CISC/CMPE452/COGS 400
Supervised Learning
Other Approaches

Ch. 4 - Text book

Farhana Zulkernine

Shortcomings of BPN

- What is the next letter in the sequence?:

  O

  T

  T

  F

  F

  S

  S

  ?

BPN is not good for sequence prediction or learning.

Shortcomings of BPN

Consider the following learning problem:

Training Data		Test data
101101	class 1	111101
011010	class 2	001100
101011	class 1	
110001	class 2	
001001	class 1	
111010	class 1	
100110	class 2	

- To what classes do the test data belong?
- At some point you stop trying to relate the individual locations to the classes and look for some other means of making the relation. Backpropagation cannot make that switch.

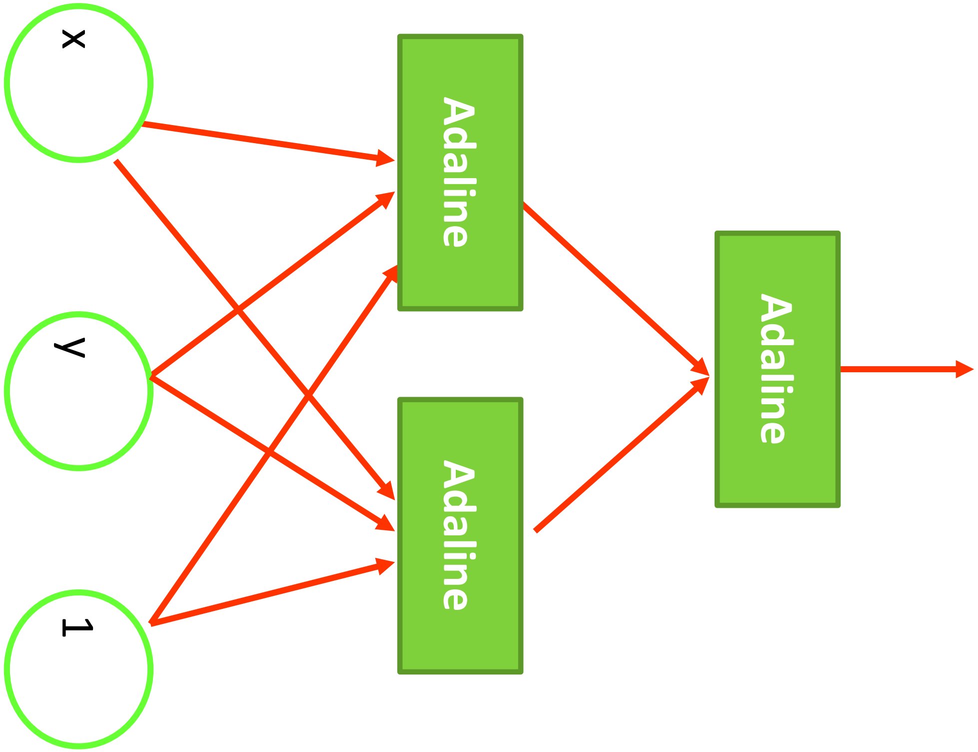
Other Networks

- Madaline
- Adaptive Multilayer Networks
- Recurrent Networks

Madaline

- Combination of many adalines.
- Uses least MSE (desired – actual) output for error correction as in the Adaline.
- Follows **“minimum disturbance”** principle for learning.
  - Only changes the weights to nodes whose net inputs are smaller than a threshold.
  - Examines the result of changing the output of such a node – look forward.
  - If a “change” results in a decrease in ANN error, ONLY then weights leading into that node are changed
- Can be multilayered with hidden and output nodes as adalines which are trained using various *Madaline Rule* training algorithm.
  - 3 versions described in the book.

Madaline



Size of NN matters: Why?

- Smaller networks are more desirable than larger ones doing the same job.
  - Faster training
  - Fewer parameters
  - Fewer training samples are required
  - Likely to generalize well for new test samples

Adaptive Multilayer Networks

- Three approaches to build a network of "optimal" size that are
  - based on sound heuristics that have been ***shown empirically to work***.
  - But none is guaranteed to result in optimal size.

1. (–) A large network may be built and then **“pruned"** by eliminating nodes and connections that can be considered unimportant.
2. (+) Starting with a very small network, the size of the network is repeatedly increased by small increments until performance is satisfactory.

Adaptive Multilayer (cont…)

3. (−+) A *sufficiently* large network is trained, and unimportant connections and nodes are then pruned, following which new nodes with random weights are re-introduced and the network is retrained.

Pruning continues until a network of acceptable size and performance level is obtained, or further pruning attempts become unsuccessful.

NN Pruning

- Pruning a connection corresponds to changing the connection weight from w to 0,

  i.e., $\Delta w = -w$

Train a network large enough to solve the problem at hand;
repeat
    Find a node or connection whose removal does not
        penalize performance beyond desirable tolerance levels;
    Delete this node or connection;
    (Optional:) Retrain the resulting network
until further pruning degrades performance excessively.

Fig. 4.1 (−) Generic network pruning algorithm.

Identify Unimportant Node/Connection

1. **Connections associated with weights of small magnitude may be eliminated** from the trained network.
2. **Connections whose existence does not significantly affect network outputs (or error) may be pruned (abnormal feature)**. These may be detected by
   - Examining the change in network output when a connection weight is changed to 0 or $\Delta w = -w$.
   - Testing whether $\partial y / \partial w$ is negligible.

Disadvantage

- Takes time to train a network.
- Takes time to train then prune the network.
- If the weights have already converged then pruning nodes or connections results in significant degradation in performance.

Adaptive Network Pruning

- As an alternative to first training large networks and then pruning them, several network pruning algorithms have been proposed that ***adaptively*** build up larger networks from smaller ones.
  - **Marchand’s algorithm**
  - **Upstart algorithm**
  - **Neural Tree**
  - **Cascade Correlation**
  - **Tiling Algorithm**

Marchand's Algorithm

- Marchand's algorithm obtains an "optimal" size network for classification problems, repeatedly adding a perceptron node to the hidden layer.
  - Example: Feedforward NN with one hidden layer where perceptron nodes are added repeatedly.

let T_k^+ and T_k^- represent the nonempty sets of training samples of two classes that remain to be correctly classified, a new node is added, whose weights are trained such that either $|T_{k+1}^+| < |T_k^+|$ or $|T_{k+1}^-| < |T_k^-|$, and $(T_{k+1}^- \cup T_{k+1}^+) \subset (T_k^- \cup T_k^+)$, ensuring that the algorithm terminates eventually at the mth step, when either T_m^- or T_m^+ is empty.

Marchand's Algorithm

Example: For a Two Class Problem

Remains to be classified properly

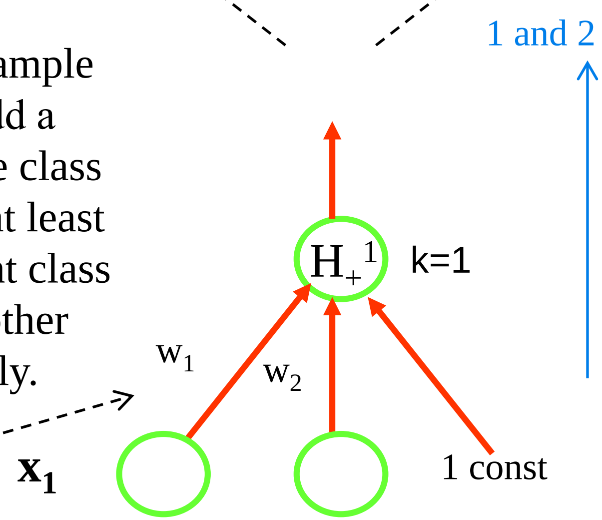
$T_{k+1}^{+} = \{\mathbf{x_2}, \mathbf{x_5}\}$ vectors

Step 2:Create T_{k+1}^{+} and T_{k+1}^{+}

$T_{k+1}^{-} = T_k^{-}$ vectors

1. At k^{th} iteration, take a sample from any class (+ or −), add a hidden node H of the same class and train network so that at least one sample from the parent class and all samples from the other class are classified correctly.

3. Add the **misclassified** samples from parent class T_k^{+} to the set T_{k+1}^{+} and all samples of the other class are moved to T_{k+1}^{-}.



Step 1: Pick a sample from T_k^{+} and add H_+^1

$T_k^{+} = \{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \ldots\}$ vectors

$T_k^{-} = \{\mathbf{y_1}, \mathbf{y_2}, \mathbf{y_3}, \ldots\}$ vectors

x and **y** are 2-dimensional input vectors of classes + and −

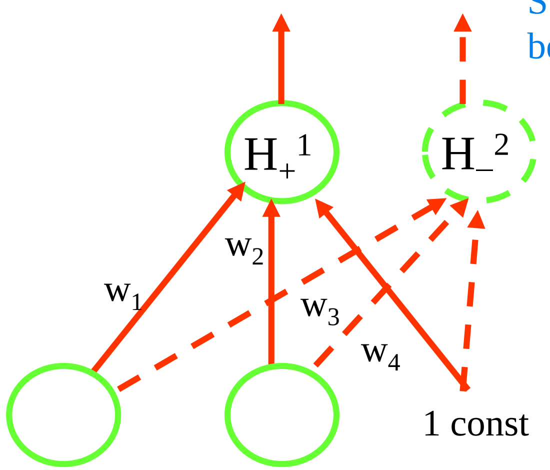
Marchand's Algorithm (cont…)

Remains to be classified properly

$T_{k+1}^{+} = T_{k}^{+}$ vectors

$T_{k+1}^{-} = \{\mathbf{y}_2, \mathbf{y}_3\}$ vectors

Step 5: Continue until one of the sets become empty



4. k = k + 1.
5. Continue steps 1 to 5 until one of the sample sets T_k^{+} or T_k^{-} becomes empty.

3 and 4

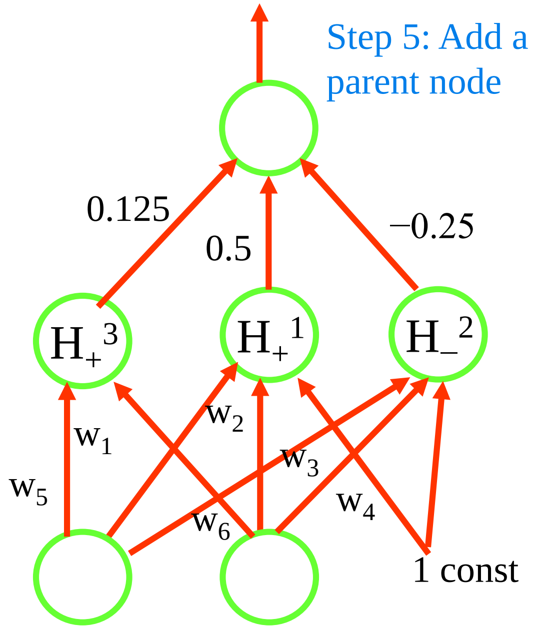
$T_{k}^{+} = \{\mathbf{x}_2, \mathbf{x}_5, \mathbf{x}_1, \ldots, \}$ vectors

Step 4: Go to next iteration

$T_{k}^{-} = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \ldots\}$ vectors

x and **y** are 2-dimensional input vectors of classes + and −

Marchand's Algorithm (cont…)



5. Add a parent output node with inputs from all H_+ and H_- such that $w_k = 1/2^k$ for all nodes in H_+ and $w_k = -1/2^k$ for all nodes in H_-. This ensures that the nodes added later do not modify the correct results obtained by earlier nodes.

$T_k^+ = \{\ \}$ vectors

$T_k^- = \{\mathbf{y}_3, \ldots\}$ vectors

x and **y** are 2-dimensional input vectors of classes + and −

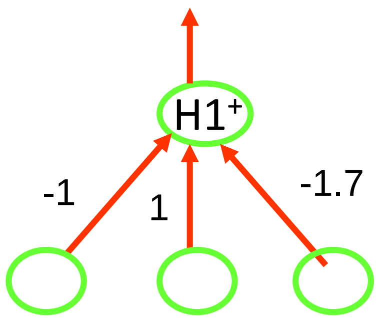
Example 4.1 – Corner Isolation Problem

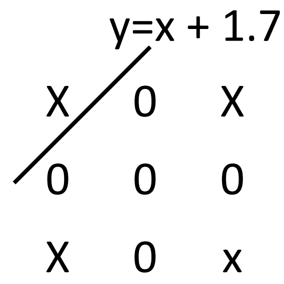
- Two dimensional input patterns ϵ $[-1, +1]^2$
- Two output classes ϵ [1, 0] that are NOT linearly separable (least no. of misclassification using Adaline = 3).
- Class I: {(-1,1), (-1,-1), (1,1), (1, -1)} = T_0^+
  - Desired output =1
- Class II: {(-1,0), (0,-1), (0,1), (1, 0), (0, 0)} = T_0^-
  - Desired output = 0

Adding Hidden Nodes

Corner Isolation Problem

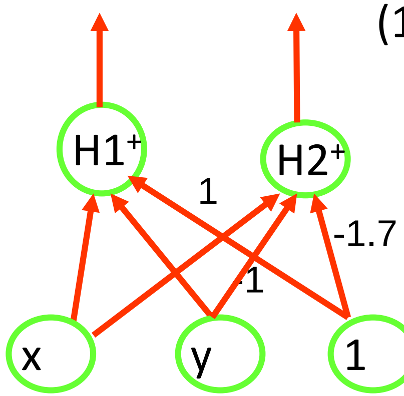
(-1,1)

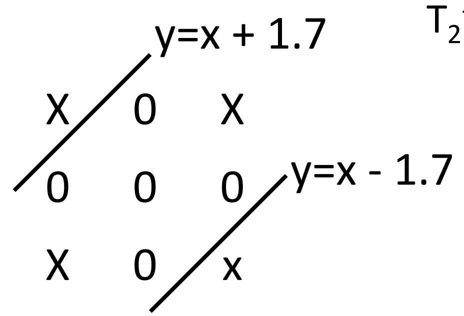




$T_1^+ = \{(-1,-1), (1,1), (1, -1)\}$
$T_1^- = \{(-1,0), (0,-1), (0,1), (1, 0), (0, 0)\}$

(1, -1)

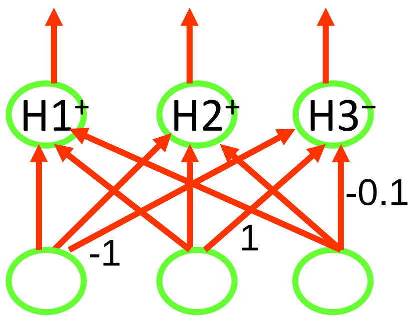


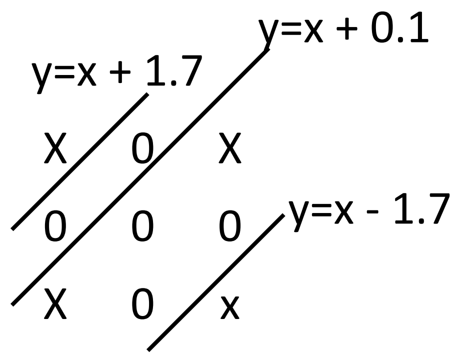


$T_2^+ = \{(-1,-1), (1,1)\}$
$T_2^- = T_1^- = \{(-1,0), (0,-1), (0,1), (1, 0), (0, 0)\}$

Adding Hidden Nodes

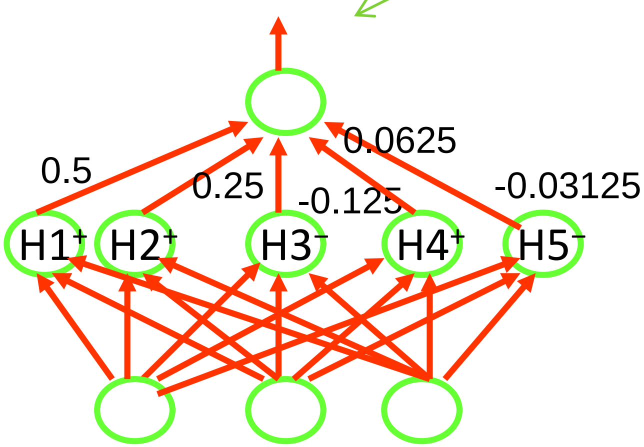
(-1,0), (0,-1)

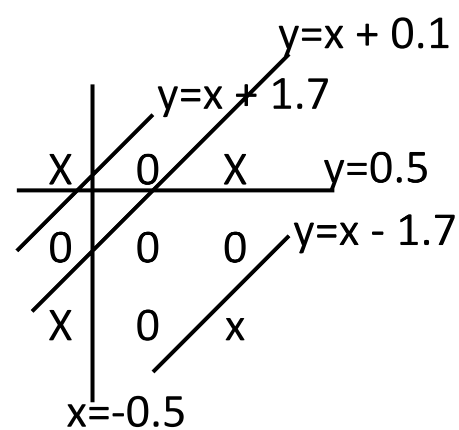




$T_3^+ = T_2^+ = \{(-1,-1), (1,1)\}$
$T_3^- = \{(0,-1), (1, 0), (0, 0)\}$

Could be designed using 4 nodes – not optimum





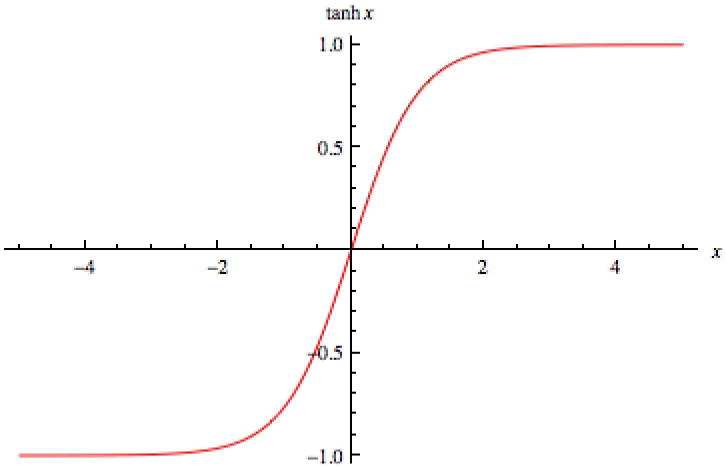
Cascade Correlation

- This algorithm has two important features:
  (1) the cascade architecture development, and
  (2) correlation learning.
- The architecture is not strictly feedforward. New **single-node hidden layers** are successively added to a *steadily growing layered* neural network in *between output and previous hidden layer* until performance is judged adequate.
- Each node may employ a nonlinear node function such as the hyperbolic tangent, whose output lies in the closed interval $[-1.0, 1.0]$.

Hyperbolic Tangent

- Tangent $\tan z \equiv \sin z / \cos z$
- Hyperbolic tangent $\tanh z \equiv \sinh z / \cosh z$

$$\equiv (e^{z} - e^{-z}) / (e^{z} + e^{-z})$$



Training in Cascade Corr. Network

- Fahlman and Lebiere suggest using the Quickprop learning algorithm.
- When a node is added, its input weights are trained first.
- Then all the weights on the connections to the output layer are trained while leaving other weights unchanged.
- Weights to each new hidden node are trained to maximize covariance with current network error.

CC Network

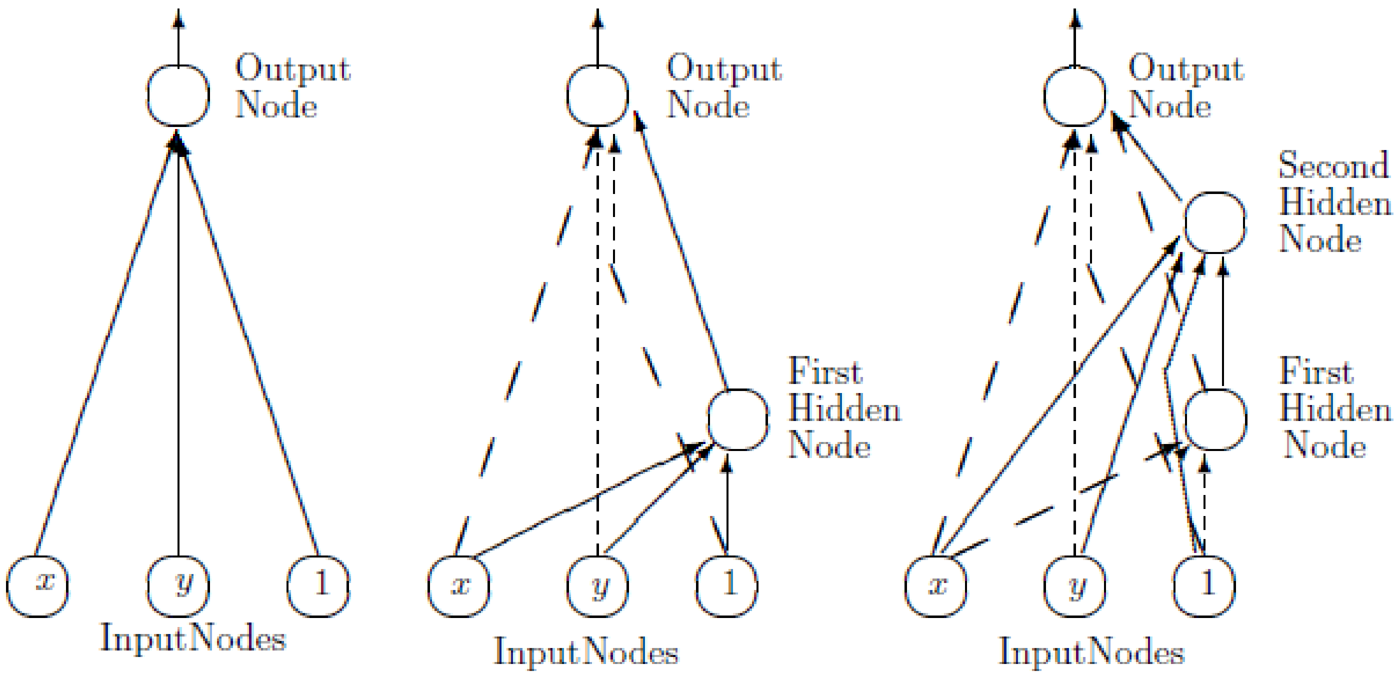


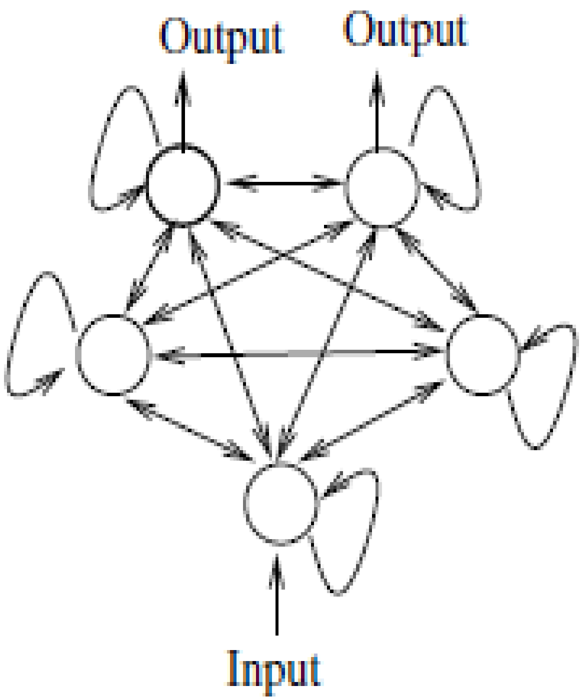
Fig: Cascade network applied to the corner isolation problem. Solid lines show the node being added.

Prediction Networks

- Prediction problems constitute a special subclass of function approximation problems, in which the **values of variables need to be determined from values at previous instants**.
- Two classes of neural networks have been used for prediction tasks:
  - Recurrent networks and
  - Feedforward networks.

Recurrent Networks

- Recurrent neural networks contain **connections from output nodes to hidden layer and/or input layer** nodes, and they allow **interconnections between nodes of the same layer**, particularly between the nodes of hidden layers.
- All biological neural networks are recurrent.



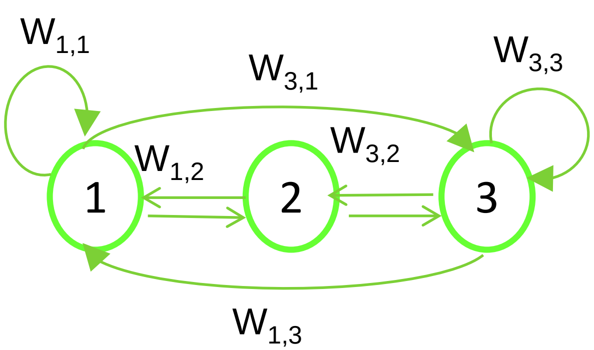
Recurrent Networks (cont…)

- Rumelhart, Hinton, and Williams (1986) view recurrent networks as feedforward networks with a large number of layers.
- **Each layer is thought of as representing a time delay** in the network.
- Each node is connected to all other nodes in the same layer and in the next layer (time sequence).

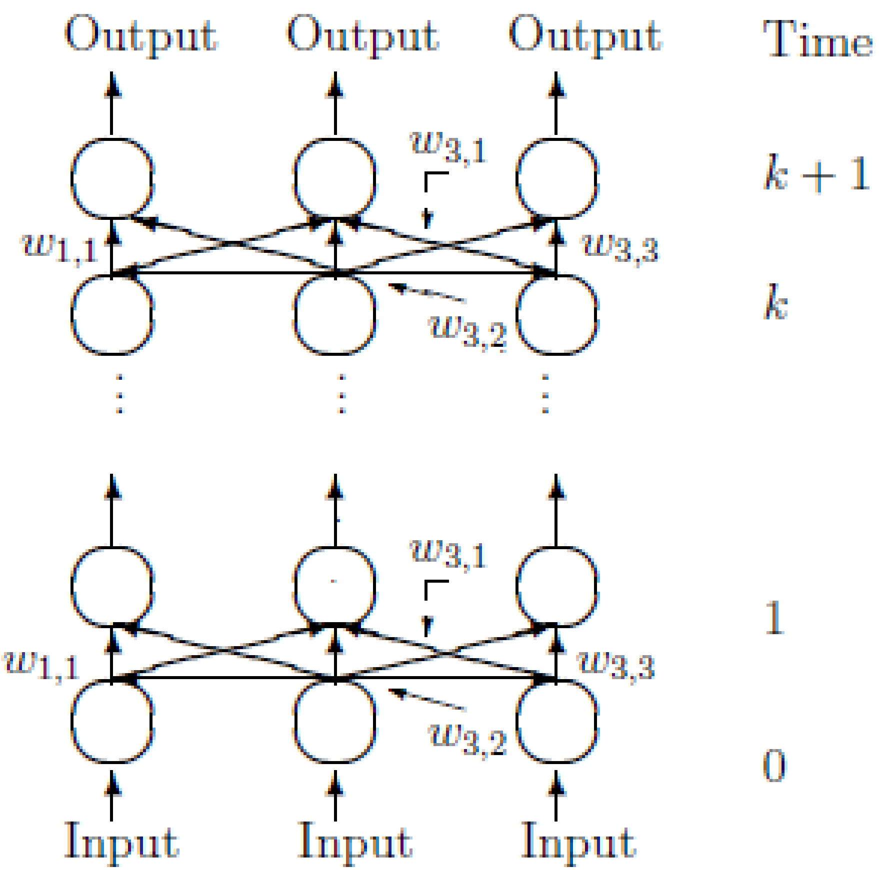
Training & Applications of RNN

- Applications
  - RNNs can approximate arbitrary dynamical systems with arbitrary precision.
  - Pattern recognition, temporal prediction
- Training
  - Applies both supervised and unsupervised learning.
    - Supervised learning is used for prediction.
    - Unsupervised learning is used for associative memory models, pattern approximation.
  - Many variations of training algorithms are used with RNN.

Rumelhart's Recurrent Network



(a)



(b)

(a) Fully connected recurrent neural network with 3 nodes
(b) Equivalent feedforward version for Rumelhart's training procedure.

Recurrent Networks (cont…)

- Their training procedure is essentially the same as the backpropagation algorithm.
- Using this approach, the fully connected neural network with three nodes is considered equivalent to a feedforward neural network with *k* hidden layers.
- Weights in different layers are constrained to be identical, to capture the structure of a recurrent network: $w_{ji}^{(l,\, l\text{-}1)} = w_{ij}^{(l\text{-}1,\, l\text{-}2)}$

Williams and Zipser's Approach

- Another training procedure for a recurrent network with hidden nodes, proposed by Williams and Zipser (1989), differs from backpropagation.
- The net input to the k^{th} node consists of the inputs from other nodes (o) as well as external inputs (i).

$$net_k(t) = \sum_{l \in U} w_{kl} o_l(t) + \sum_{l \in I} w_{kl} i_l(t) = \sum_{l \in U \cup I} w_{kl} z_l(t) \quad \text{.....(1)}$$

- U is the set of internal input nodes and I is the set of external input nodes.

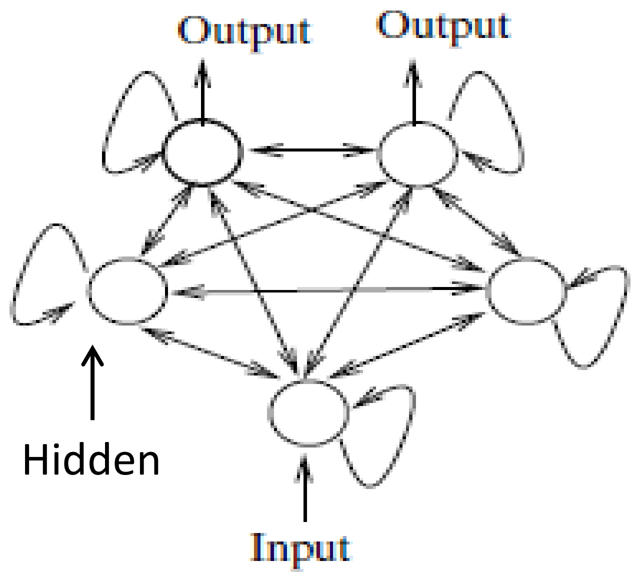
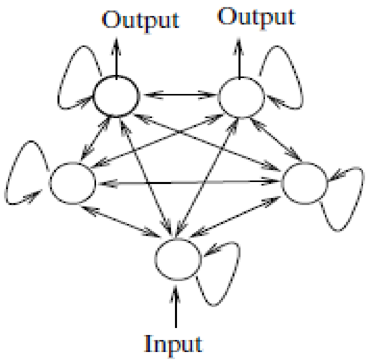


Figure : Recurrent network with hidden nodes, to which Williams and Zipser's training procedure can be applied.

Williams-Zipser's (cont…)

- Error $E(t) = \sum_k (d_k(t) - o_k(t))^2 = \sum_k e_k(t)^2$ …(2)



- The training algorithm uses the same gradient descent learning.

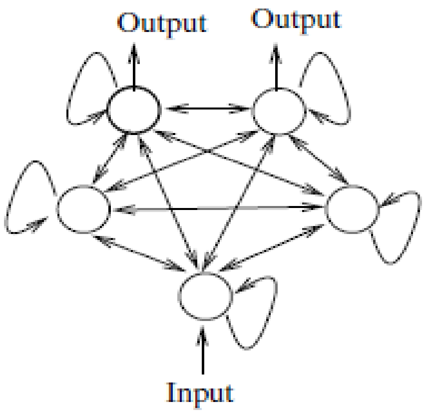
  Therefore, $\Delta w_{ji}(t) = -\eta\, (\partial E(t)/\partial w_{ji})$ …(3)

- The output of each node (k) at time (t+1) is a function of net input to k) at the previous instant t and it depends on outputs of other nodes:

$$o_k(t+1) = f(net_k(t)) \qquad \text{… (4)}$$

Williams-Zipser's (cont…)

- Therefore, using (1) to (4),



$$\Delta w_{ji}(t) = -\eta\ (\partial E(t)/\partial w_{ji})$$

$$= -\eta\ (\partial/\partial w_{ji})\ \sum_{k\epsilon U}(d_k(t) - o_k(t))^2$$

$$= \eta \sum_{k\epsilon U} (d_k(t) - o_k(t))\ \partial o_k(t)/\partial w_{ji}$$

(2 included in η)

Partial derivative of output,

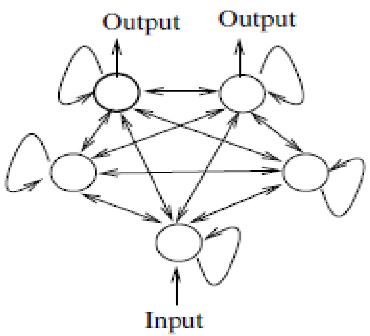
$$\partial o_k(t+1)/\partial w_{ji} = \partial/\partial w_{ji}\, f(net_k(t))$$

$$= f'(net_k(t))\ \partial/\partial w_{ji}\ (\sum_{l\epsilon U} w_{kl} z_l(t))$$

For j=k and *l*=i

$$= f'(net_k(t))\ [\ \sum_{l\epsilon U} w_{kl} \partial z_l(t)/\partial w_{ji} + \delta_{jk} z_i(t)\ ]$$

Williams-Zipser's (cont…)



- δ_{jk} is called the Kronecker delta with
  $\delta_{jk} = 1$ if $j=k$ and 0 otherwise and
  $\partial o_k(t_0)/\partial w_{ji} = 0$ since we assume that the initial state of the network has no functional dependence on the weights.

  If sigmoid function is used as output function then,

  $f'(net_k(t)) = o_k(t+1)\,[\,1 - o_k(t+1)\,]$

  for all $k \in U$, $i \in U$, $j \in U \cup I$, and $t >= t_0$

Algorithm

Figure 4.5 Williams and Zipser's Recurrent network training algorithm

Assume randomly chosen weights, $t = 0$, and

$$\frac{\partial o_k(0)}{\partial w_{i,j}} = 0, \; \texttt{for each } i, j, k.$$

while MSE is unsatisfactory and computational bounds are not exceeded **do**

Modify the weighs:

$$\Delta w_{i,j}(t) = \eta \sum_{k \in U} (d_k(t) - o_k(t)) \frac{\partial o_k(t)}{\partial w_{i,j}}$$

where U is the set of nodes with a specified target values $d_k(t)$

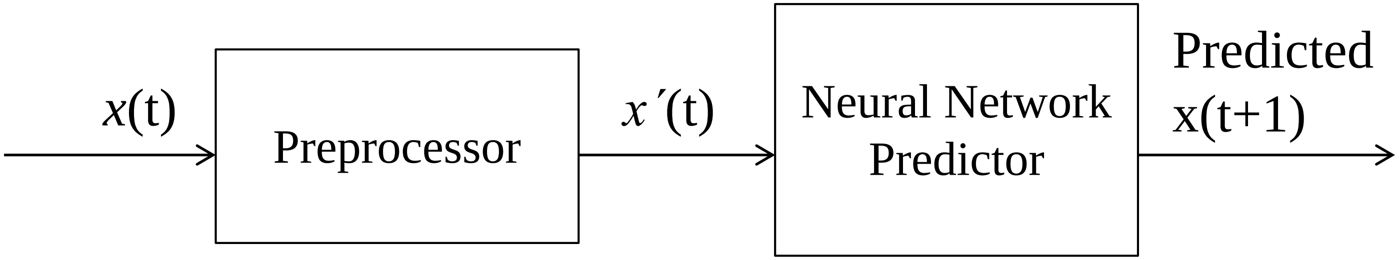
For next iteration compute $\partial o_k(t+1) / \partial w_{ji}$

Increment t

end while

Feedforward Networks for Forecasting

- The generic network model consists of a preliminary preprocessing component that transforms an external input vector x(t) into a preprocessed vector x'(t). The feedforward network is trained to compute the desired output values for a specific input x'(t).



Generic neural network model for prediction

Tapped Delay-line Neural Network (TDNN)

- Consider that x(t) is to be predicted from x(t -1), x(t - 2).
- In a simple case, x at time t consists of a single input x(t), and x´ at time t consists of the vector (x(t), x(t - 1), x(t - 2)) supplied as input to the feedforward network.
- For this example, preprocessing consists merely of storing past values of the variable and supplying them to the network along with the latest value. Such a model is sometimes called a Tapped Delay-line Neural Network (TDNN),

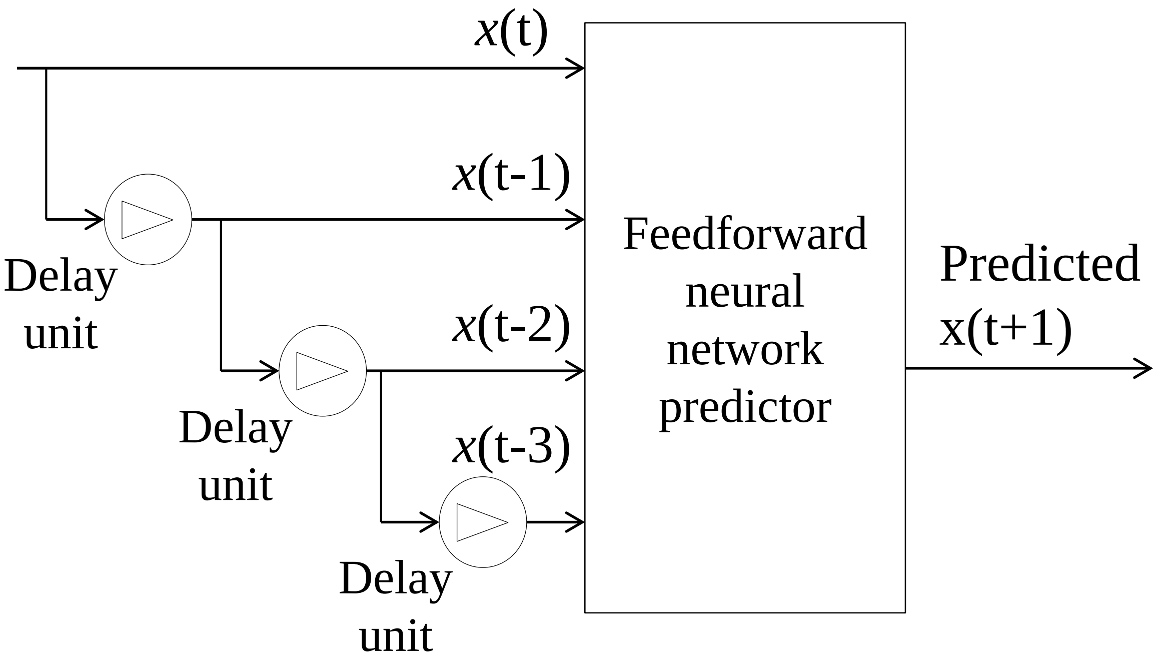
TDNN

Many preprocessing transformations for prediction problems can be described as convolution of the input sequence with a kernel function c_i which can vary for different applications.

$$x'(t) = \sum_{\tau=0}^{t} c_i(t-\tau)x(\tau)$$

For example,
for discrete time delay,

$$c_i = \begin{cases} 1 \text{ for } j=i \\ 0 \text{ otherwise} \end{cases}$$



Regularization

- Many of the NN algorithms apply regularization.
- Regularization: *Optimization of a cost function*.
- Can be expressed as: $E + \lambda|P|^2$ where E is the original cost (or error) function, P is a "stabilizer" that incorporates a priori problem-specific requirements of constraints, and λ is a constant that controls the relative importance of E and P.

Explicit and Implicit Regularization

- Can be implemented explicitly by introducing $P = \lambda \sum_j w_j^2$ in algorithms into the cost function being minimized (to penalize large weights).
  - A weight decay term may be used which favours the development of networks with smaller weight magnitudes.

    $\Delta w = -\eta\ (\partial E / \partial w) - \lambda w$
  - Smoothing penalties are used to prevent very high curvature in the output function and thus over-specializing on training data to account for outliers where $P = |\partial^2 E / \partial w_i^2|$.
- Implicit regularization is used for example, by introducing random noise in training data or connection weights (equivalent to imposing a smoothness constraint on the derivative of the squared error function with respect to input or weights).

Summary

- Backpropagation algorithm cannot address temporal prediction or classification when sufficient match is not available.
- Madaline is used to minimize change by using look ahead technique.
- Pruning is used to modify existing network size to have a more optimal size network.
- Adaptive NN used to *create* optimal size networks. Several algorithms exist.
- Recurrent and feedforward networks are better suited for temporal predictions.