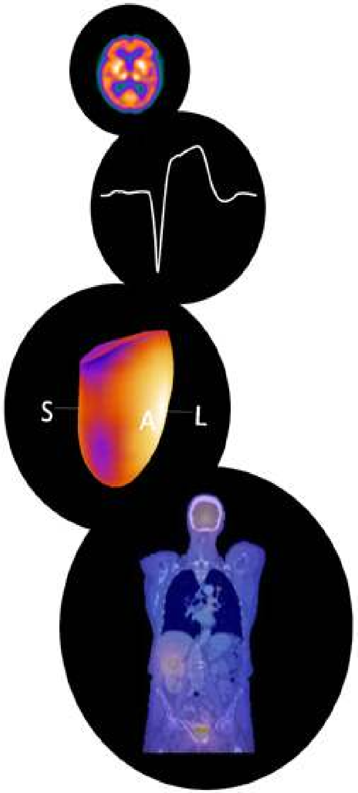


METU Informatics Institute
Min720

Pattern Classification with Bio-Medical Applications

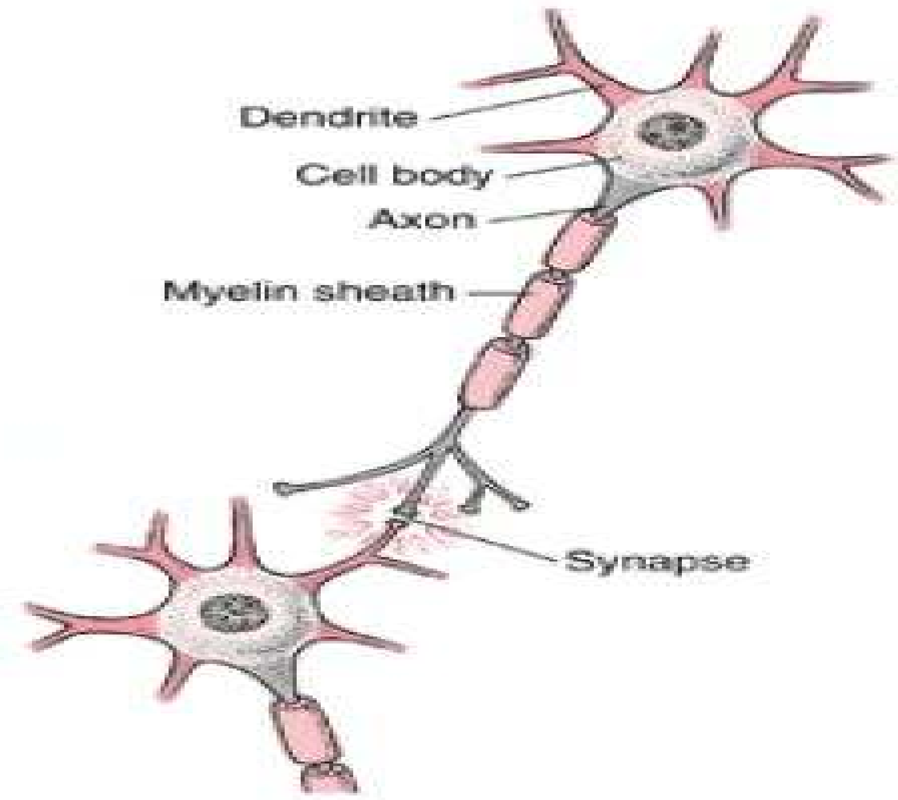
Part 8: Neural Networks

1- INTRODUCTION: BIOLOGICAL VS. ARTIFICIAL

Biological Neural Networks

A Neuron:

- A nerve cell as a part of nervous system and the brain



(Figure: http://hubpages.com/hub/Self-Affirmations)

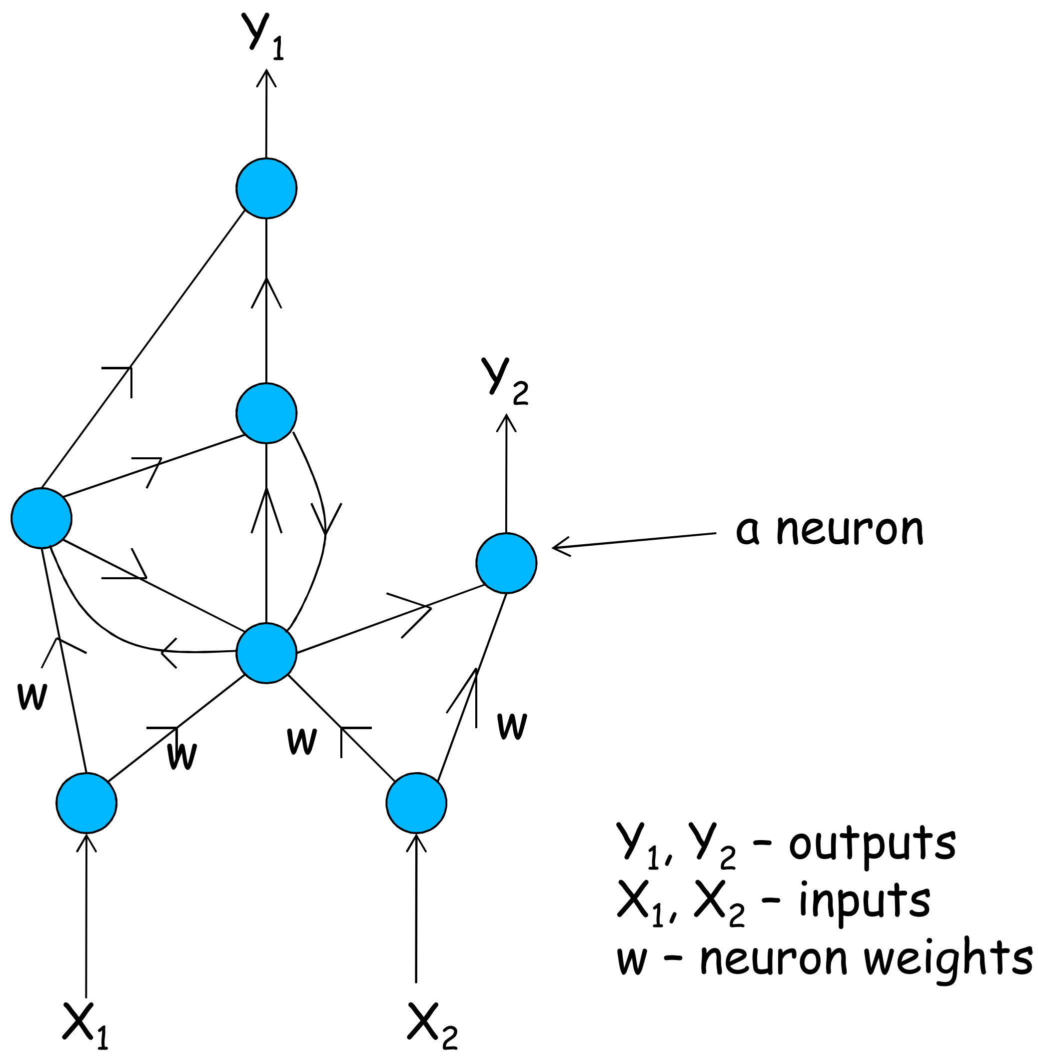
Biological Neural Networks

- There are 10 billion neurons in human brain.
- A huge number of connections
- All tasks such as thinking, reasoning, learning and recognition are performed by the information storage and transfer between neurons
- Each neuron "fires" sufficient amount of electric impulse is received from other neurons.
- The information is transferred through successive firings of many neurons through the network of neurons.

Artificial Neural Networks

An artificial NN, or ANN or (a connectionist model, a neuromorphic system) is meant to be

- A simple, computational model of the biological NN.
- A simulation of above model in solving problems in pattern recognition, optimization etc.
- A parallel architecture of simple processing elements connected densely.



An Artificial Neural Net

Any application that involves

- Classification
- Optimization
- Clustering
- Scheduling
- Feature Extraction

may use ANN!

Most of the time integrated with other methods such as

- Expert systems
- Markov models

WHY ANN?

- Easy to implement
- Self learning ability
- When parallel architectures are used, very fast.
- Performance at least as good as other approaches, in principle they provide nonlinear discriminants, so solve any P.R. problem.
- Many boards and software available

APPLICATION AREAS:

- Character Recognition
- Speech Recognition
- Texture Segmentation
- Biomedical Problems (Diagnosis)
- Signal and Image Processing (Compression)
- Business (Accounting, Marketing, Financial Analysis)

Background: Pioneering Work

1940 - McCulloch-Pitts (Earliest NN models)

1950 - Hebb- (Learning rule, 1949)
- Rosenblatt(Perceptrons, 1958-62)

1960 -Widrow, Hoff (Adaline, 1960)

1970

1980 - Hopfield, Tamk (Hopfield Model, 1985)
- RumelHart, McClelland (Backpropagation)
- Kohnen (Kohonen's nets, 1986)

1990 - Grossberg, Carpenter (ART)

90's Higher order NN's, time-delay NN, recurrent NN's, radial basis function NN
-Applications in Finance, Engineering, etc.
- Well-accepted method of classification and optimization.

2000's Becoming a bit outdated.

ANN Models:

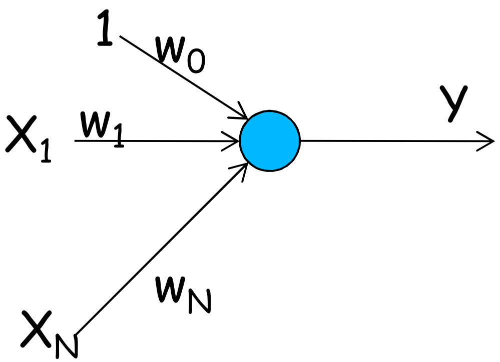
Can be examined in

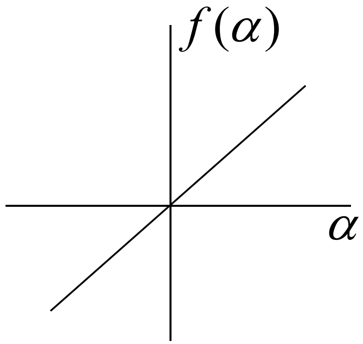
1- Single Neuron Model

2-Topology

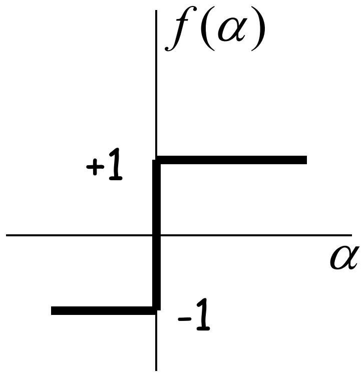
3- Learning

1- Single Neuron Model:

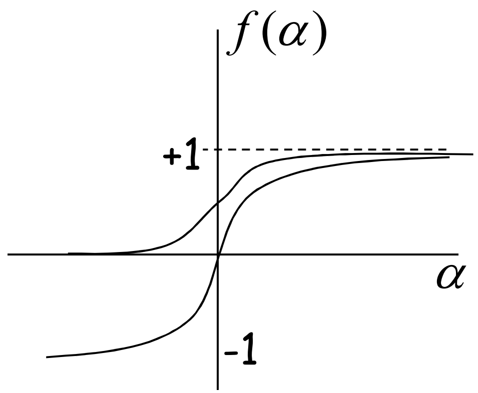




linear



Step(bipolar)



Sigmoid

General Model:

$$Y = f(\overbrace{\sum_{i=1}^{N} w_i x_i + w_o}^{\alpha}) = f(\alpha) = f(net)$$

$f(\alpha)$ - Activation function

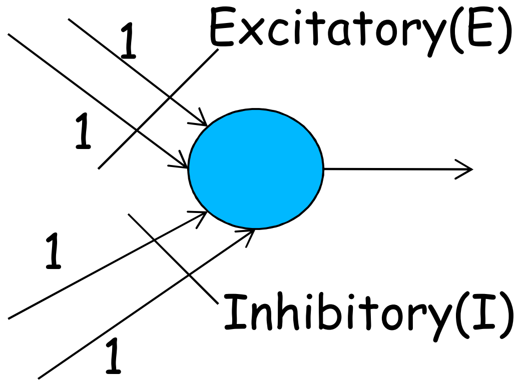
Binary threshold / Bipolar / Hardlimiter

Sigmoid $\quad f(\alpha) = 1/(1+\exp(-\alpha d))$

When d=1, $\quad \frac{df}{d\alpha} = f(1-f)$

Mc Culloch-Pitts Neuron:

- Binary Activation
- All weights of positive activations and negative activations are the same.



Fires only if

$E \geq T, I = 0$

where

$E = \sum excit.inputs$

$I = \sum inb.inputs$

T=Threshold

Higher-Order Neurons:

- The input to the threshold unit is not a linear but a multiplicative function of weights. For example, a second-order neuron has a threshold logic with

$$\alpha = \sum_{i=1}^{N} w_i x_i + w_o + \sum_{1 \le i, j \le N} w_{ij} x_i x_j$$

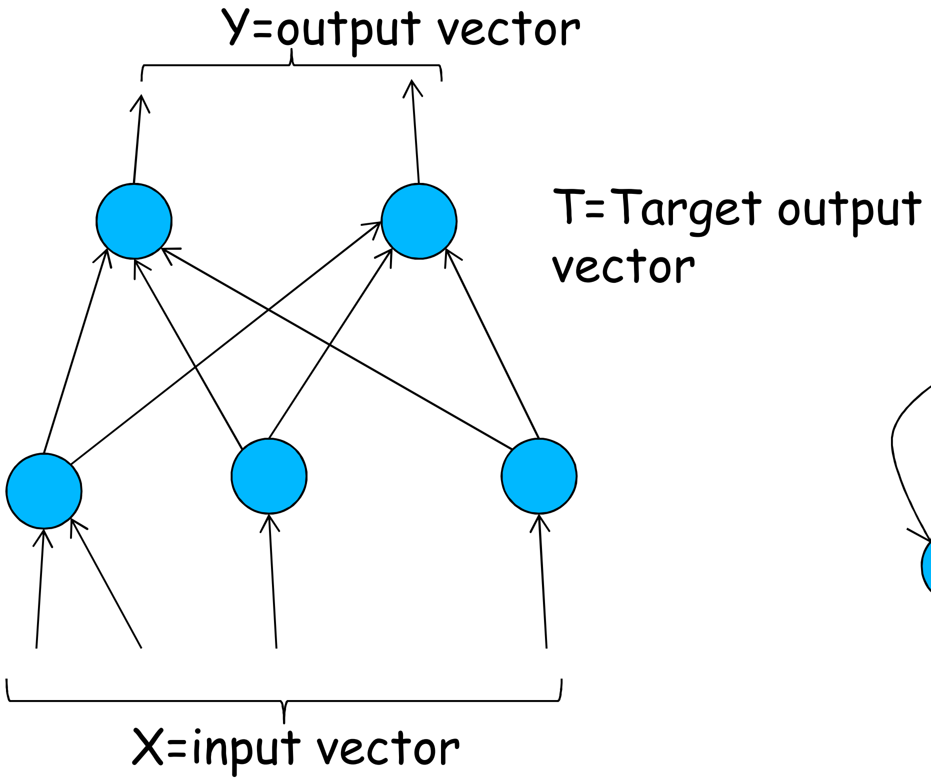
with binary inputs.

- More powerful than traditional model.

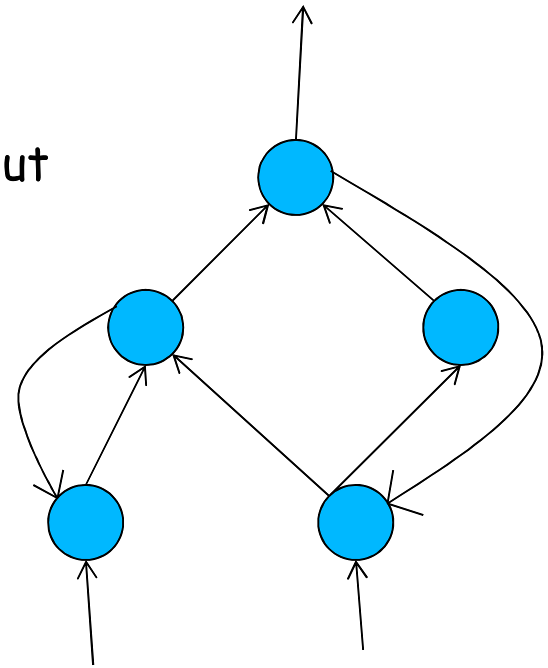
2. NN Topologies:

- 2 basic types:
- Feedforward
- Recurrent – loops allowed

- Both can be "single layer" or many successive layers.



A feed-forward net



A recurrent net

3.Learning: Means finding the weights w using the input samples so that the input-output pairs behave as desired.

supervised- samples are labeled (of known category)

P=(X,T) input-target output pair

unsupervised- samples are not labeled. Learning in general is attained by iteratively modifying the weights.

- Can be done in one step or a large no of steps.

Hebb's rule: If two interconnected neurons are both 'on' at the same time, the weight between them should be increased (by the product of the neuron values).

- Single pass over the training data
- w(new)=w(old)+xy

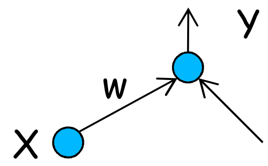
Fixed-Increment Rule (Perceptron):

- More general than Hebb's rule – iterative
- $w(new) = w(old) + \sigma xt$ (change only if error occurs.)

t – target value – assumed to be '1' (if desired), '0'(if not desired).

σ is the learning rate.

Delta Rule: Used in multilayer perceptrons. Iterative.



$$w(new) = w(old) + \sigma(t - y)x$$

- where t is the target value and the y is the obtained value. (t is assumed to be continuous)

- Assumes that the activation function is identity.

$$\Delta w = \sigma(t - y)x \approx w(new) - w(old)$$

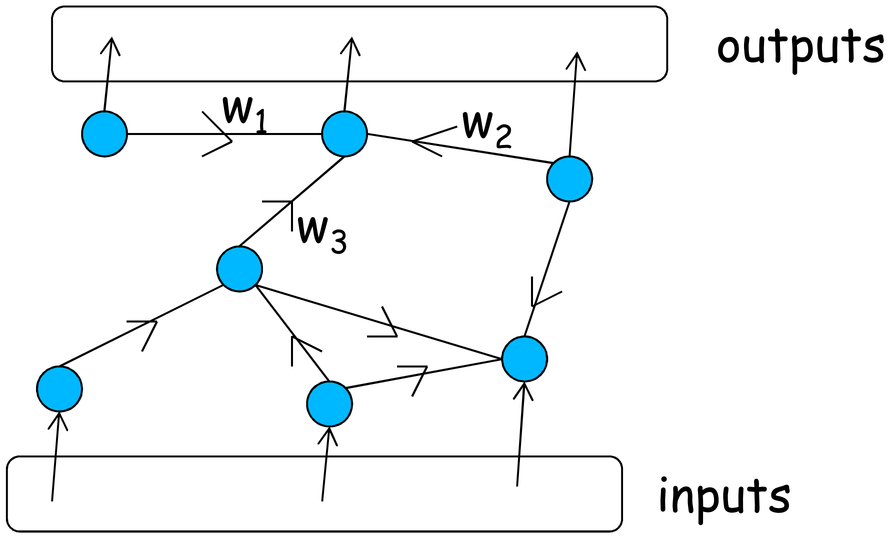
Extended Delta Rule: Modified for a differentiable activation function.

$$\Delta w = \sigma(t - y)xf'(\alpha)$$

PATTERN RECOGNITION USING NEURAL NETS

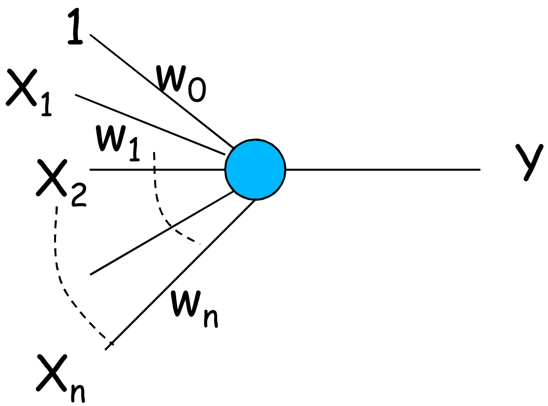
- A neural network (connectionist system) imitate the neurons in human brain.
- In human brain there are 10^{13} neurons.

A neural net model



- Each processing element either "fires" or it "does not fire"
- W_i – weights between neurons and inputs to the neurons.

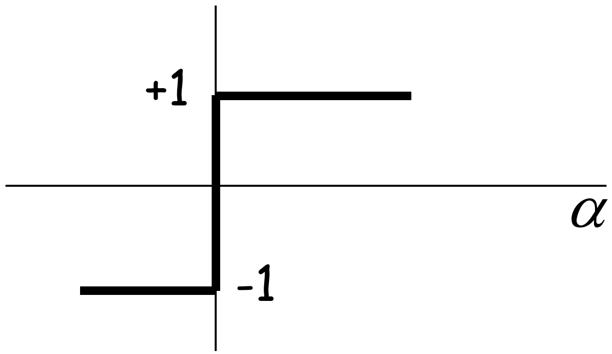
The model for each neuron:



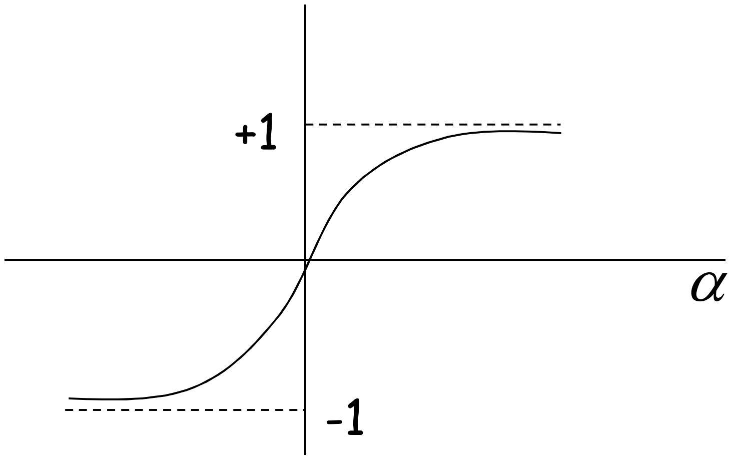
$$Y = f\left(\sum_{i=0}^{n} w_i x_i\right) = f\left(\sum_{i=1}^{n} w_i x_i - w_0\right) = f(\alpha)$$

f- activation function, normally nonlinear

Hard-limiter



Sigmoid



Sigmoid – $f(\alpha) = \dfrac{1}{1+e^{-\alpha}}$

TOPOLOGY: How neurons are connected to each other.

- Once the topology is determined, then the weights are to be found, using "learning samples". The process of finding the weights is called the learning algorithm.
- Negative weights – inhibitory
- Positive weights - excitatory

How can a NN be used for Pattern Classification?

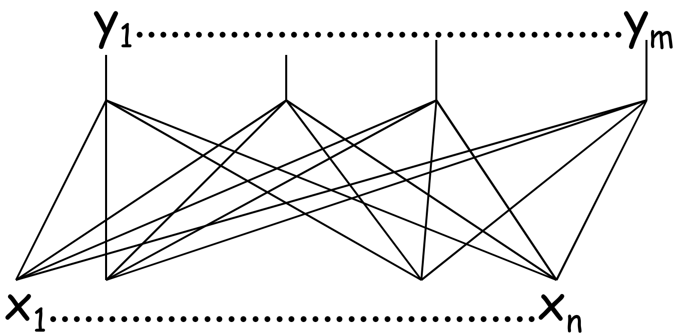
- Inputs are “feature vectors”
- Each output represent one category.
- For a given input, one of the outputs “fire” (The output that gives you the highest value). So the input sample is classified to that category.

Many topologies used for P.R.

- Hopfield Net
- Hamming Net
- Multilayer perceptron
- Kohonen’s feature map
- Boltzman Machines

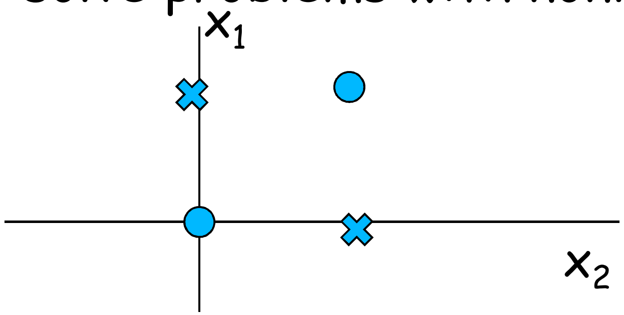
MULTILAYER PERCEPTRON

Single layer



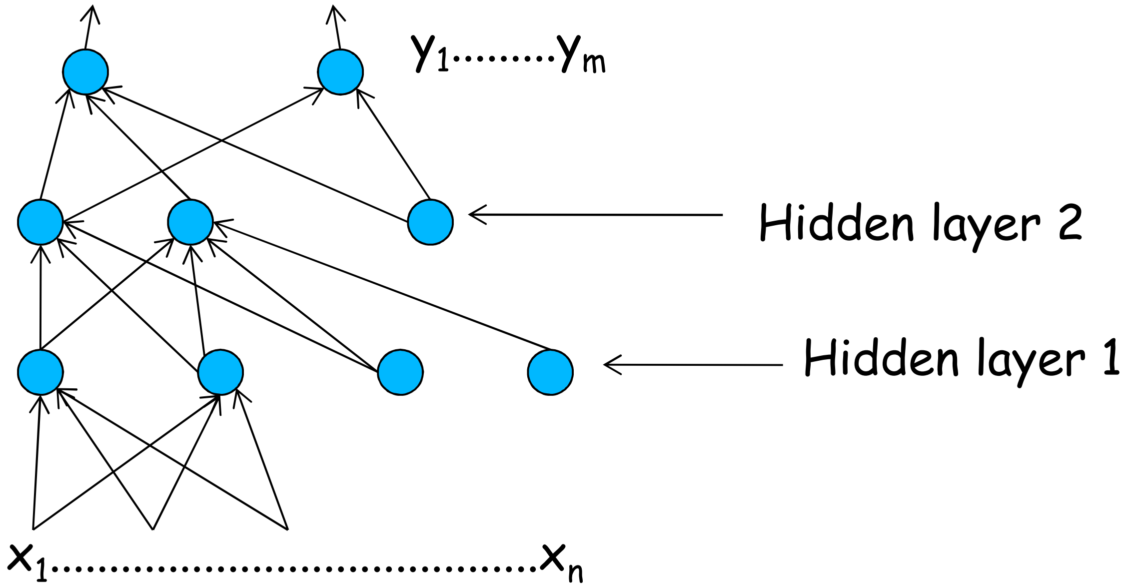
Linear discriminants:

- Cannot solve problems with nonlinear decision boundaries



•XOR problem
No linear solution exists

Multilayer Perceptron



Fully connected multilayer perceptron

- It was shown that a MLP with 2 hidden layers can solve any decision boundaries.

Learning in MLP:

Found in mid 80's.

Back Propagation Learning Algorithm

1- Start with arbitrary weights

2- Present the learning samples one by one to inputs of the network.

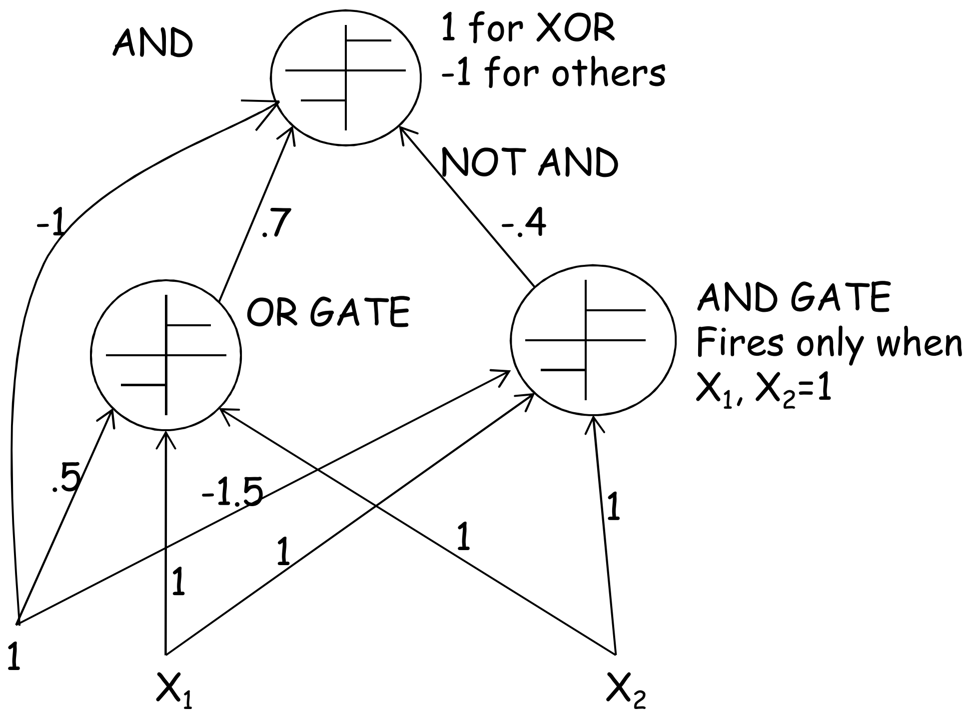
- If the network outputs are not as desired (y=1 for the corresponding output and 0 for the others)

    - adjust weights starting from top level by trying to reduce the differences

3- Propagate adjustments downwards until you reach the bottom layer.

4- Continue repeating 2 & 3 for all samples again & again until all samples are correctly classified.

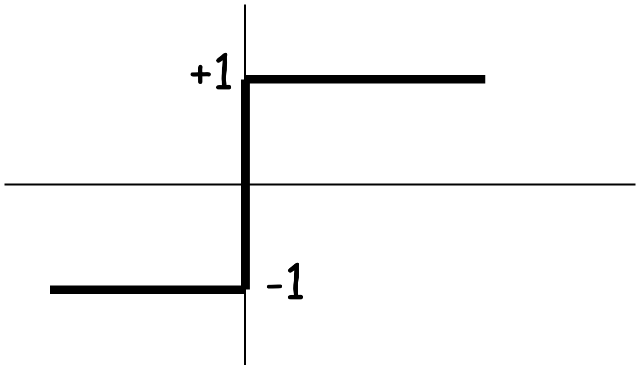
Example:



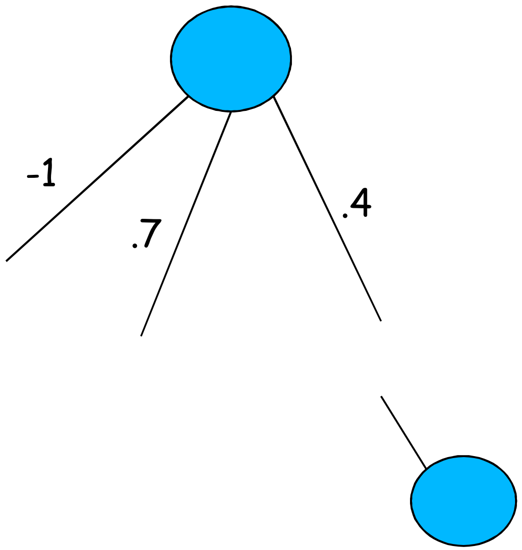
X_1, X_2=1 or -1
Output of neurons: 1 or -1

Output=1 for X_1, X_2=1,-1 or -1,1
=-1 for other combinations

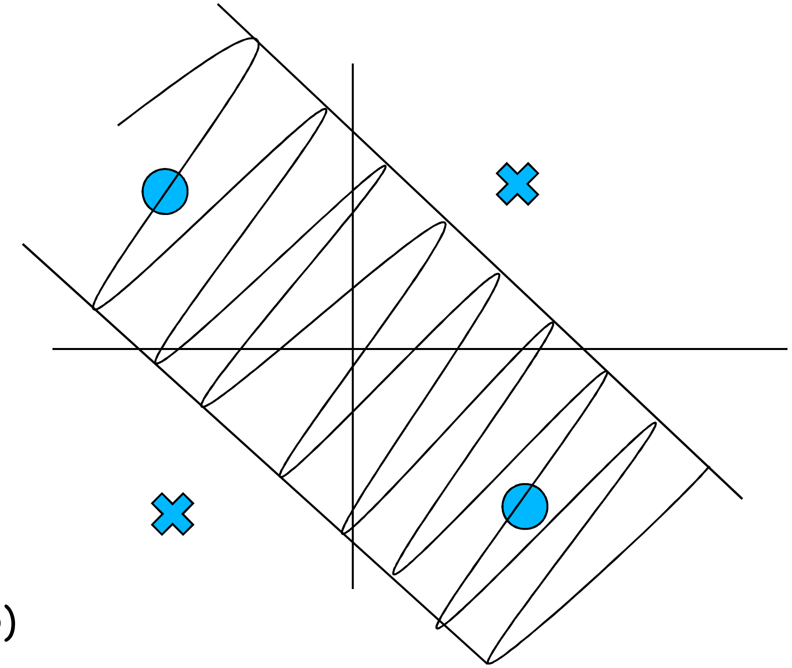
$$(X_1 + X_2)(\overline{X_1 X_2}) = (X_1 + X_2)(\overline{X_1} + \overline{X_2}) = \overline{X_1} X_2 + X_1 \overline{X_2}$$
$$= X_1 \quad XOR \quad X_2$$



Activation function

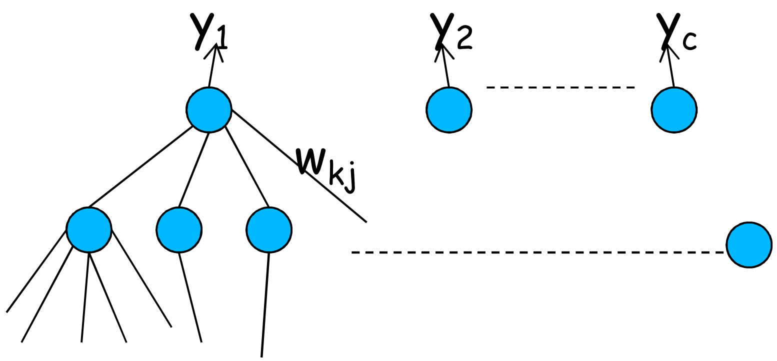


Take -(NOT AND)



Expressive Power of Multilayer Networks

- If we assume there are 3 layers as above, (input, output, one hidden layer).
- For classification, if there are c categories, d features and m hidden nodes. Each output is our familiar discriminant function.



$$y_k = g_k(X) = f\left(\sum_{j=1}^{m} w_{kj} f\left(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\right) + w_{k0}\right)$$

- By allowing f to be continuous and varying, is the formula for the discriminant function for a 3-layer (one input, one hidden, one output) network. (m – number of nodes in the hidden layer)

- "Any continuous function can be implemented with a layer 3 – layer network" as above, with sufficient number of hidden units. (Kolmogorov (1957)). <u>That means, any boundary can be implemented.</u>
- Then, optimum bayes rule, ($g(x)$ – a posteriori probabilities) can be implemented with such network!

$$g(X) = \sum_{j=1}^{2n+1} I_j\left(\sum_{i=1}^{d} w_{ij}(x_i)\right)$$

<u>In practice:</u>

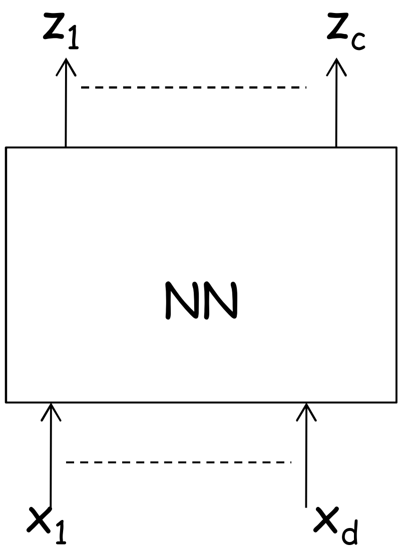
- How many nodes?
- How do we make it learn the weights with learning samples?
- Activity function?

<u>Back-Propagation Algorithm</u>

- Learning algorithm for multilayer feed-forward network, found in 80's by Rumelhart et al. at MIT.

- We have a sample set (labeled).

$$\{X_1, X_2, \ldots\ldots\ldots\ldots, X_n\}$$



$t_1, \ldots.., t_c$ - Target output

t_i-high, t_j low for $i \neq j$

$and \quad X_k \in C_i$

$z_1, \ldots\ldots, z_c$ -actual output

We want:

- Find W (weight vector) so that difference between the target output and the actual output is minimized. Criterion function

$$J(W) = \frac{1}{2}\sum (t_k - z_k)^2 = \frac{1}{2}\|T - Z\|^2$$

is minimized for the given learning set.

The Back Propagation Algorithm works basically as follows

1- Arbitrary initial weights are assigned to all connections

2- A learning sample is presented at the input, which will cause arbitrary outputs to be peaked. Sigmoid nonlinearities are used to find the output of each node.

3- Topmost layer's weights are changed to force the outputs to desired values.

4- Moving down the layers, each layers weights are updated to force the desired outputs.

5- Iteration continues by using all the training samples many times, until a set of weights that will result with correct outputs for all learning samples are found. (or $J(W) < \theta$)

The weights are changed according to the following criteria:

- If the node j is any node and i is one of the nodes a layer below (connected to node j), update w_{ij} as follows

  $w_{ij}(t+1) = w_{ij}(t) + \eta\delta_j Xi$ (Generalized delta rule)

- Where X_j is either the output of node I or is an input and

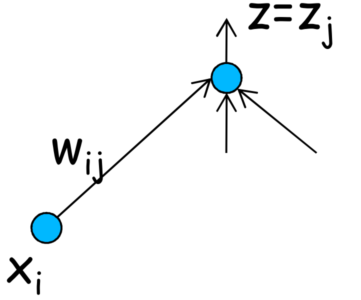
  $\delta_j = z_j(1 - z_j)(t_j - z_j)$

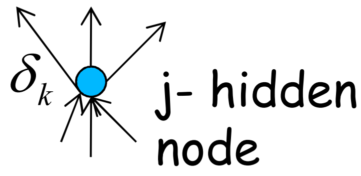
- in case j is an output node z_j is the output and t_j is the desired output at node j.

If j is an intermediate node,

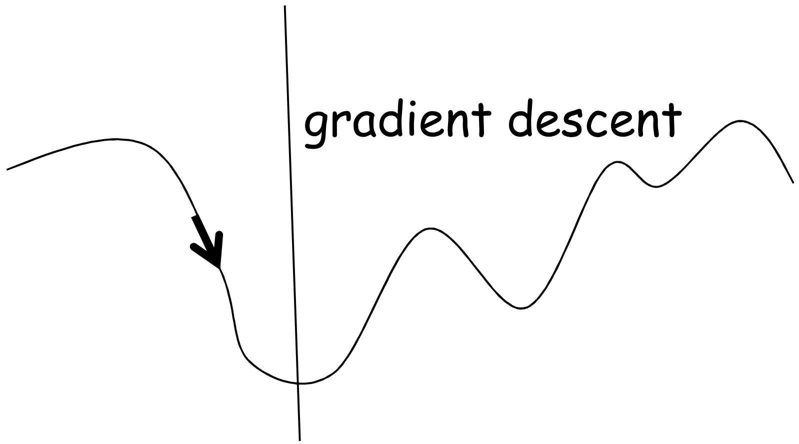
$$\delta_j = x_j(1 - x_j)\sum_{k=1}^{m} \delta_k w_{jk}$$

where x_j is the output of node j.





In case of j is output z_j



How do we get these updates?

Apply gradient descent algorithm to the network.

$$J(W) = \frac{1}{2}\|T - Z\|^2$$

Gradient Descent: move towards the direction of the negative of the gradient of J(W)

$$\Delta W = -\eta \frac{\partial J}{\partial W}$$

η - learning rate

For each component w_{ij}

$$\Delta w_{ij} = -\eta \frac{\partial J}{\partial w_{ij}}$$

$$w_{ij}(t+1) = w(t) + \Delta w_{ij}$$

Now we have to evaluate $\frac{\partial J}{\partial w_{ij}}$ for output and hidden nodes. But we can write this as follows:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial \alpha_j} \frac{\partial \alpha_j}{\partial w_{ij}}$$

where

$$\alpha_j = f\left(\sum_{i=1}^{k} w_{ij} X_i \right)$$

But $\dfrac{\partial \alpha_j}{\partial w_{ij}} = X_i$

Now call $\dfrac{\partial J}{\partial \alpha_j} = \delta_j$

Then, $\Delta w_{ij} = -\eta \delta_j X_i$

Now, δ_j will vary depending on j being an output node or hidden node.

Output node use chain rule

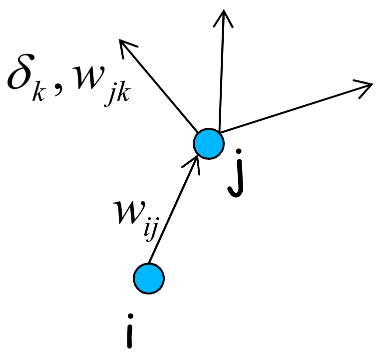
$$\frac{\partial J}{\partial net_j} = \frac{\partial J}{\partial z_j} \underbrace{\frac{\partial z_j}{\partial net_j}}_{\text{Derivative of the activation function (sigmoid)}}$$

$$= (t_j - z_j)(z_j)(1 - z_j)$$

So, $w_{ij}(t+1) = w_{ij}(t) + \eta (z_j)(1 - z_j)(t_j - z_j) X_i$

Hidden Node: use chain rule again

$$\delta_j = \frac{\partial J}{\partial net_j}$$



$$\Delta w_{ij} = \frac{\partial J}{\partial w_{ij}} = \underset{\text{evaluate}}{\underset{\uparrow}{\frac{\partial J}{\partial y_j}}} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

Now

$$\frac{\partial J}{\partial y_i} = \frac{\partial}{\partial y_i}\left[\frac{1}{2}\sum \left[t_k - z_k\right]^2\right]$$

$$= \frac{1}{2}\sum_{k=1}^{c} \frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial y_j} = \sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial z_k}{\partial y_j}$$

$$= -\sum \underbrace{(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j}}_{\delta_k} = -\sum_{k=1}^{c} \delta_k w_{kj}$$

So

$$w_{ij}(t+1) = w_{ij}(t) + \eta\left[\sum_{k=1}^{c} \delta_k w_{ki}\right] X_i$$