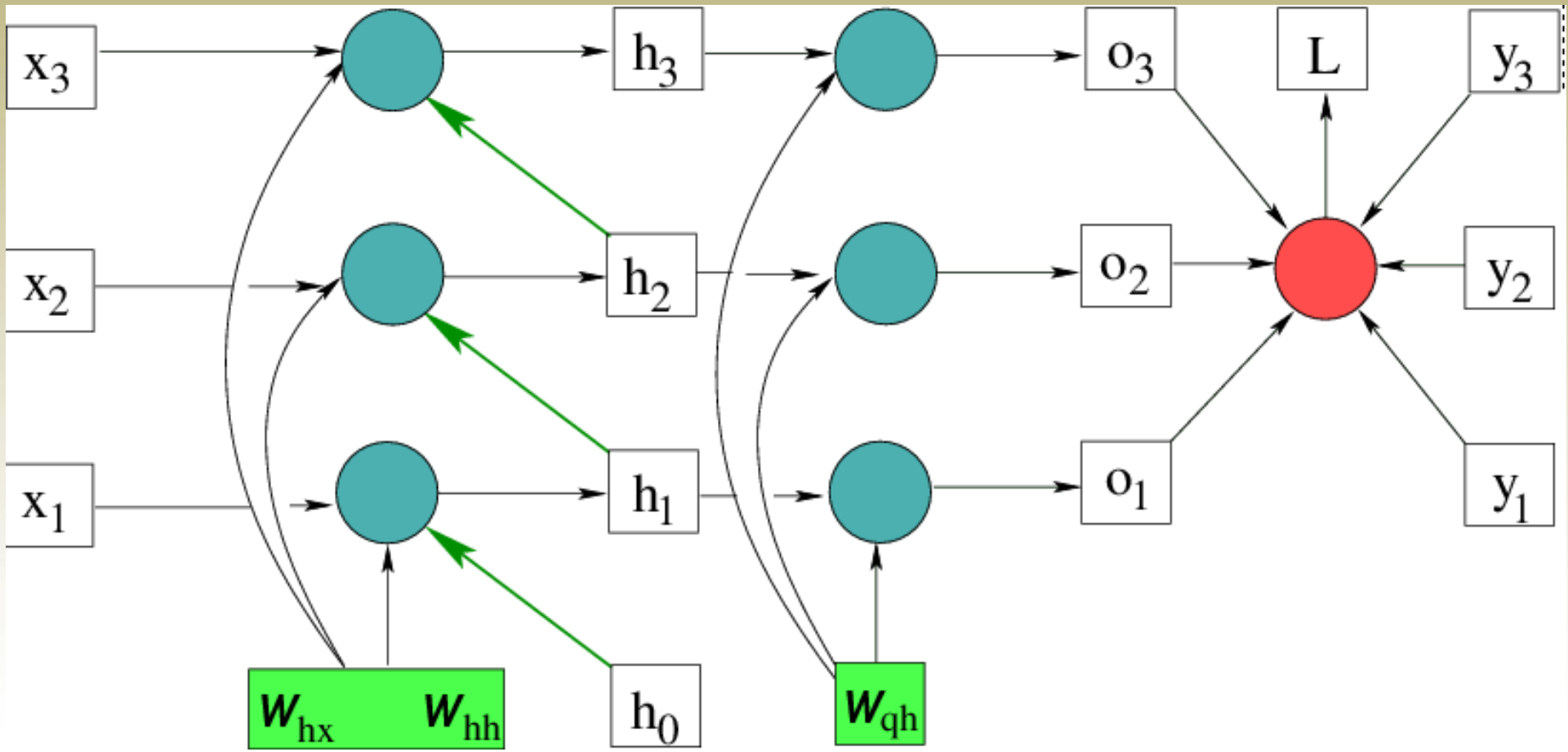
$$\mathbf{o}_t = \mathbf{W}_{qh}\mathbf{h}_t$$

$$L = \frac{1}{T}\sum_{t=1}^{T} l(\mathbf{o}_t, y_t)$$

Therefore, the model parameters are: $\mathbf{W}_{hx}$, $\mathbf{W}_{hh}$, $\mathbf{W}_{qh}$, and the derivative of the loss function L with respect to the model output at any time step t is given by:

$$\frac{\partial L}{\partial \mathbf{o}_t} = \frac{1}{T}\frac{\partial l(\mathbf{o}_t, y_t)}{\partial \mathbf{o}_t} \qquad \text{(RNN-1)}$$

Detailed BPTT in a three-layer RNN



Detailed BPTT in a three-layer RNN

We will calculate the gradient of the loss function with respect to the parameter $\mathbf{W}_{qh}$ in the output layer:

$$\frac{\partial L}{\partial \mathbf{W}_{qh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_{qh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{o}_t} \mathbf{h}_t^{\mathcal{T}} \qquad \text{(RNN-2)}$$

Denotes the transpose

At the final time step T, the loss function L depends on the hidden state $\mathbf{h}_t$ only via $\mathbf{o}_T$. We can easily compute the following gradient:

$$\frac{\partial L}{\partial \mathbf{h}_T} = \frac{\partial L}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_T} = \mathbf{W}_{qh}^{\mathcal{T}} \frac{\partial L}{\partial \mathbf{o}_t} \qquad \text{(RNN-3)}$$

Can be calculated via (RNN-1)

Detailed BPTT in a three-layer RNN

For time steps t < T, the loss function L depends on $\mathbf{h}_t$ via $\mathbf{h}_{t+1}$ and $\mathbf{o}_t$.
We compute the gradient recurrently as follows:

The green arrows

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial L}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} = \mathbf{W}_{hh}^{\mathcal{T}} \frac{\partial L}{\partial \mathbf{h}_{t+1}} + \mathbf{W}_{qh}^{\mathcal{T}} \frac{\partial L}{\partial \mathbf{o}_t} \qquad \text{(RNN-4)}$$

Expanding this recurrence for any time step $1 \le t \le T$, we get:

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{i=t}^{T} \left(\mathbf{W}_{hh}^{\mathcal{T}}\right)^{T-i} \mathbf{W}_{qh}^{\mathcal{T}} \frac{\partial L}{\partial \mathbf{o}_{T+t-i}} \qquad \text{(RNN-5)}$$

If eigenvalues < 1 → vanish
If eigenvalues > 1 → explode

Detailed BPTT in a three-layer RNN

Finally, the loss function L depends on $\mathbf{W}_{hx}$ and $\mathbf{W}_{hh}$ in the hidden layer via hidden states $\mathbf{h}_1$, $\mathbf{h}_2$,…, $\mathbf{h}_T$. We compute the respective gradients applying the chain rule and we get:

$$\frac{\partial L}{\partial \mathbf{W}_{hx}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hx}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{x}_t^{\mathcal{T}} \qquad \text{(RNN-6)}$$

$$\frac{\partial L}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{h}_{t-1}^{\mathcal{T}} \qquad \text{(RNN-7)}$$

Can be calculated via (RNN-3) and (RNN-5)