VI. ADAPTIVE NETWORKS: TEMPORAL PROCESSING

- Static vs Dynamic Systems
- Topologies for Temporal Processing
- Time Delay Neural Networks (TDNN)
- FIR Multilayer Perceptron
- Temporal Backpropagation Learning
- Recurrent Neural Networks
- Real Time Recurrent Learning

Temporal Network Classification

Static:

- All parameters are fixed with respect to time very hard to represent time functions

Dynamic:

- Memory is required
- Memory can be in the form of time delays
- Suitable for representing time-related functions

6.1. Static vs Dynamic Systems

- There are systems that do not have instantaneous responses to inputs. They need some time to stabilize their outputs for a given input excitation.
- A linear combiner has memory, that is, it preserves the past values of the input internally in the tap delay line.
- Another important class of systems with nonzero transient response is the recurrent topology.
- Both these systems (memory based or recurrent) are called dynamical networks.
- All analog systems are dynamic, because their responses take a finite time to reach a steady value. This naturally creates the concept of time scale.

- In static modeling we are using statistical properties of the data clusters to distinguish them. In time phenomena, time imposes a structure in the input space that may be used to separate data clusters with overlapping statistics; in other words, the sequence in which the points are visited is different, and this difference can be used for separation.
- Dynamic neural networks are topologies designed to explicitly include time relationships in the input-output mappings.
- Short Term memory- Long Term memory
- Feedforward- Recurrent Networks

Memory in Process

- We can introduce memory effects with two principal ways:
  - *Implicit:* e.g. Time lagged signal as input to a static network or as recurrent connections
  - *Explicit:* e.g. Temporal Backpropagation Method
- In the implicit form, we assume that the environment from which we collect examples of (input signal, output sequence) is *stationary.* For the explicit form the environment could be *non-stationary*, i.e. the network can track the changes in the structure of the signal.

Memory in Practice

- The time delayed approach includes two basic types of networks:
  - *Implicit Representation of Time:* We combine a memory structure in the input layer of the network with a static network model
  - *Explicit Representation of Time:* We explicitly allow the network to code time, by generalising the network weights from scalars to vectors, as in TBP (Temporal Backpropagation).
- Typical forms of memories that are used are the *Tapped Delay Line* and the *Gamma Memory* family.

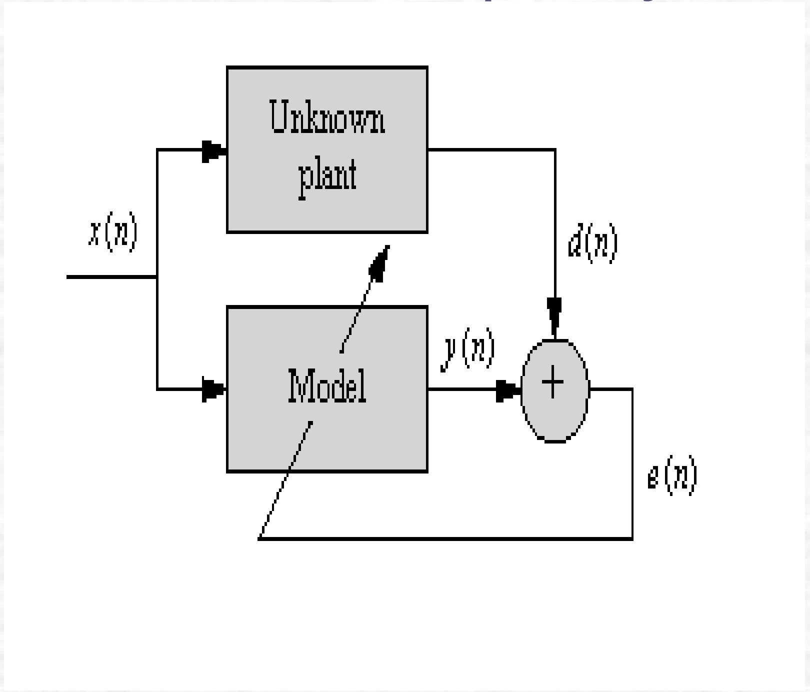
Nonlinear System Identification with Neural Networks

- we are interested in modeling the input-output relationship of an unknown plant. This problem is a practical application of function approximation. In fact, we can consider the output of the system $d(n)$ as a fixed but unknown function of $x(n)$, that is,

  $d(n) = f(x(n))$. The role of the model is to approximate the function $f(.)$ by using a set of bases that are defined by the topology of the model system.

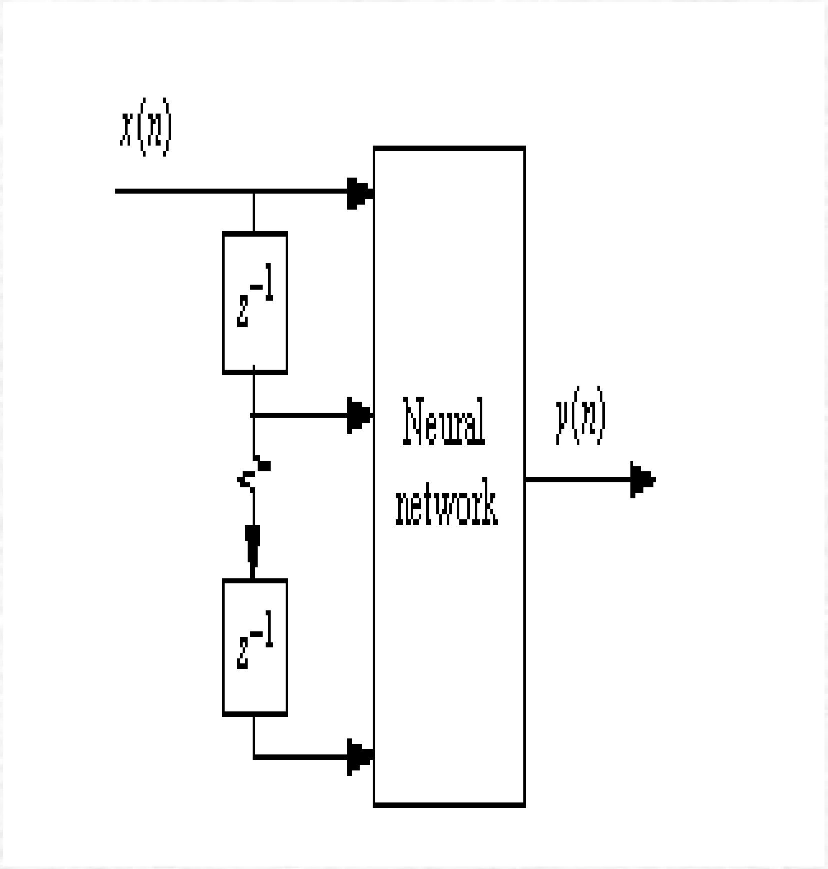
Nonlinear System Identification by using Focused TLFN (Time Lagged FN)

- The short-term memory in TLFNs can be of any type and distributed in any layer. In this section we study a special case of TLFNs called focused TLFNs. the memory in the network is restricted to the input layer



- The standart back-prop can be used to perform nonlinear predictionof a stationary time series.
- The task is to find an appropriate network and a set of parameters which can best model an unknown target system.

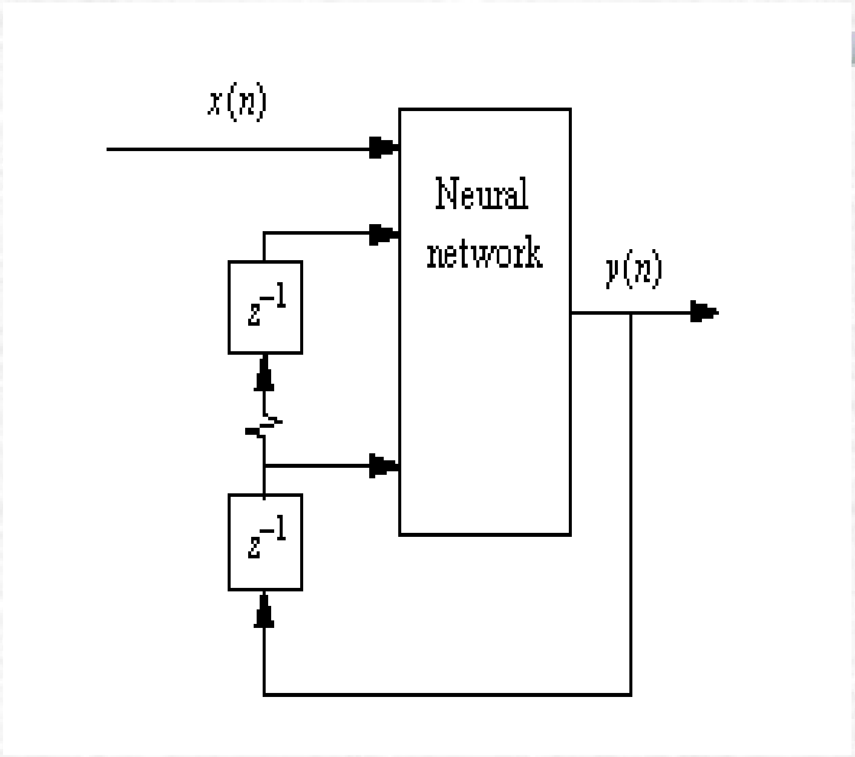
NMA-Nonlinear Moving Average



- In nonlinear moving-average (NMA) modeling, the output of the model is a nonlinear function of its input

y(n+1)=f(x(n),x(n-1),..,x(n-k+1))

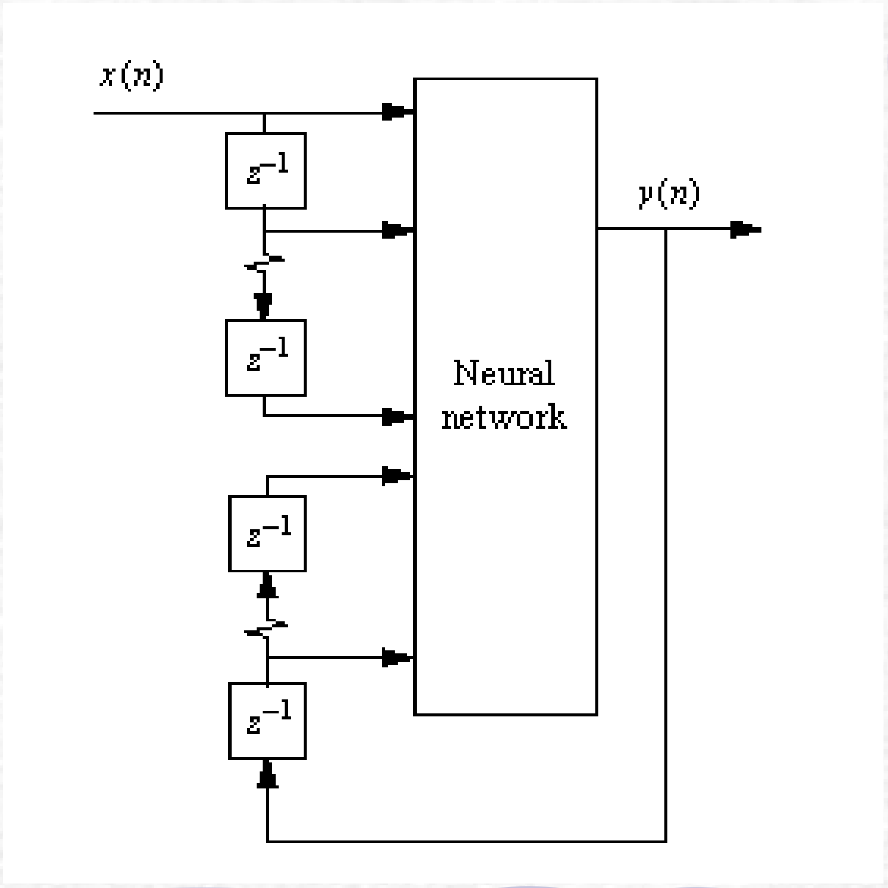
NARX- Nonlinear AutoRegressive with eXogenaoous input



- In nonlinear autoregressive with external input (NARX) models the output of the model is given by

y(n+1)=f(y(n),y(n-1),..,y(n-k+1),x(n))

NARMA- Nonlinear AutoRegressive Moving-Average



- It is the most general class of nonlinear models and is a blend of the two previous types

  y(n+1)=f(y(n),y(n-1),..,y(n-k+1),
  x(n-1),…,x(n-j))

- A fundamental aspect of system identification is to understand when a given model should be used given the knowledge that we have about the plant.

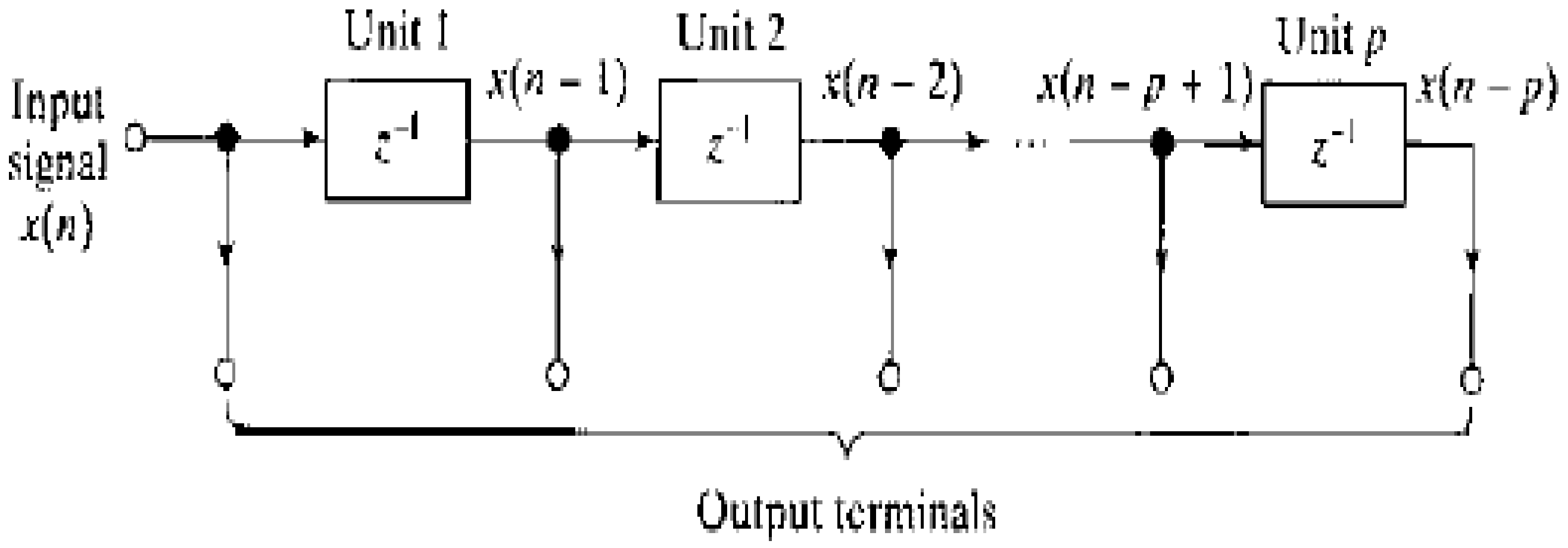
6.2. Time Delay Neural Networks

- The TDNN topology is embodied in an FIR multilayer perceptron.
- The FIR perceptron attain dynamic behaviour with its synapse designed as an FIR filter.
- For its training, the standart Back-prop can be used by constructing a static equivalent (unfolding the FIR-MLP in time) or use Temporal Back-prop (Wan).

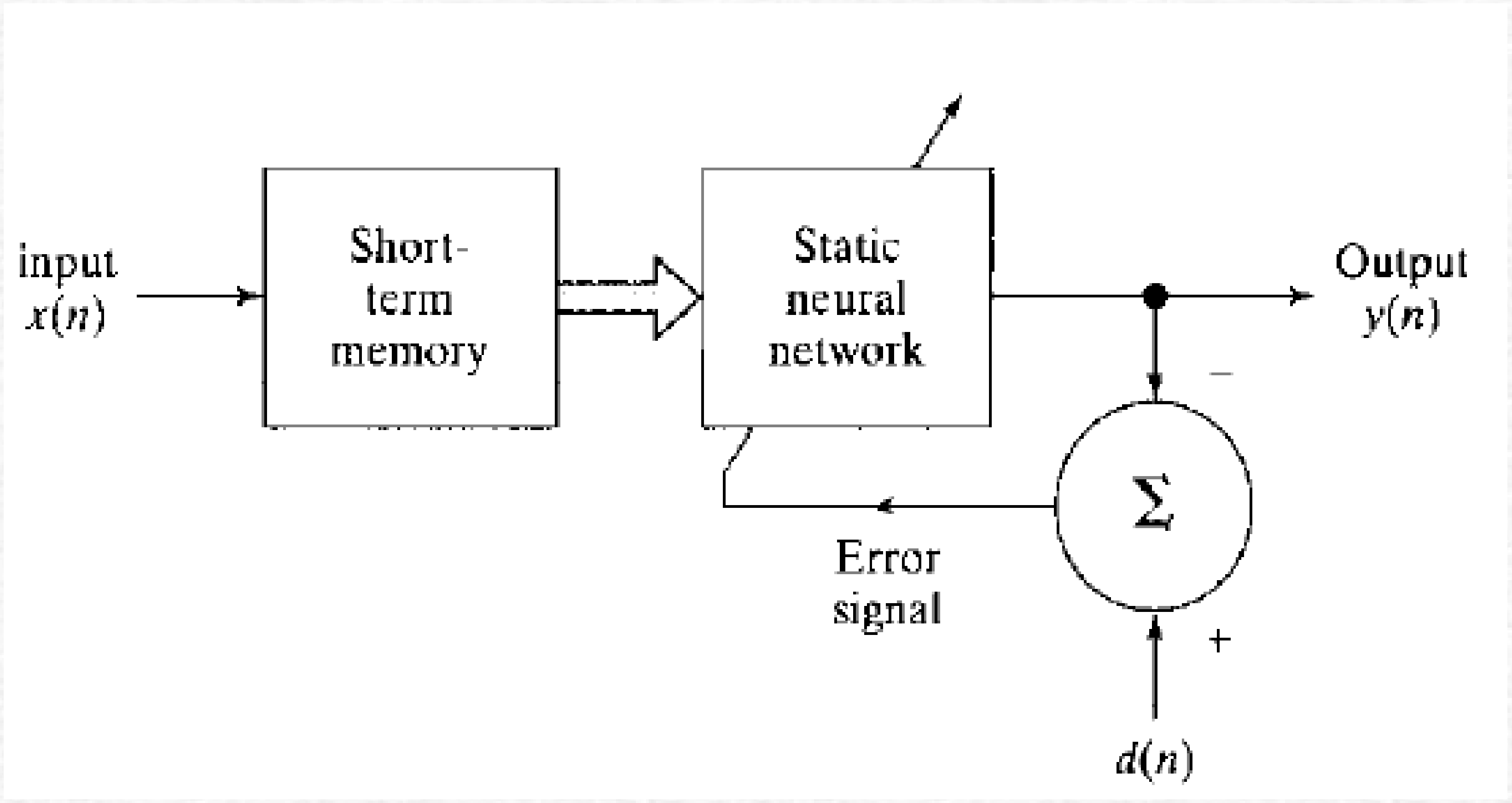
Distributed vs. Focused Networks

- Focused Network
  - Additional Neurons added at input for memory
  - Suitable for shift invariant series
- Distributed Network
  - Memory added to each neuron in the form of additional inputs
  - Allows for time variant series

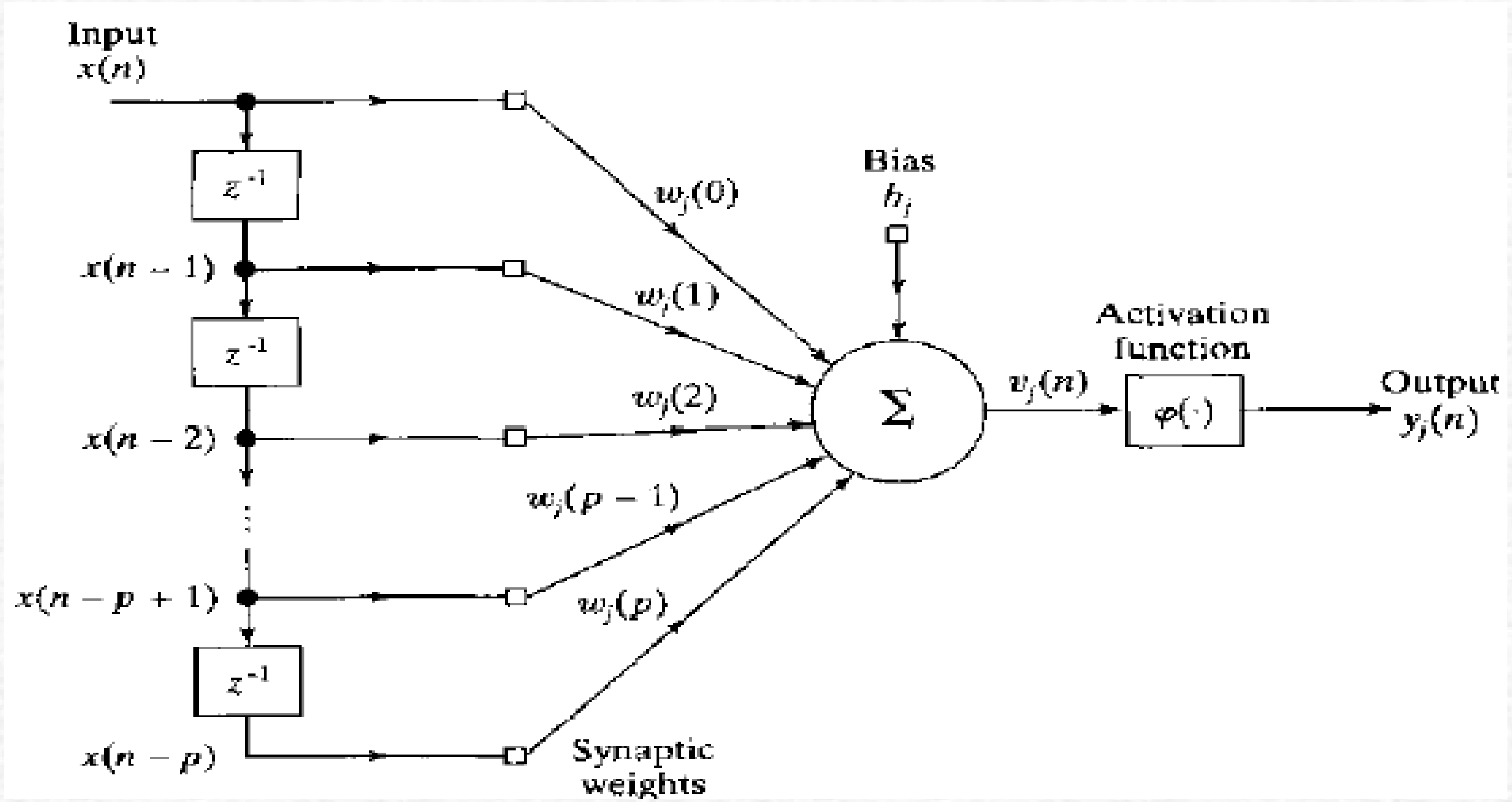
Tapped Delay Memory



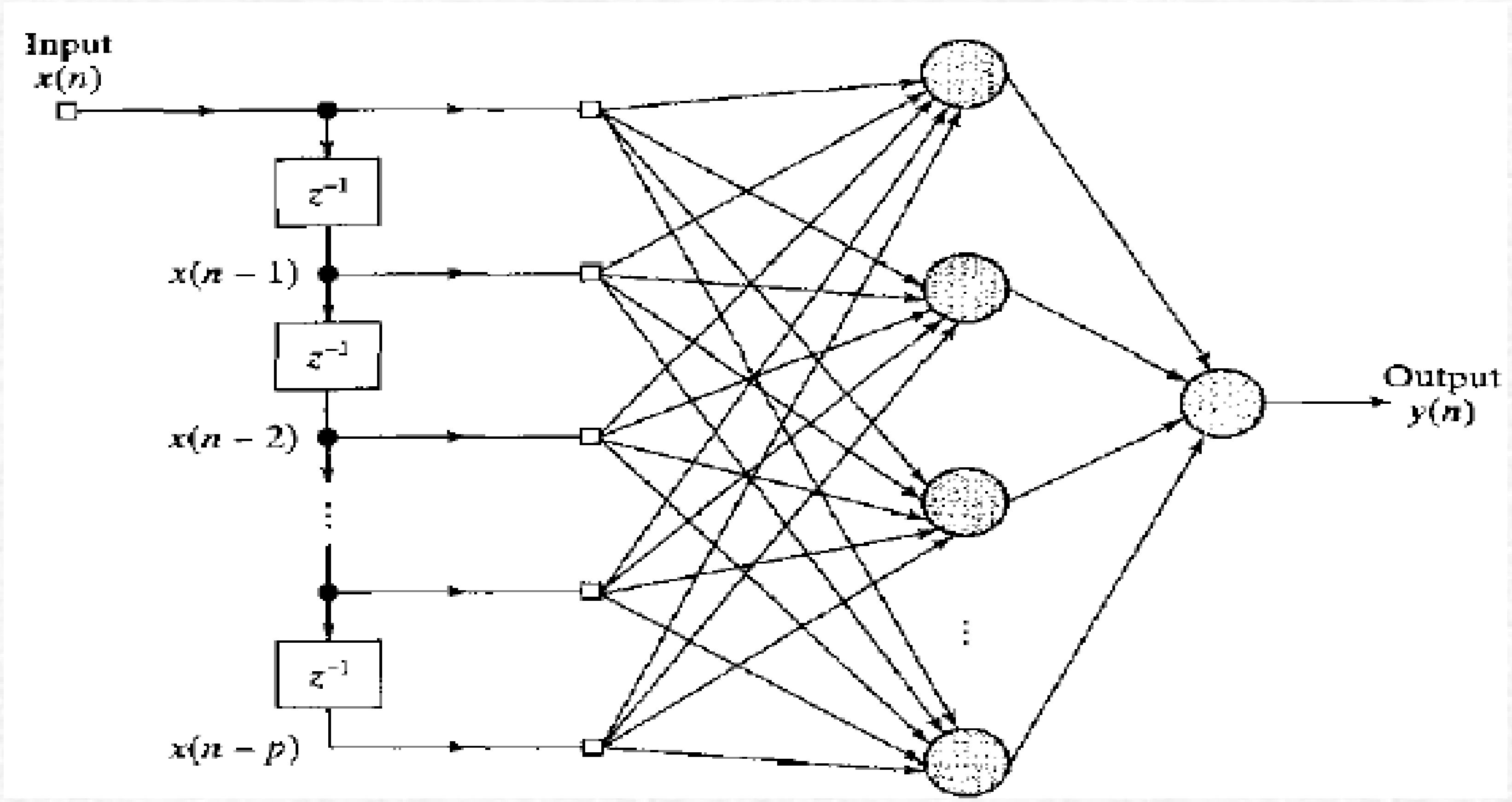
Time Lagged Feedforward Networks



Focused Neuron Model



Focused Time Lagged Feedforward Network



- NETtalk (Sejnowski and Rosenberg, 1987) was the first demonstration of a massively parallel distributed network that converts English speech to phonemes.
- NETtalk was based on a multilayer perceptron with an input layer of 203 sensory nodes, a hidden layer of 80 neurons, and an output layer of 26 neurons. All the neurons used sigmoid activation functions. The synaptic connections in the network were specified by a total of 18629 weights. The standard back-propagation algorithm was used to train the network.

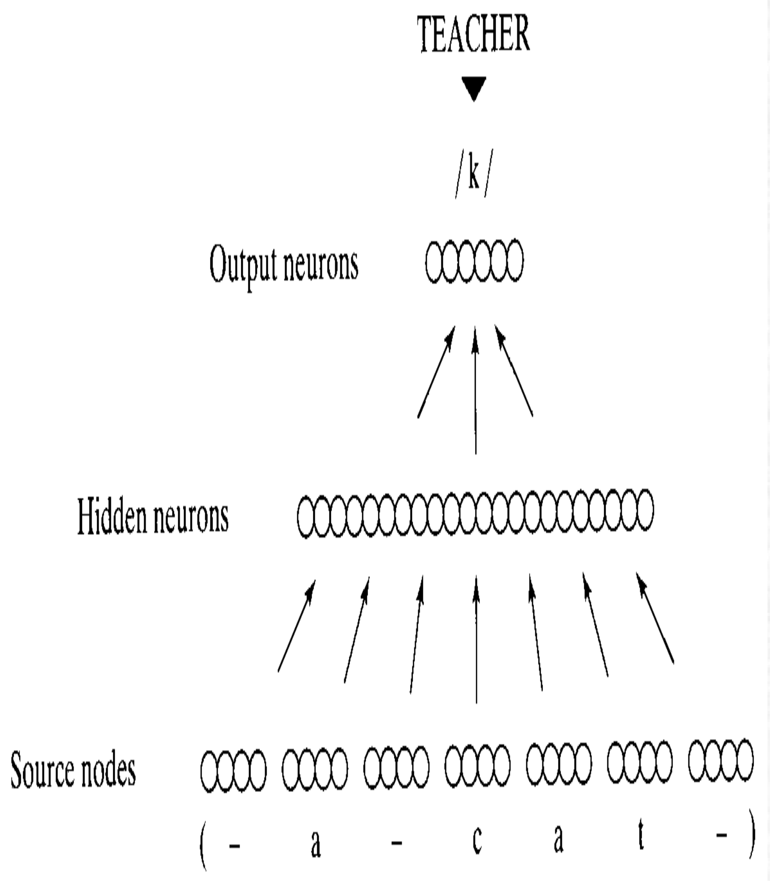


FIGURE 13.6 Schematic diagram of the NETtalk network architecture.

TDNN

- Time-delay neural network(TDNN) (Lang and Hinton, 1988) is a multilayer feedforward network
- The TDNN is a multilayer feedforward network whose hidden neurons and output neurons are replicated across time. It was devised to capture explicitly the concept of time symmetry as encountered in the recognition of isolated word (phoneme) using a spectrogram

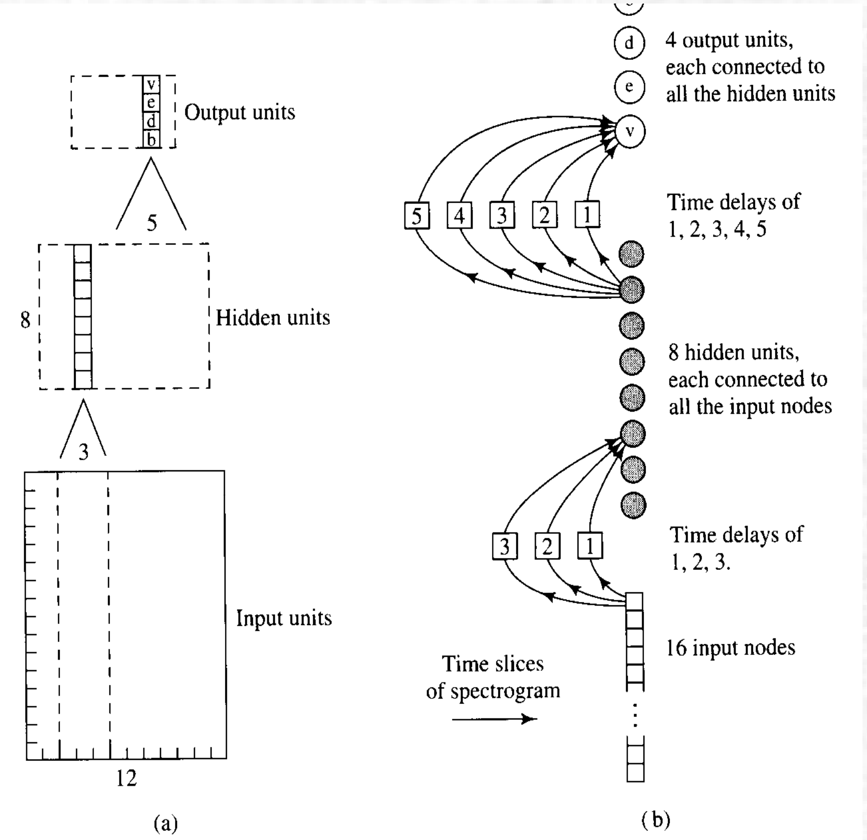

v
e
d
b
Output units
5
8
Hidden units
3
Input units
12
(a)
d
e
v
4 output units, each connected to all the hidden units
5 4 3 2 1
Time delays of 1, 2, 3, 4, 5
8 hidden units, each connected to all the input nodes
3 2 1
Time delays of 1, 2, 3.
16 input nodes
Time slices of spectrogram
(b)

FIGURE 13.7 (a) A network whose hidden neurons and output neurons are replicated across time. (b) Time delay neural network (TDNN) representation. (From K. J. Lang and G. E. Hinton, 1988, with permission)

Training Algorithms

$$E(n) = \frac{1}{2}\sum_{j} e_j^2(n)$$

$$\underbrace{w_{ji}(n+1)}_{\text{update value}} = \underbrace{w_{ji}(n)}_{\text{old value}} + \underbrace{\Delta w_{ji}(n)}_{\text{correction}}$$

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}}$$

- Unfolding the network in time,
- Temporal Backpropagation.

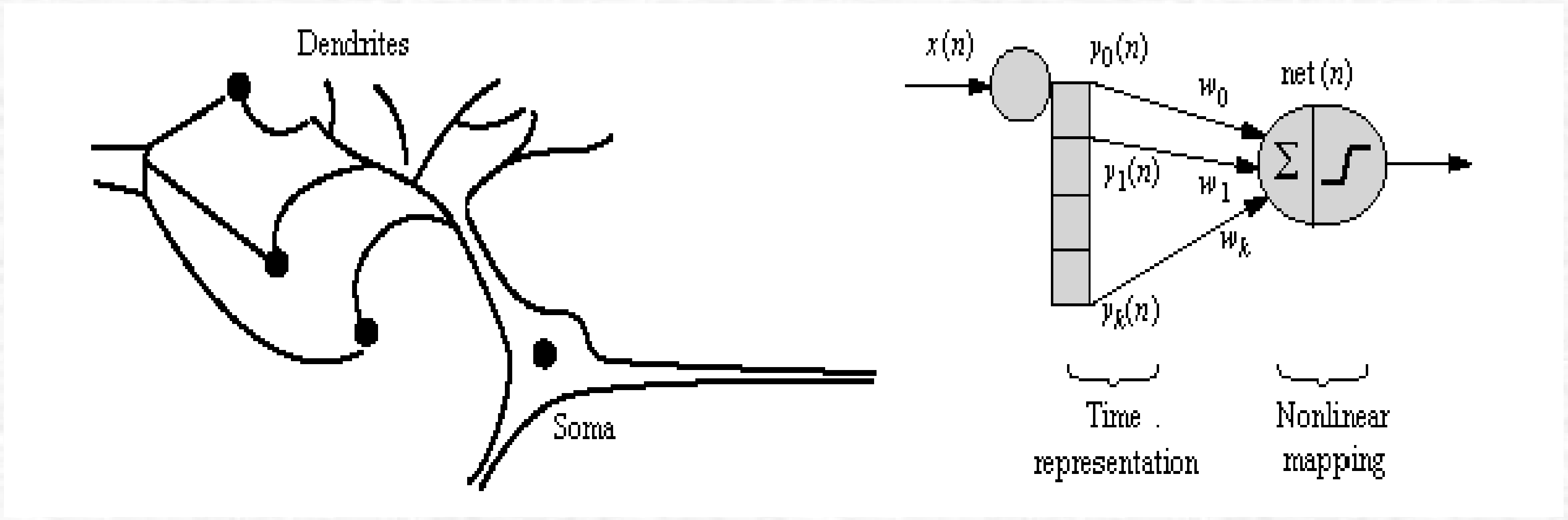
Unfolding the network in time

- Remove all the time delays by expanding the network into an equivalent ‘static’ network, then apply the standart backpropagation algorithm: All the time dependences are made external.
- Some disadvantages:

1. There is a loss of a sense of symmetry between the forward and the backward propagation for the error gradients,
2. There is no nice recursive formula,
3. There is a need for global bookkeeping to keep track of static weights in the equivalent network.

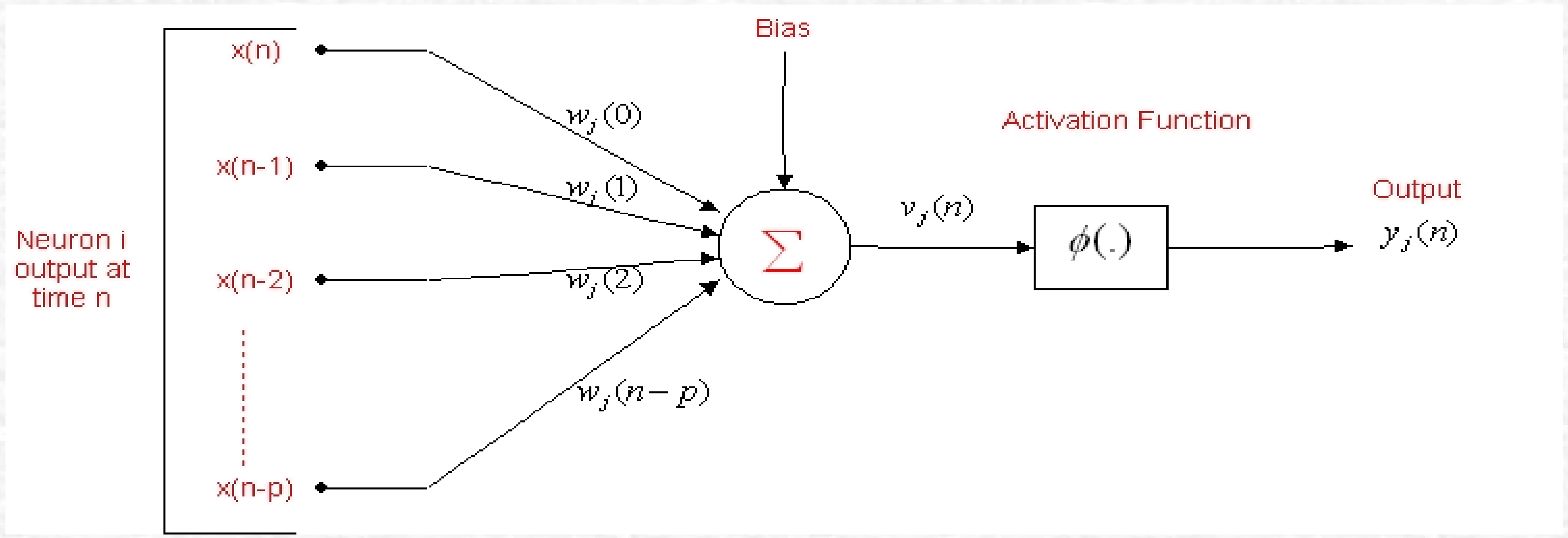
FIR Multilayer Perceptron

- AI neurons are based on a finite-duration impulse response (FIR) model.
- Similarity between biological neuron and the FIR perceptron



FIR Synapse

- In explicit time representation, neurons have a spatio-temporal structure, i.e. its synapse arriving to a neuron is not a scalar number but a vector of weights, which are used for convolution of the time-delayed input signal of a previous neuron with the synapse.
- A schematic representation of a neuron is given below:



FIR Synapse

- The output of the neuron in this case is given by:

$$y_j(n) = \varphi\left(\sum_{l=0}^{p} w_j(l)\,x(n-l) + b_j\right)$$

- In case of a whole network, for example assuming a single output node and a linear output layer, the response is given by:

$$y(n) = \sum_{j=1}^{m_1} w_j y_j(n) = \sum_{j=1}^{m_1} w_j \varphi\left(\sum_{l=0}^{p} w_j(l)\,x(n-l) + b_j\right) + b_0$$

- Where p is the depth of the memory and b_0 is the bias of the output neuron

- In the more general case, where we have multiple neurons at the output layer, we have for neuron j of any layer:

$$y_j(n) = \varphi\left(\sum_{i=1}^{m_0} \sum_{l=0}^{p} w_{ji}(l) x_i(n-l) + b_j \right)$$

- The output of any synapse is given by the *convolution sum*:

$$s_{ji}(n) = \vec{w}_{ji}^{\,T} \vec{x}_i(n) = \sum_{l=0}^{p} w_{ji}(l) x_i(n-l)$$

Temporal Backpropagation

- The approach is based on an approximation to the method of deepest descent.

→ $\mathbf{w}_{ji}(n+1) = \mathbf{w}_{ji}(n) - \eta\, \delta_j(n)\mathbf{x}_i(n)$ where the explicit form of the local gradient $\delta_j(n)$ depends on wether or not neuron j lies in the output layer or in a hidden layer of the network.

- we prefer to calculate the gradient of the cost function as follows:

$$\frac{\partial E_{total}}{\partial \vec{w}_{ji}} = \sum_n \frac{\partial E_{\text{total}}}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \vec{w}_{ji}}$$

- Note that in general holds:

$$\frac{\partial E_{total}}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \vec{w}_{ji}} \neq \frac{\partial E(n)}{\partial \vec{w}_{ji}}$$

- The equality is correct only when we take the sum over all time.

Weight Adjustment

- To calculate the weight update we use the steepest descent method:

$$\vec{w}_{ji}(n+1) = \vec{w}_{ji}(n) - \eta \frac{\partial E_{total}}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \vec{w}_{ji}(n)}$$

- Where η is the learning rate.
- We calculate the terms in the above relation as follows:

$$\frac{\partial v_j(n)}{\partial \vec{w}_{ji}(n)} = \vec{x}_i(n)$$

- This is by definition the induced field $v_j(n)$

- For the output layer the local gradient is given by:

$$\delta_j(n) = -\frac{\partial E_{total}}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial v_j(n)} = e_j(n)\varphi'(v_j(n))$$

- For a hidden layer we assume that neuron j is connected to a set A of neurons in the next layer (hidden or output). Then we have:

$$\delta_j(n) = -\frac{\partial E_{total}}{\partial v_j(n)}$$

$$= -\sum_{r \in A} \sum_{k} \frac{\partial E_{total}}{\partial v_r(k)} \frac{\partial v_r(k)}{\partial v_j(n)}$$

- By re-writing we get the following:

$$\delta_j(n) = \sum_{r\in A}\sum_k \delta_r(k)\frac{\partial v_r(k)}{\partial v_j(n)}$$

$$= \sum_{r\in A}\sum_k \delta_r(k)\frac{\partial v_r(k)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)}$$

- Finally we putting all together we get:

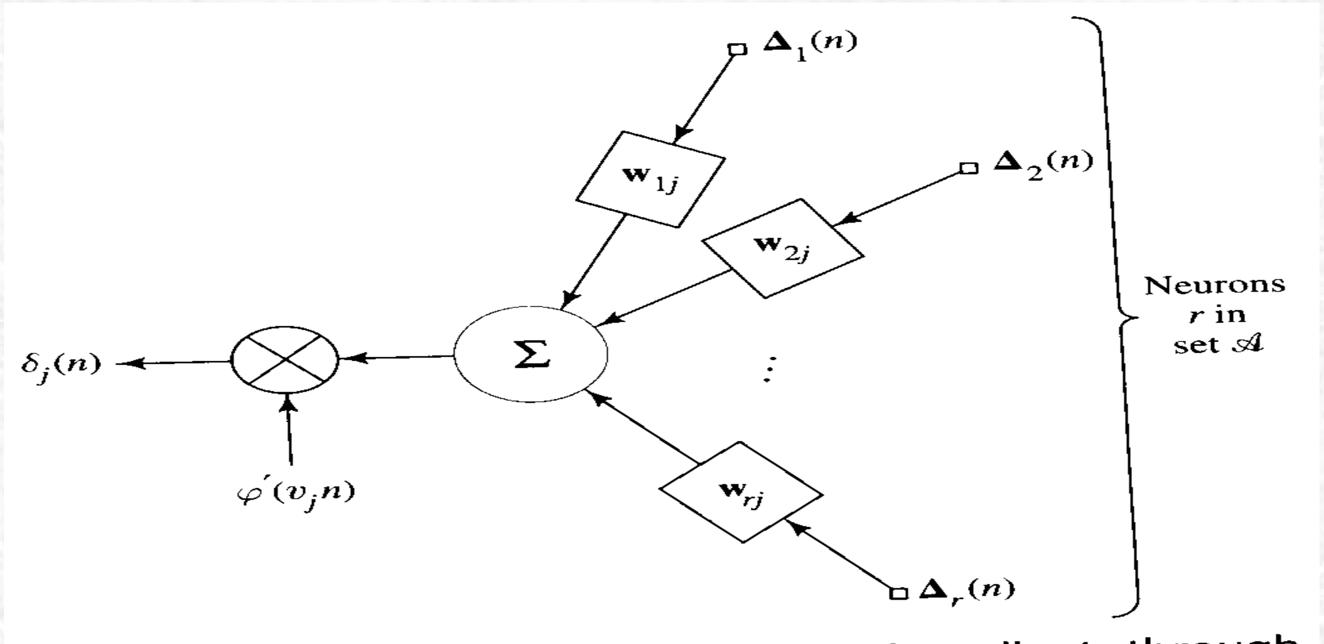
$$\delta_j(n) = \phi'(v_j(n))\sum_{r\in A}\sum_{k=n}^{n+p}\delta_r(k)w_{rj}(k-l)$$

$$= \phi'(v_j(n))\sum_{r\in A}\sum_{l=0}^{p}\delta_r(n+l)w_{rj}(n)$$

$\delta_j(n) = e_j(n)\,\phi'(v_j(n))$, neuron j in the output layer

$\delta_j(n) = \phi'(v_j(n))\sum_{r\in A}\Delta^T{}_r(n)\,\mathbf{w}_{rj}$, neuron j in a hidden layer

a vector generalization of the standard backpropagation algorithm.



$\boldsymbol{\Delta}_1(n)$
$\mathbf{w}_{1j}$
$\boldsymbol{\Delta}_2(n)$
$\mathbf{w}_{2j}$
Neurons r in set $\mathscr{A}$
Σ
$\delta_j(n)$
$\varphi'(v_j n)$
$\mathbf{w}_{rj}$
$\boldsymbol{\Delta}_r(n)$

FIGURE 13.17 Back propagation of local gradients through a distributed TLFN.

Summary

- Output Layer:

$$\delta_j(n) = e_j(n)\varphi_j^{'}(n)$$

$$\mathbf{w}_{ji}(n+1) = \mathbf{w}_{ji}(n) + \eta\delta_j(n)\mathbf{x}_j(n)$$

- **Hidden Layer:**

$$\delta_j(n-lp) = \varphi_j^{'}(\upsilon_j(n-lp))\sum_{r\in A}\Delta_r^{T}(n-lp)\mathbf{w}_{rj}$$

$$\mathbf{w}_{ji}(n+1) - \mathbf{w}_{ji}(n) + \eta\delta_j(n-lp)\mathbf{x}_i(n-lp)$$

6.2. Recurrent Neural Networks

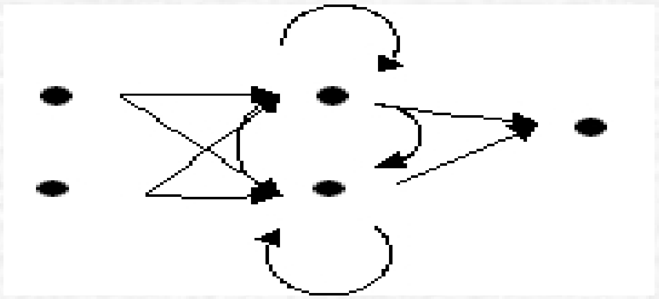
- There are two candidates for temporal processing

1. Time Lagged Neural Networks (Focused or Distributed)
2. Recurrent (Feedback) Neural Networks

- ➢ Local (Activation, Synapse, Output)
- ➢ Global

Recurrent Networks

- A network is called recurrent when there are connections which feedback to previous layers or neurons, including self-connections. An example is shown next:



- Successful early models of recurrent networks are:
    - Jordan Network
    - Elman Network

Local Feedbacks

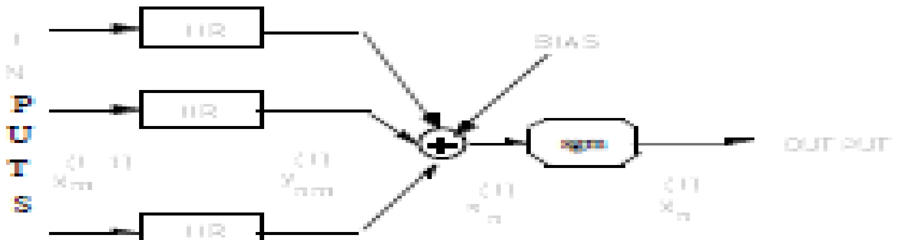


Fig. 1 Model of the neuron for the IIR MLP.

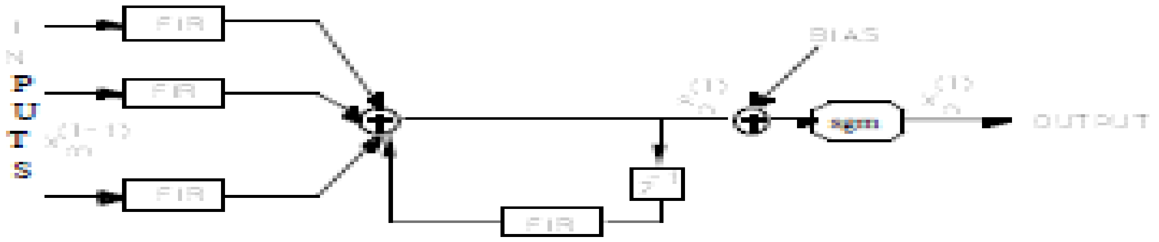


Fig. 2 Model of the neuron for the locally recurrent, activation feedback MLP.

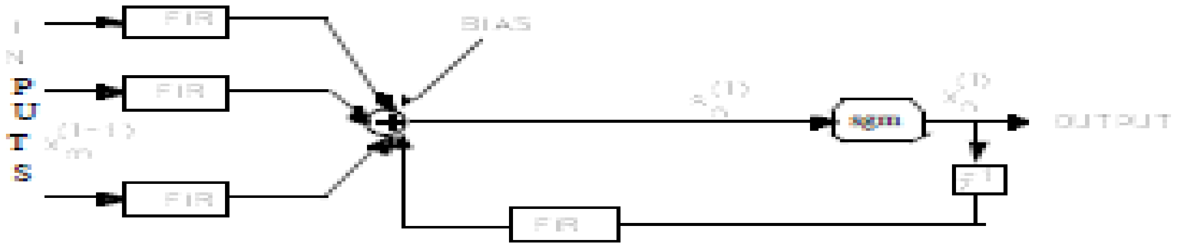


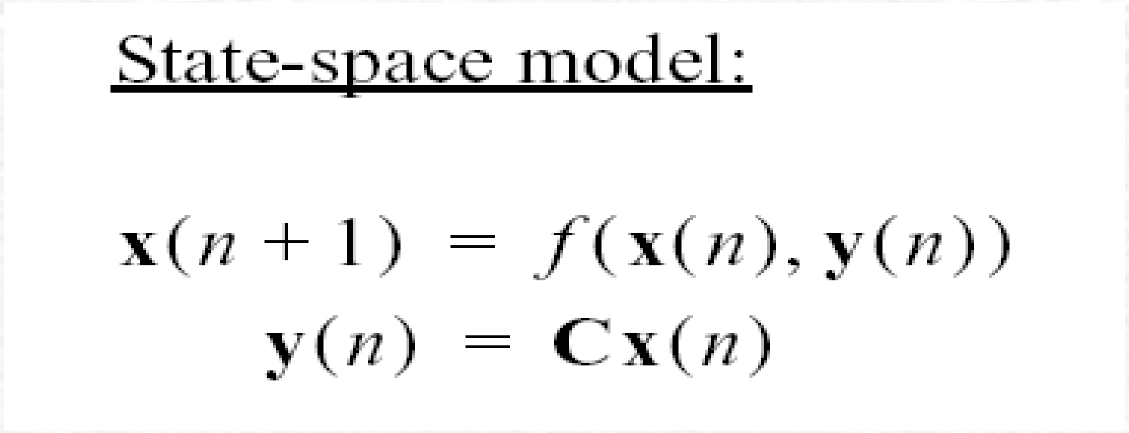
Fig. 3 Model of the neuron for the locally recurrent, output feedback MLP.

Learning Algorithms

- BPTT- Backpropagation Through Time (Werbos, Williams and Peng)
- RTRL- Real Time Recurrent Learning (Williams and Zipser, Lefebvre, Wan)
- DEKF- (Decoupled) Extended Kalman Filter (Puskorius and Feldkamp)

Real Time Recurrent Learning

- NARX model – Globally Recurrent Network
- y(n+1)= F(y(n),...,y(n-q),u(n),...,u(n-m))



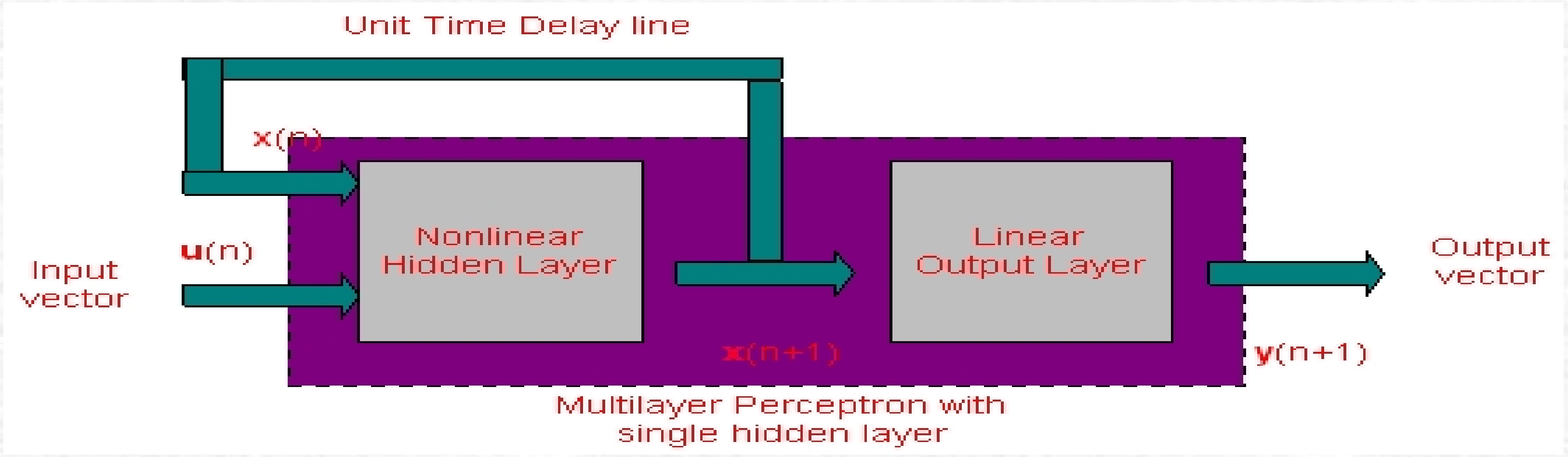
State-space model:

$$\mathbf{x}(n+1) = f(\mathbf{x}(n), \mathbf{y}(n))$$
$$\mathbf{y}(n) = \mathbf{C}\mathbf{x}(n)$$

- The output layer is linear (C is the matrix of synapsis)

- Where **f** is a suitable nonlinear function characterising the hidden layer. **x** is the state vector, as it is produced by the hidden layer. It has q components. **y** is the output vector and it has p components. The input vector is given by **u** and it has m components.

- A schematic representation of the network is given below:



Recurrent Multilayer Perceptron

- $X_I(n+1) = \varphi_I(X_I(n), u(n))$ First Layer

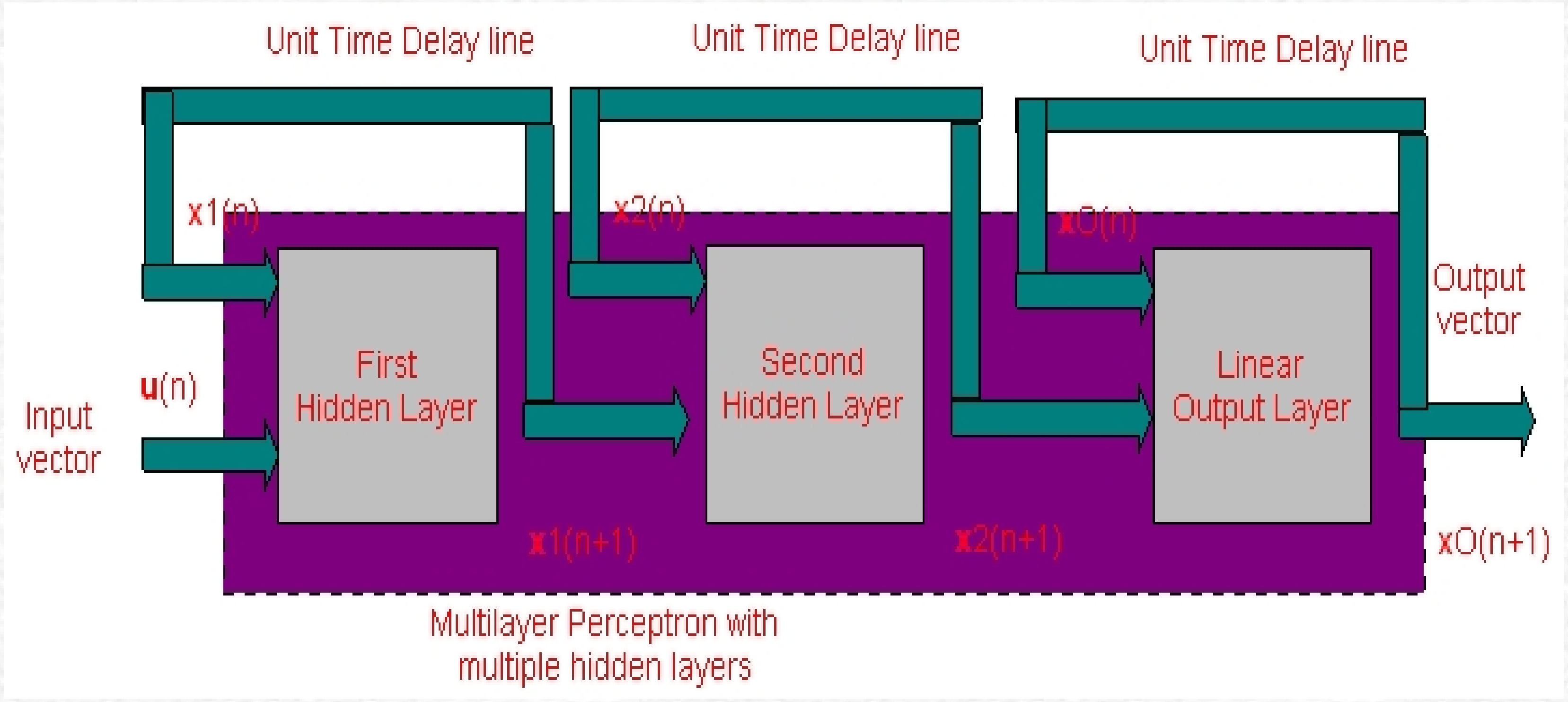
  $X_{II}(n+1) = \varphi_{II}(X_{II}(n), X_I(n+1))$ Second Layer

  ....

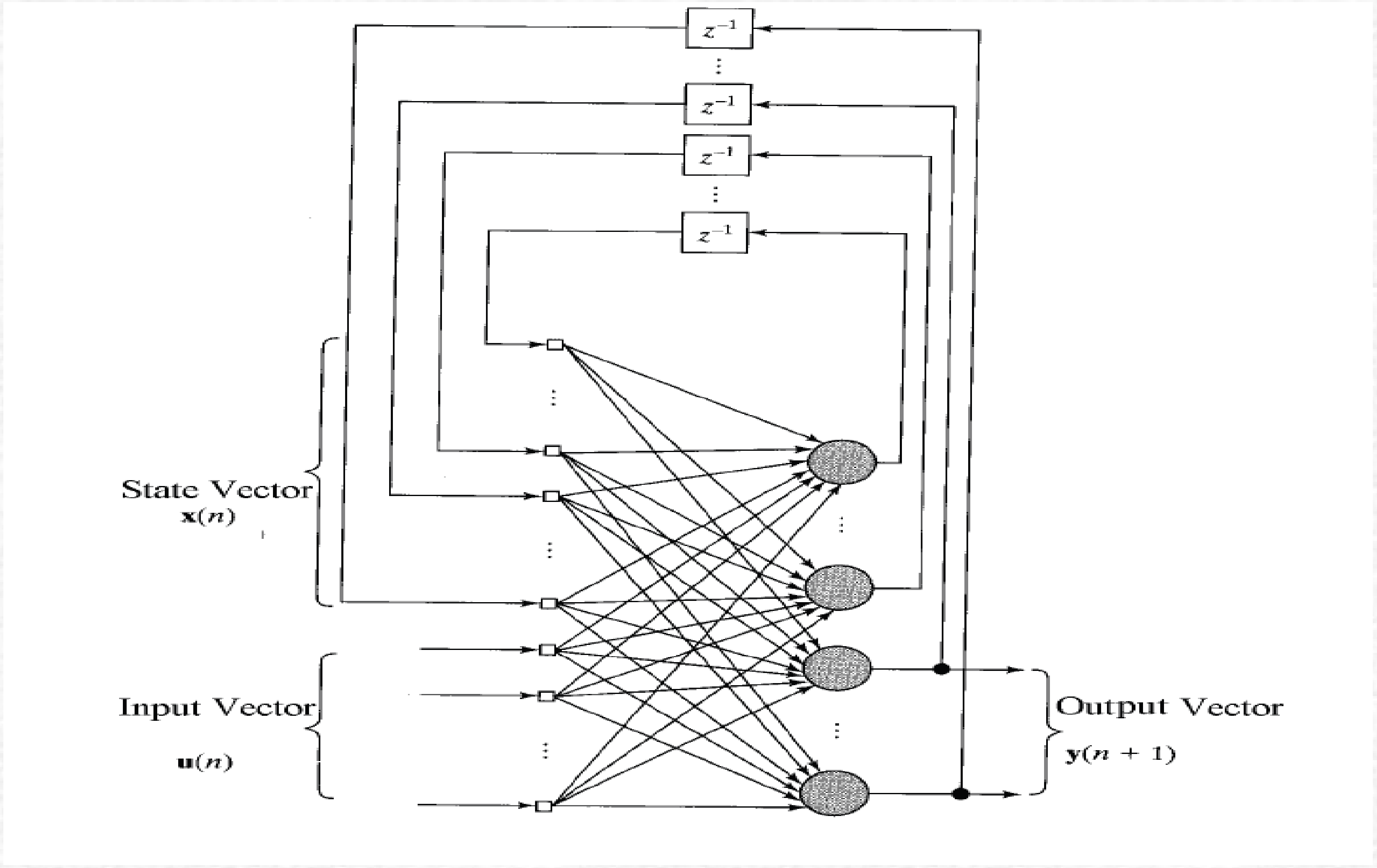
  $X_o(n+1) = \varphi_o(X_o(n), X_k(n+1))$ Output layer

$\varphi_I(.,.)$ denotes activation functions characterizing the layers (input, hiddens,output) and K denotes the number of hidden layers

- Where the functions $\phi_I(\bullet)$, $\phi_{II}(\bullet)$ and $\phi_O(\bullet)$ denote the
- Activation functions of the corresponding layer.
- A schematic representation is given below:



Fully Connected Recurrent Network



The network has two distinct layers: a concatenated input-feedback layer, and a processing layer of computation nodes

$$\mathbf{x}(n+1) = \varphi(\mathbf{W}_a\mathbf{x}(n) + \mathbf{W}_b\mathbf{u}(n))$$

$$\mathbf{y}(n) = \mathbf{C}\mathbf{x}(n)$$

$$\varphi: \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \rightarrow \begin{bmatrix} \varphi(x_1) \\ \varphi(x_2) \\ \vdots \\ \varphi(x_q) \end{bmatrix}$$

$\mathbf{w}_j = [\ \mathbf{w}_{a,j}\ \ \mathbf{w}_{b,j}\ ]'_{q+m+1,1}$ $j=1,\ldots,q$

Three new matrices

- $\xi(n)=[\mathbf{x}(n)_{q,1}\ \mathbf{u}(n)_{m+1,1}]'$
- $\Lambda_j = \partial \mathbf{x}(n)/\partial \mathbf{w}_j$
- $\mathbf{U}_j(n)=[\ 0.._,\ \xi(n)..0\ ]'_{q,\ q+m+1}$ (j th raw)

$$\Phi(n)=\mathrm{diag}(\varphi'(\mathbf{w}_1^T\xi(n))\ldots\ \varphi'(\mathbf{w}_j^T\xi(n))\ldots\ \varphi'(\mathbf{w}_q^T\xi(n)))$$

$$\Lambda_j\,(n+1) = \Phi(n)[\mathbf{W}_a\,(n)\,\Lambda_j\,(n) + \mathbf{U}_j(n)]$$

- This recursive equation describes the *nonlinear state dynamics*

Approximation to the method of steepest descent

- $\mathbf{e}(n)=\mathbf{d}(n)-\mathbf{y}(n)=\mathbf{d}(n)-C\mathbf{x}(n)$
- $\partial\varepsilon(n)/\partial\mathbf{w}_j=(\partial\mathbf{e}(n)/\partial\mathbf{w}_j)\mathbf{e}(n)$

  $=-C(\partial\mathbf{x}(n)/\partial\mathbf{w}_j)\mathbf{e}(n) \quad j=1,..,q$
- $\Lambda_j = \partial\mathbf{x}(n)/\partial\mathbf{w}_j$

$\Delta \mathbf{w}_j = -\eta \partial \varepsilon(n) / \partial \mathbf{w}_j = -\eta \, C \, \Lambda_j(n) \, \mathbf{e}(n)$

$\Lambda_j (n) = \Phi(n-1)[\mathbf{W}_a (n-1) \Lambda_j (n-1) + \mathbf{U}_j(n-1)]$

$\Lambda_j (n) = 0$ for all j initially