Artificial Intelligence
CS 401

(Artificial Neural Network-IV)
Lecture No. 14

Instructor: Dr. Kashif Zafar

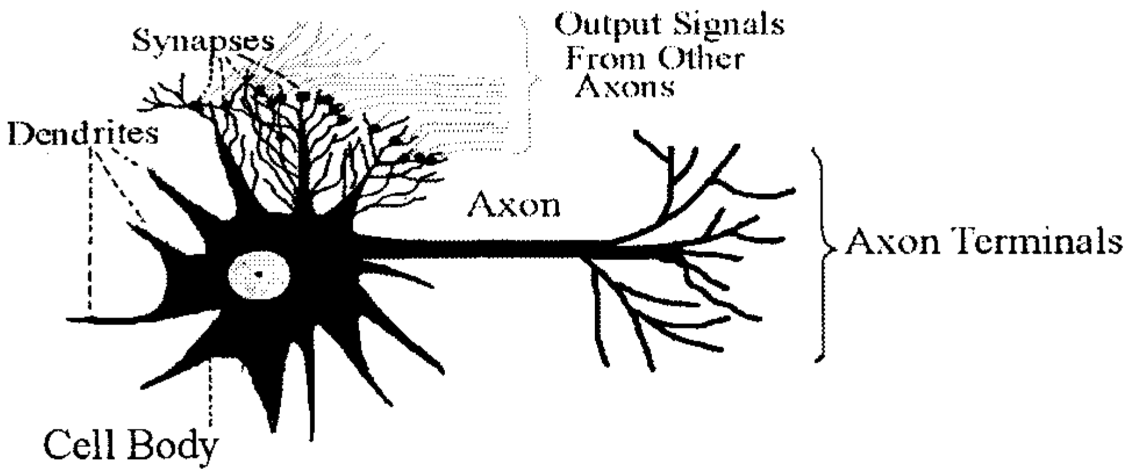
November 25, 2021

National University of Computer and Emerging Sciences,
Lahore

NEURAL NETWORKS

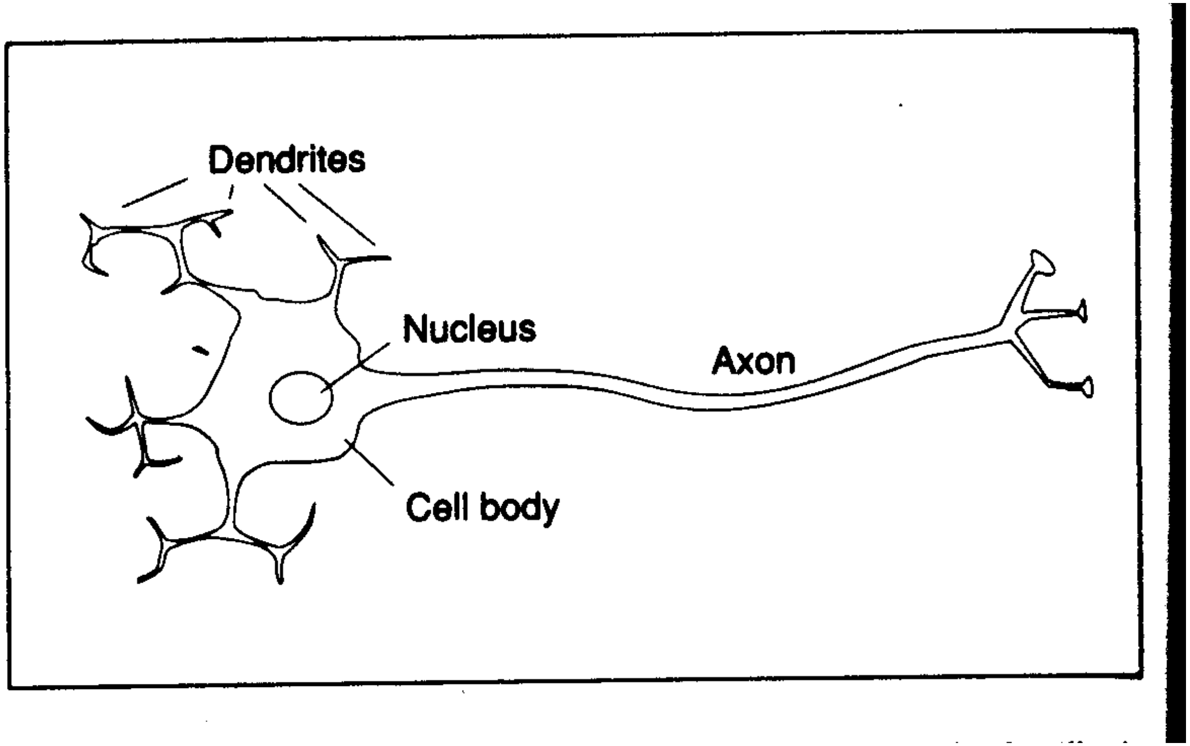
Biological Neuron

About 100 billion neuron in the human brain



NEURAL NETWORKS

Biological Neuron



NEURAL NETWORKS

Biological Neuron

The main body of the cell collects the incoming signals from the other neurons through its dendrites

The incoming signals are constantly being summed in the cell body

If the result of the summation crosses as certain threshold, the cell body emits a signal of its own (called firing of the neuron)

This signal passes through the neuron’s axon, from where the dendrites of other neurons pick it up

NEURAL NETWORKS

Biological Neuron

There are 1,000 to 10,000 dendrites in each neuron (few millimeters long)

There is only one axon (several centimeters long)

The connection between dendrites and axon is electrochemical and it is called synapses

The synapses modify the signal while passing it on to dendrites

The human learning is stored in these synapses, and the connection of neurons with other neurons

NEURAL NETWORKS

Biological Neuron

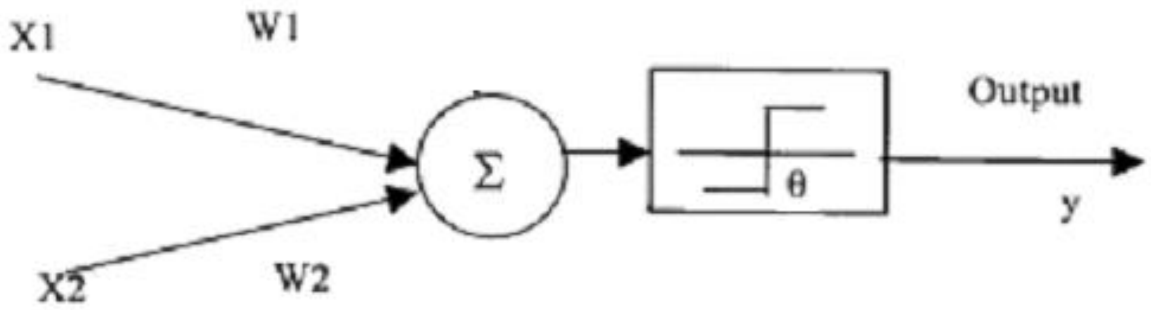
The human learning is stored in these synapses, and the connection of neurons with other neurons

If stimulus at a dendrite causes the neuron to fire, then the connection between that dendrite and axon is strengthened

If the arrival of stimulus does not cause the neuron to fire, the connection weakens over time

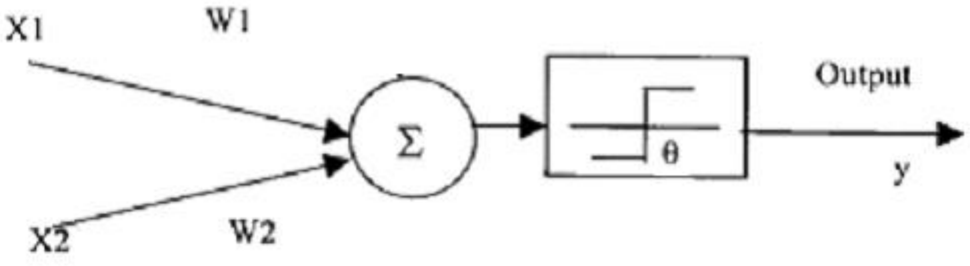
NEURAL NETWORKS

Artificial Neuron Model



NEURAL NETWORKS

Artificial Neuron Model



Implementation of AND function

Let $W_1 = W_2 = 1$

X_1	X_2	$X_1W_1 + X_2W_2$	Y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

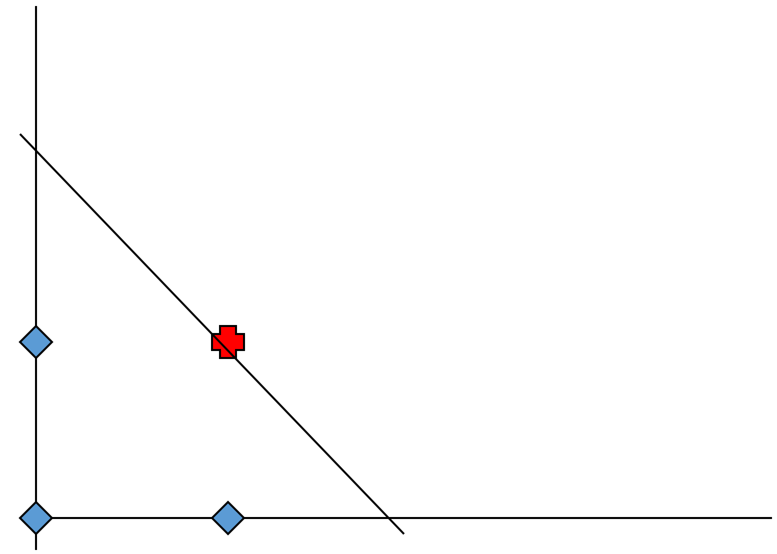
If we make $\theta = 2$ (or any value >1 but <=2), we will get correct results with a unit step activation function

NEURAL NETWORKS

Artificial Neuron Model

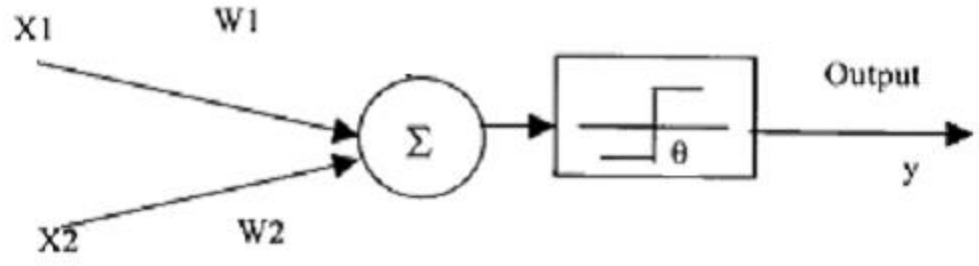
If we place the 4 points in a two coordinate system (X1 and X2), we have drawn a line from (2, 0) to (0, 2) in the resulting plane

Any new data falling on the left side of the line will give an output of zero and the data on the right side of the line will be classified as one



NEURAL NETWORKS

Artificial Neuron Model



Implementation of OR function

Let $W_1 = W_2 = 1$

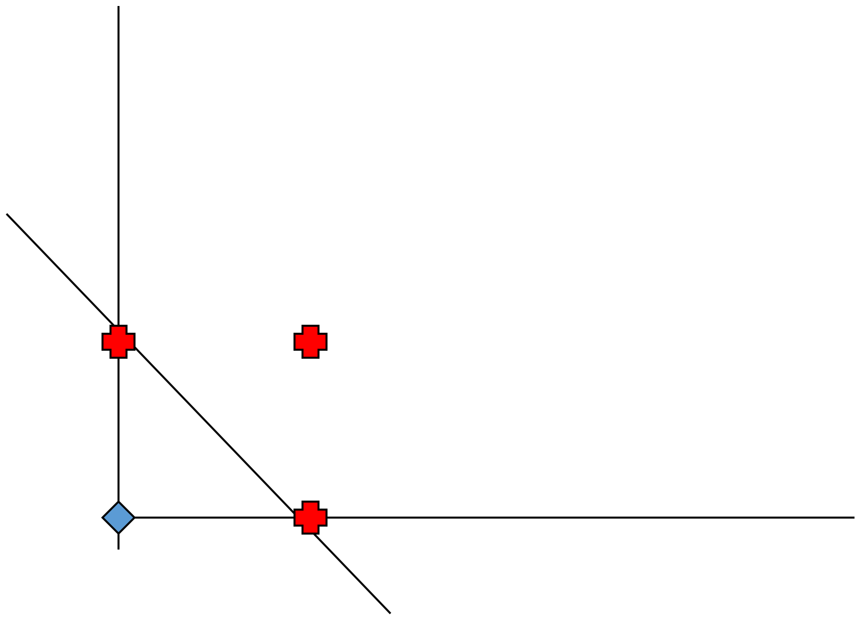
X_1	X_2	$X_1W_1 + X_2W_2$	Y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	1

If we make θ = 1 (or any value >0 but <=1), we will get correct results with a unit step activation function

NEURAL NETWORKS

Artificial Neuron Model

If we place the 4 points in a two coordinate system (X1 and X2), we have drawn a line from (1, 0) to (0, 1) in the resulting plane



Any new data falling on the left side of the line will give an output of zero and the data on the right side of the line will be classified as one

Perceptron Learning Rule

Show the mathematical working of Artificial Neural Network by taking the case in figure below. First two columns are the input values for X1 and X2 and the third column is the desired output.

0	0	0
0	1	0
1	0	1
1	1	1

- **Learning rate = 0.2**
- **Threshold = 0.5**
- **Actual output = W1X1+W2X2**
- **Next weight adjustment = Wn+ΔWn**
- **Change in weight = ΔWn = learning rate * (desired output- actual output) * Xn**
- **In Figure. First two columns are input vectors x1 and x2 and last column is the desired output y.**

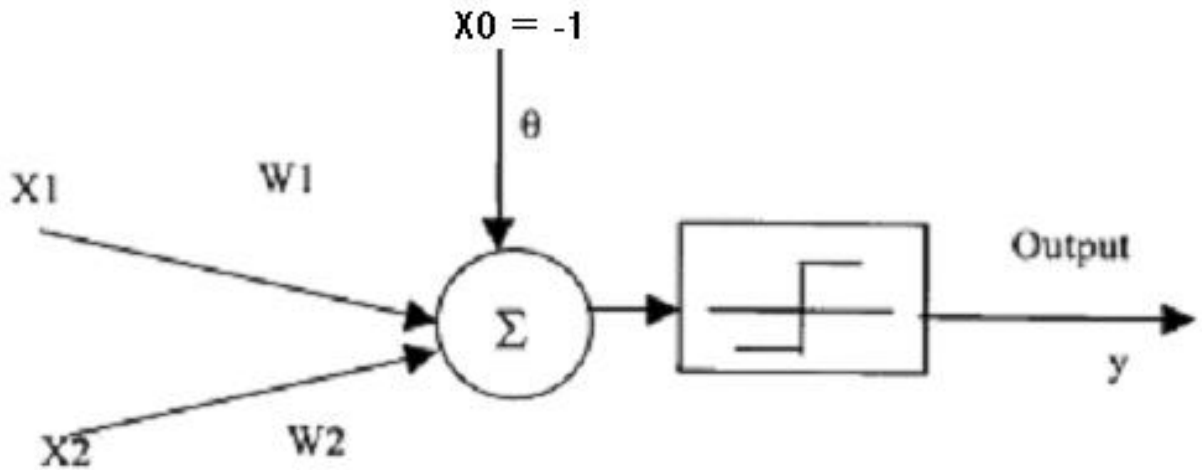
- **Show the complete iterations for acquiring the desired output?**

x1	x2	w1	w2	d	y	Δ w1	Δ w2

NEURAL NETWORKS

Artificial Neuron Model

If we want to utilize a unit step function centered at zero for both AND and OR neurons, we can incorporate another input X0 constantly set at –1
The weight W0 corresponding to this input would be the θ, calculated previously



NEURAL NETWORKS

Setting of weights (Training)

X_1	X_2	Y
1.0	1.0	1
9.4	6.4	-1
2.5	2.1	1
8.0	7.7	-1
0.5	2.2	1
7.9	8.4	-1
7.0	7.0	-1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	-1

NEURAL NETWORKS

Training of weights

Supervised training: the classes of training samples are known

Random initialization of weights
(threshold or bias is considered as a weight and its input is fixed at 1)

NEURAL NETWORKS

Training of weights: Algorithm

For each input, calculate the output with the current weights

The error will be equal to $y_{desired} - f(\sum x_i w_i)$

How to reduce this error?

Input is fixed

We have to change weights

Which weight to change?

Since we don't know which weight is contributing how much to the error, hence we change all weights

NEURAL NETWORKS

Training of weights: Algorithm

How much will be the change in a weight?

The change in each connection weight has to be proportional to the error and the magnitude of its input

(we assume that each weight is contributing to the error a value proportional to its input)

Change each weight by $\delta w_i = \eta(y - f(\sum x_i w_i))x_i$

The learning rate η **is fixed at a small value, so that we may not only learn the current data sample, but also the retain the previous learning**

NEURAL NETWORKS

Training of weights: Algorithm

Iterate repeatedly over the whole data set.

Stop when the combined error of all the inputs is below a certain threshold

NEURAL NETWORKS

Training of weights

For our example let [0.75, 0.5, –0.6] be the initial weights (set randomly)

Let the learning rate be 0.2

The first data sample is classified correctly
The second data sample gives an error of –2 and the weight vector should be updated

$$W_{new} = W_{old} + 0.2 * Error * X$$
$$= \begin{pmatrix} -3.01 \\ -2.06 \\ -1.00 \end{pmatrix}$$

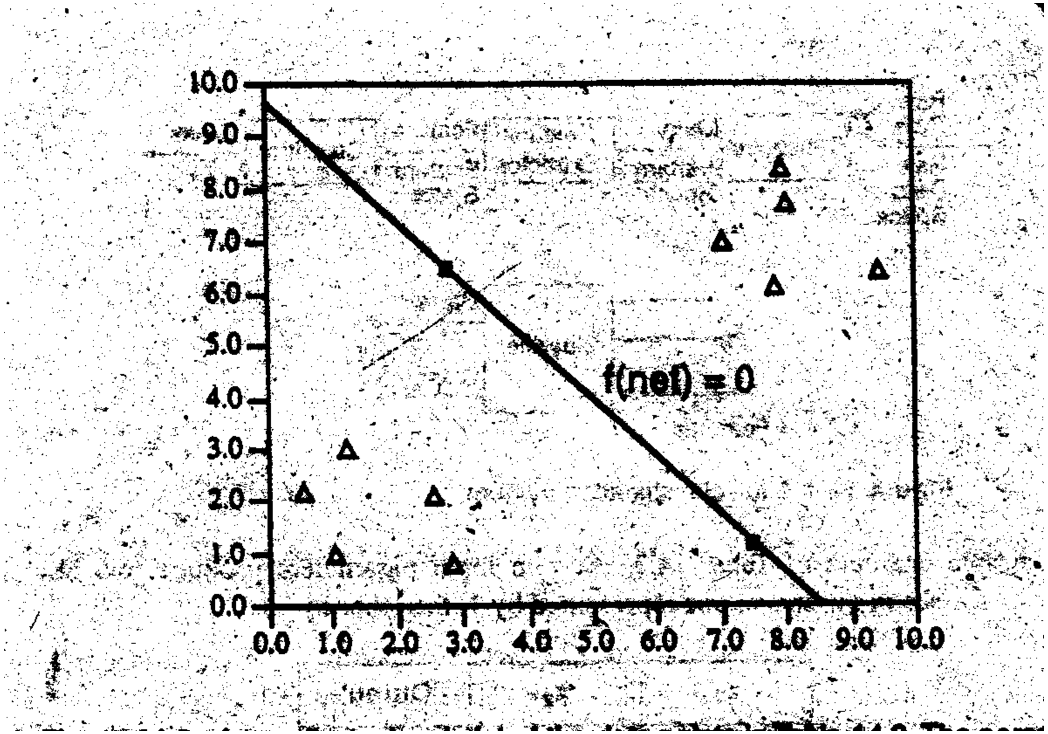
NEURAL NETWORKS

Training of weights

After 500 iterations the weights converge to [-1.3, -1.1, 10.9]
i.e. output = f(∑xiwi) = f(1.3x1 –1.1x2 + 10.9)

To draw the boundary line we take the output as zero on the boundary i.e.

$1.3x_1 - 1.1x_2 + 10.9 = 0$



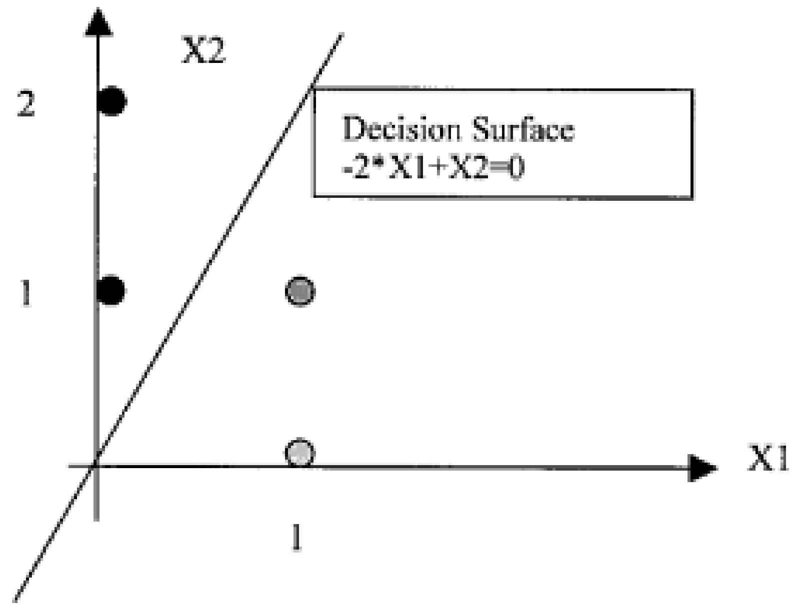
Reading Assignment & References

Section 10.1.1, 10.2.1 and 10.2.2 of Luger

NEURAL NETWORKS

Linearly Separable Problems

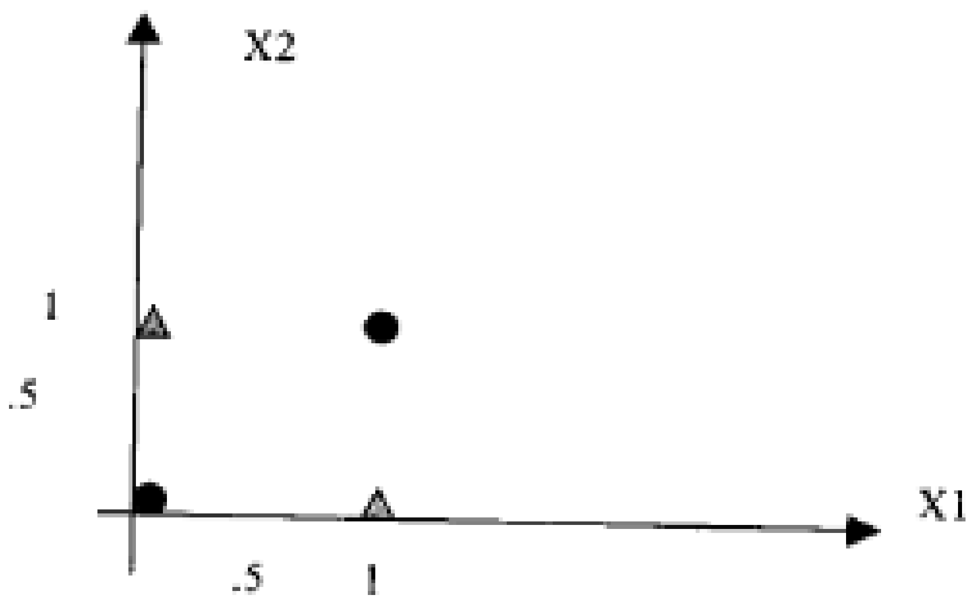
If the data can be correctly divided into two categories by a line or hyper-plane



NEURAL NETWORKS

Non Linear Problems

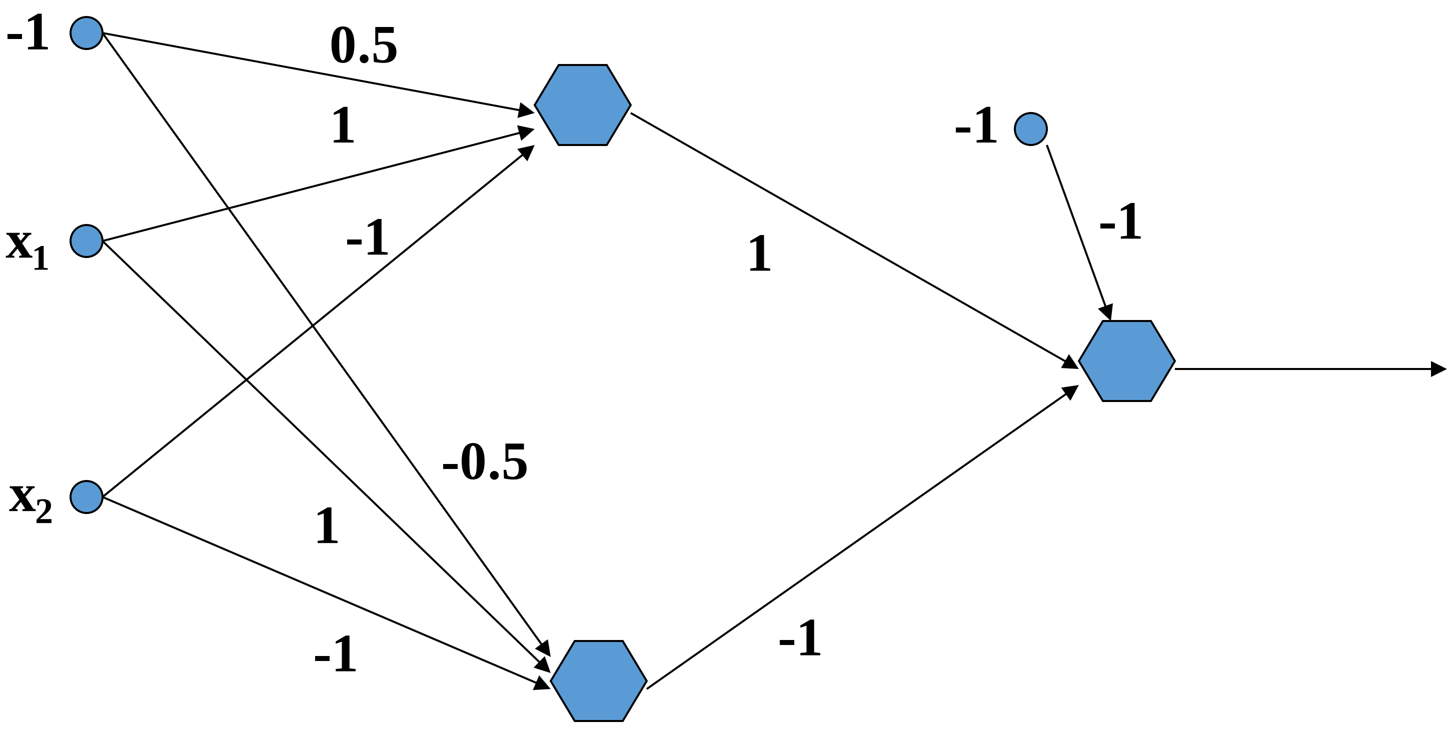
If the data cannot be correctly separated by a single line or plane, then it is not linearly separable. A single layer of neurons cannot classify the input patterns that are not linearly separable; e.g. exclusive OR problem



NEURAL NETWORKS

Non Linear Problems

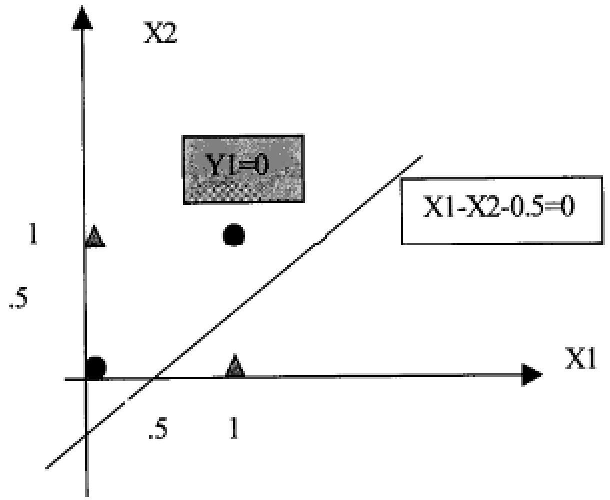
For such problems we need two or more layers of neurons



Non Linear Problems

The first neuron's output is
Point (0, 0) => Sum = -0.5 and Y1 = 0
Point (0, 1) => Sum = -1.5 and Y1 = 0
Point (1, 0) => Sum = 0.5 and Y1 = 1
Point (1, 1) => Sum = -0.5 and Y1 = 0

It places a line

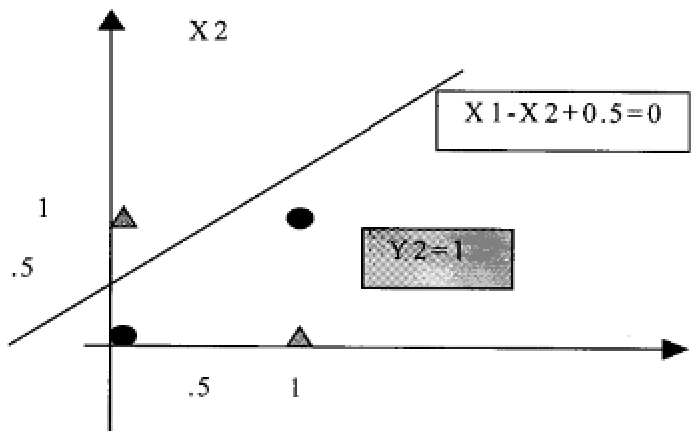


NEURAL NETWORKS

Non Linear Problems

The second neuron's output is
Point (0, 0) => Sum = 0.5 and Y2 = 1
Point (0, 1) => Sum = -0.5 and Y2 = 0
Point (1, 0) => Sum = 1.5 and Y2 = 1
Point (1, 1) => Sum = 0.5 and Y2 = 1

It places a line

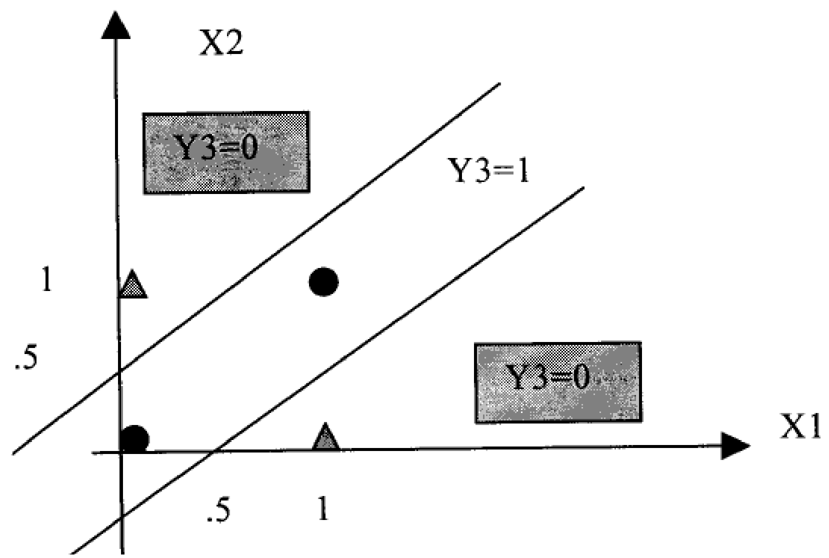


NEURAL NETWORKS

The input sum for the neuron of the 2nd layer = Y1 – Y2 + 1

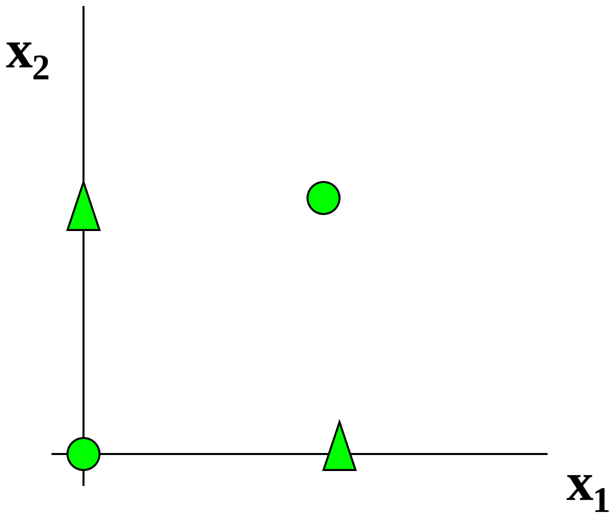
For point (0,1) Y1 = 0, Y2 = 0, Sum = 1, Y3 = 0
For point (1,0) Y1 = 1, Y2 = 1, Sum = 1, Y3 = 0
For point (0,0) Y1 = 0, Y2 = 1, Sum = 0, Y3 = 1
For point (1,1) Y1 = 0, Y2 = 1, Sum = 0, Y3 = 1

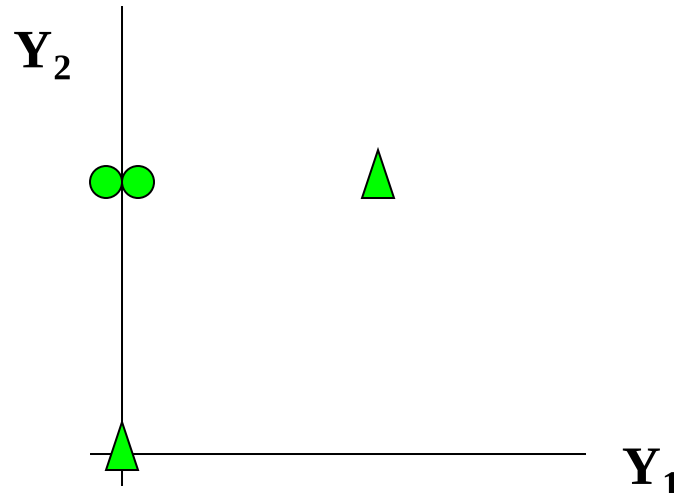
Thus the neuron of the 2nd layer separates the data correctly



NEURAL NETWORKS

The first layer projects the input data (x_1, x_2) into another dimension (Y_1, Y_2)

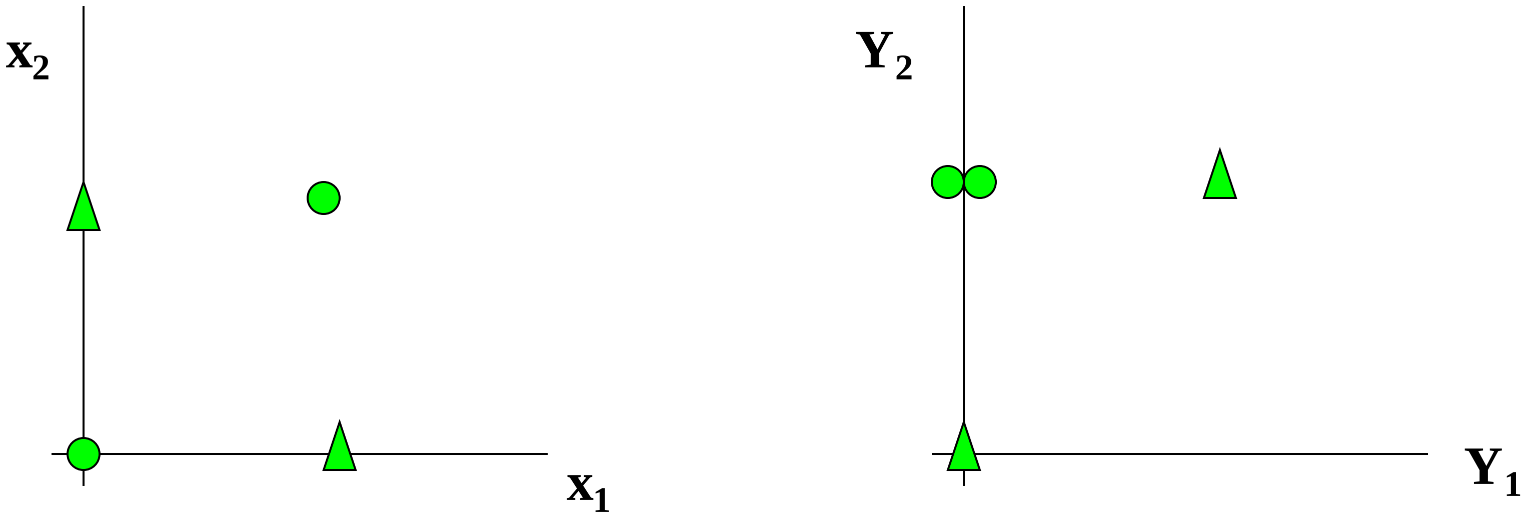




In dimension (Y_1, Y_2), the data becomes linearly seperable

NEURAL NETWORKS

The first layer projects the input data (x_1, x_2) into another dimension (Y_1, Y_2)

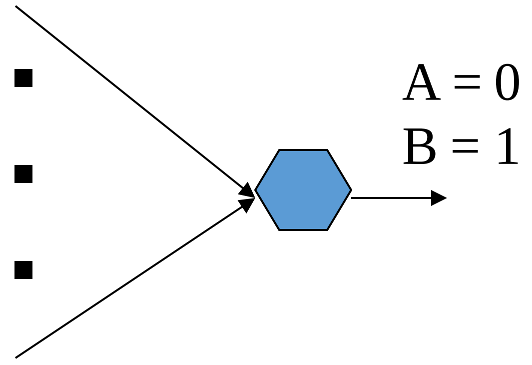


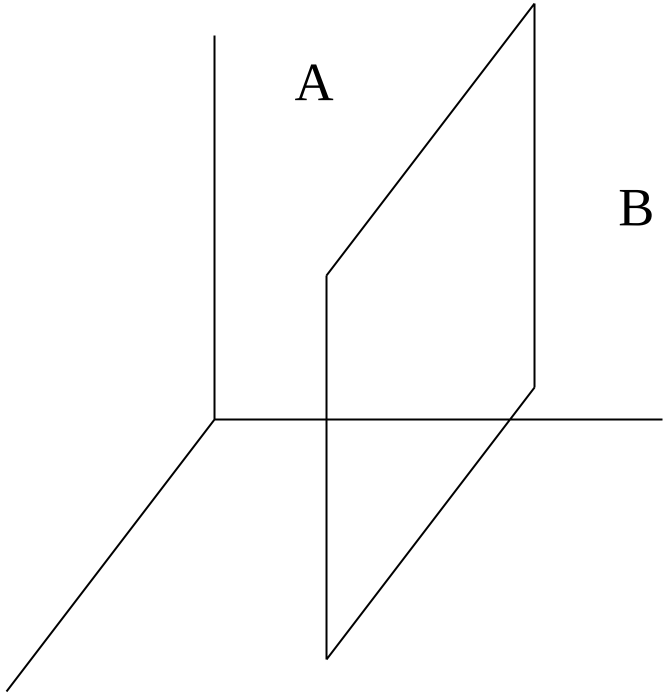
In dimension (Y_1, Y_2), the data becomes linearly seperable

NEURAL NETWORKS

Multi Layer Perceptron: Utilization

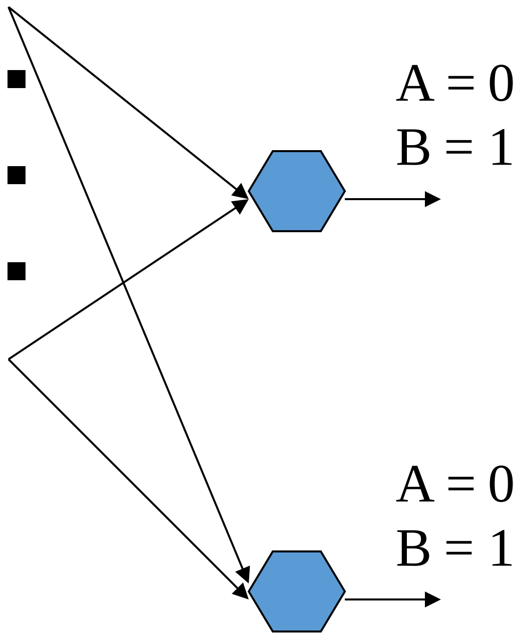
Hence a single neuron with unit step activation function can classify the input into two categories

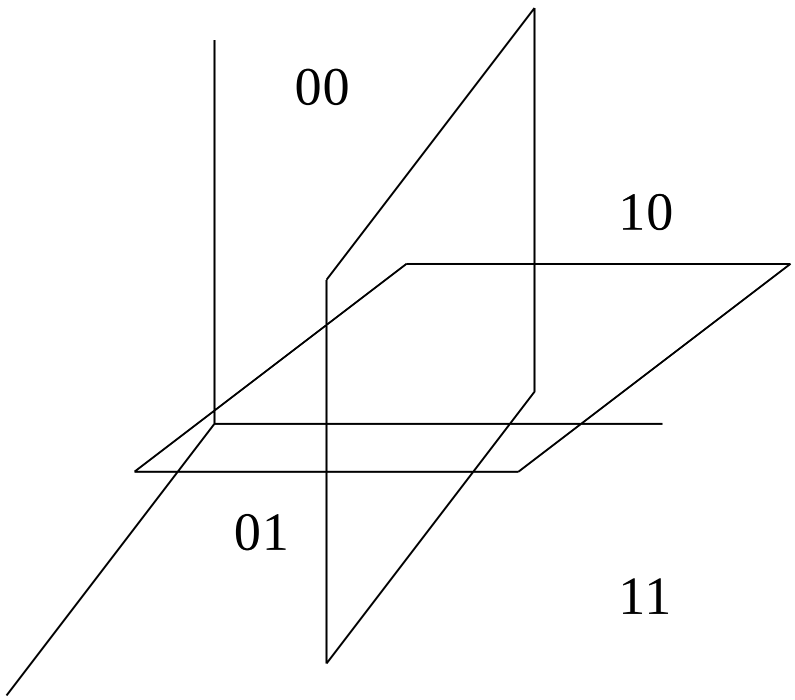




Multi Layer Perceptron: Utilization

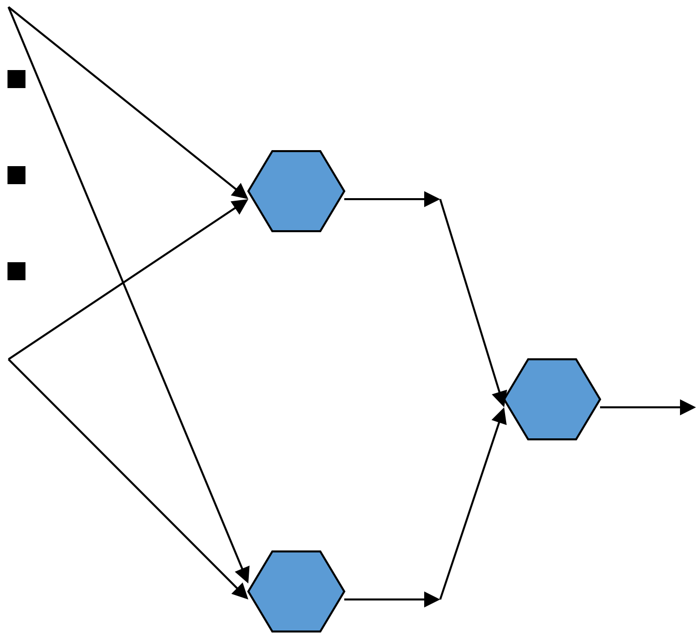
Two neurons with unit step activation function can classify the input into four categories

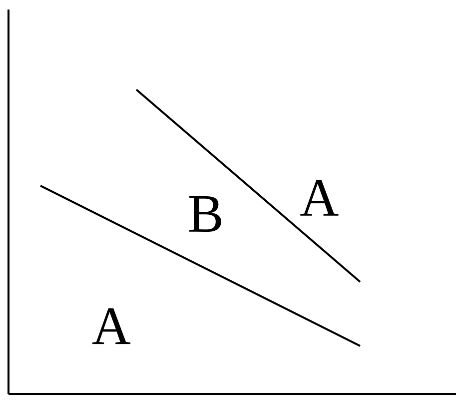




Multi Layer Perceptron: Utilization

Two layers of neurons can classify complex types of inputs





Common Activation Functions

1. Binary Step Function

(a "threshold" or "Heaviside" function)

$$f(x) = \begin{cases} 0 & if \;\; x < 0 \\ 1 & if \;\; x \geq 0 \end{cases}$$

2. Bipolar Step Function

$$f(x) = \begin{cases} -1 & if \;\; x < 0 \\ 1 & if \;\; x \geq 0 \end{cases}$$

3. Binary Sigmoid Function

(Logistic Sigmoid)

$$f(x) = \frac{1}{1+\exp(-\sigma x)}; \; f'(x) = \sigma f(x)[1 - f(x)]$$

3. Bipolar Sigmoid

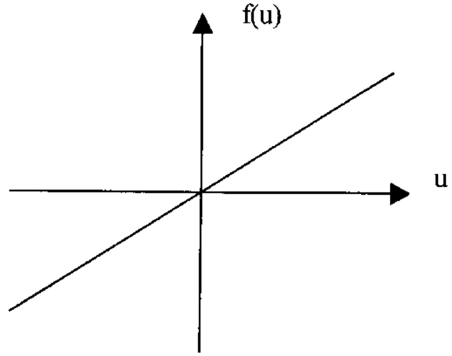
$$g(x) = \frac{1-\exp(-\sigma x)}{1+\exp(-\sigma x)}; \; g'(x) = \frac{\sigma}{2}[1+g(x)][1-g(x)]$$

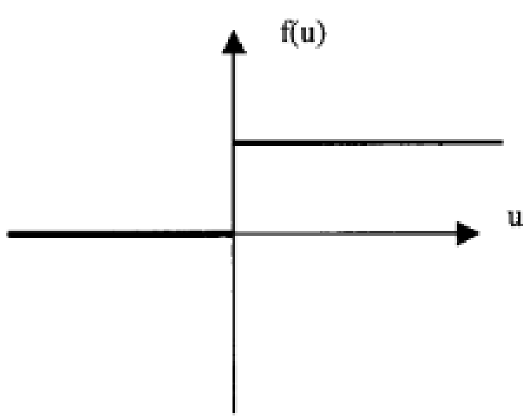
4. Hyperbolic - Tangent

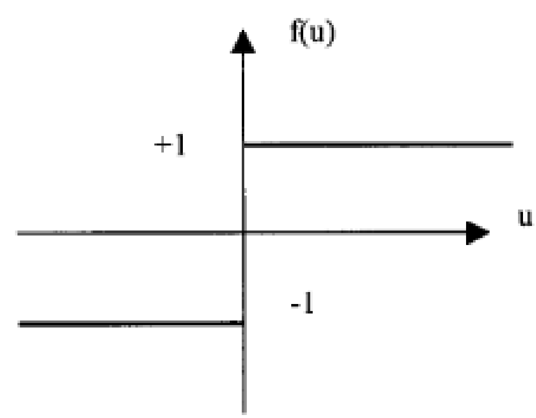
$$h(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}; \; h'(x) = [1+h(x)][1-h(x)]$$

NEURAL NETWORKS

Multi Layer Perceptron: Activation functions



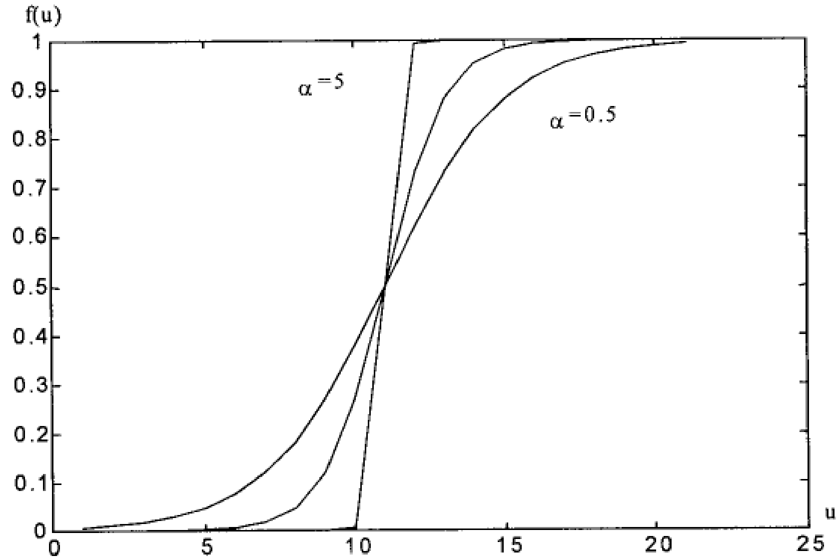




NEURAL NETWORKS

Multi Layer Perceptron: Activation functions

$$f(x) = \frac{1}{1+e^{-\alpha x}}, \quad 0 \le f(x) \le 1 \qquad (2.5)$$



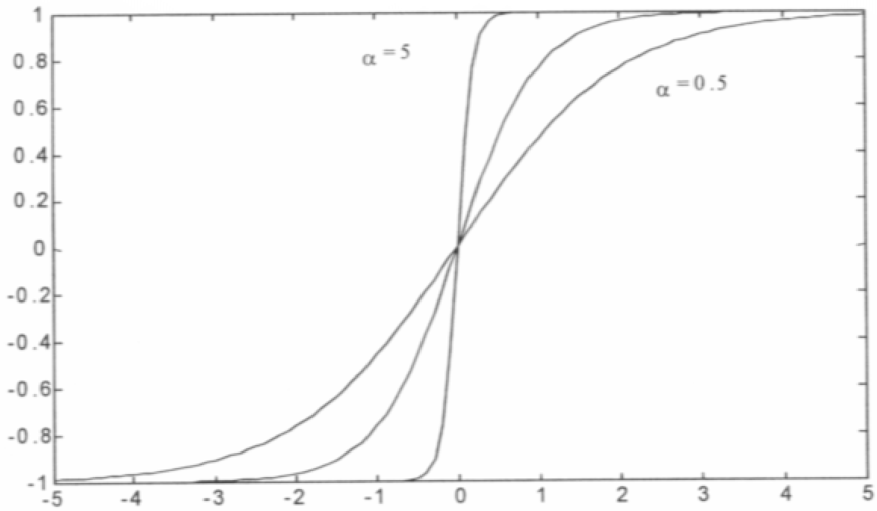
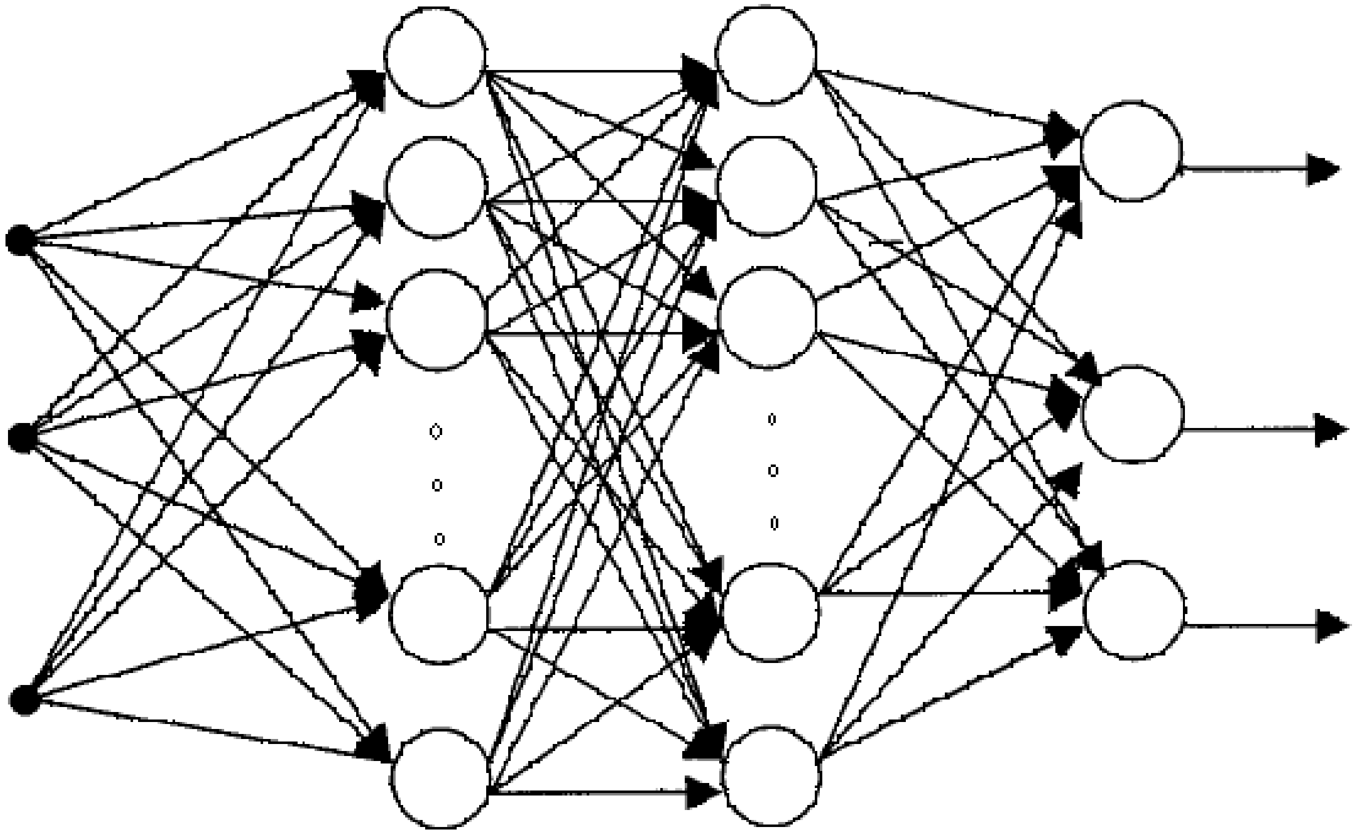


Figure 2.8: A Tangent Hyperbolic Activation Function.

NEURAL NETWORKS

Multi Layer Perceptron: Architecture & Forward Pass



Input Units **Hidden Units** **Output Units**

Backpropagation Algorithm

- **Set up the architecture & initialize the weights of the network**
- **Apply the training pairs (input-output vectors) from the training set, one by one**
- **For each training pair, calculate the output of the network**
- **Calculate the error between actual output & desired output**
- **Propagate the error backwards & adjust the weights in such a way that minimizes the error**
- **Repeat the above steps for each pair in the training set until the error for the set is lower than the required minimum error**

NEURAL NETWORKS

Multi Layer Perceptron:

Training by Backpropagation Algorithm

Let E = accumulative error over a data set. It is a function of network weights

$$E = \sum_{training\ samples} \sum_j (d_j - O_j)^2$$

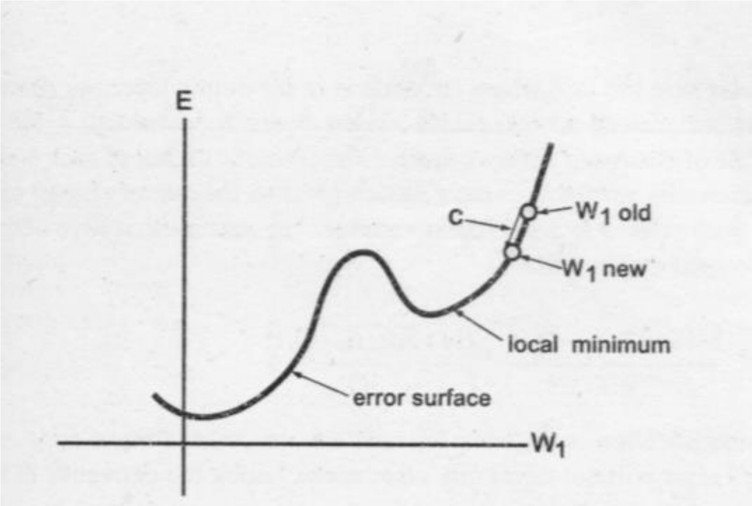
d_j is the desired output of node j and O_j is the actual output

The error is squared so that the positive and negative errors may not cancel each other out during summation

Multi Layer Perceptron:

Training by Backpropagation Algorithm

Each weight configuration can be represented by a point on an error surface

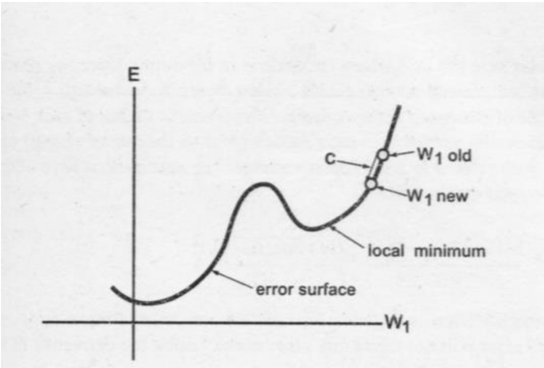


NEURAL NETWORKS

Multi Layer Perceptron: Training

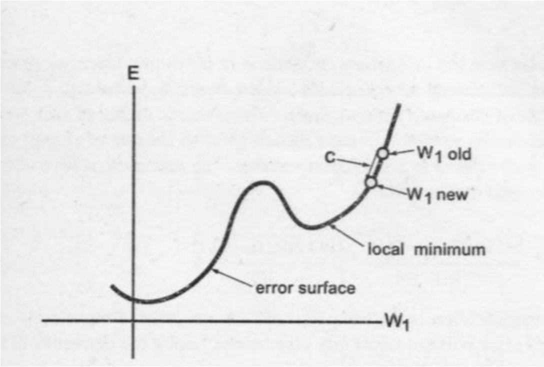
Starting from a random weight configuration, we want our training algorithm to move in the direction where error is reduced more rapidly

Delta rule attempts to minimize the local error and uses the derivative of the error to find the slope of the error space in the region local to a particular point



NEURAL NETWORKS

Multi Layer Perceptron: Training



Delta rule uses gradient descent:

$$\Delta w_{ij} = -c\,(\partial Error/\,\partial w_{ij})$$

Let current weight be 4

Then $\partial Error/\,\partial w_{ij} = 9.5 - 9.0/5 - 3 = 0.25$

The new weight will be $w_{new} = w_{old} - c{*}0.25 = 3.875$

if c = 0.5 (c is the learning rate)

If the error curve had been downward, then

$$\partial Error/\,\partial w_{ij} = 9.0 - 9.5/5 - 3 = -0.25$$

The new weight will be

$$w_{new} = w_{old} - c{*}{-0.25} = 4.125$$

NEURAL NETWORKS

Multi Layer Perceptron: Training

Delta rule:

$$\Delta w_{ij} = -c\,(\partial Error / \partial w_{ij})$$

If the learning constant "c" is large (more than 0.5), weights move quickly to optimal value but there is a risk of overshooting the minimum or oscillation around optimum weights

If "c" is small, the training is less prone to these problems but system does not learn quickly; also the algorithm may get stuck in local minima

NEURAL NETWORKS

Multi Layer Perceptron: Training

The weights are updated incrementally, following the presentation of each training example

This corresponds to a stochastic approximation to gradient descent

To obtain the true gradient of Error, one would consider all of the training examples before altering the weight values

The stochastic approximation avoids costly computations per weight update

NEURAL NETWORKS

Multi Layer Perceptron: Training of Output Layer Weights

Randomly set the weights
Present first training input vector to the network
Calculate the outputs of all neurons
The inputs to the last layer of neurons would be the output of 2^{nd} last layer
We calculate the Error of all the output neurons and now we wish to change the weights of an output neuron "j" so that its error reduces

We use Delta rule:

$$\Delta w_{ij} = -c\ (\partial Error /\ \partial w_{ij})$$

NEURAL NETWORKS

Multi Layer Perceptron: Training of Output Layer Weights

The equation $\partial Error/ \partial w_{ij}$ means that we want the rate of change of network error as a function of the change in one of weights of an output node j

Since for our current training sample

$$Error = \sum_j (d_j - O_j)^2$$

Where O_j is itself a function of other variables (including w_{ij}), therefore we use partial derivatives (they gives us the rate of change of a multi-variable function w.r.t a particular variable)

Multi Layer Perceptron: Training of Output Layer Weights

To calculate this quantity we use chain rule

The Error is only indirectly dependent on w_{ij}, but it is directly dependent on variable O_j

$$\partial Error / \partial w_{ij} = (\partial Error / \partial O_j) \cdot (\partial O_j / \partial w_{ij})$$

$\partial Error / \partial O_j$ = rate of change of error w.r.t output of node j

Now $\partial Error / \partial O_j = \sum_j (d_j - O_j)^2 / \partial O_j = -2(d_j - O_j)$

For $\partial O_j / \partial w_{ij}$ we have $(\partial O_j / \partial act_j)(\partial act_j / \partial w_{ij})$

$(\partial O_j / \partial act_j) = (\partial f(act)_j / \partial act_j) = f'(act_j)$

$(\partial act_j / \partial w_{ij}) = (\partial \sum_i x_i w_{ij} / \partial w_{ij}) = x_i$

Hence $\Delta w_{ij} = -c\,(\partial Error / \partial w_{ij}) = -c[-2(d_j - O_j) \cdot f'(act_j) \cdot x_i]$

NEURAL NETWORKS

Multi Layer Perceptron: Training of Hidden Layer Weights

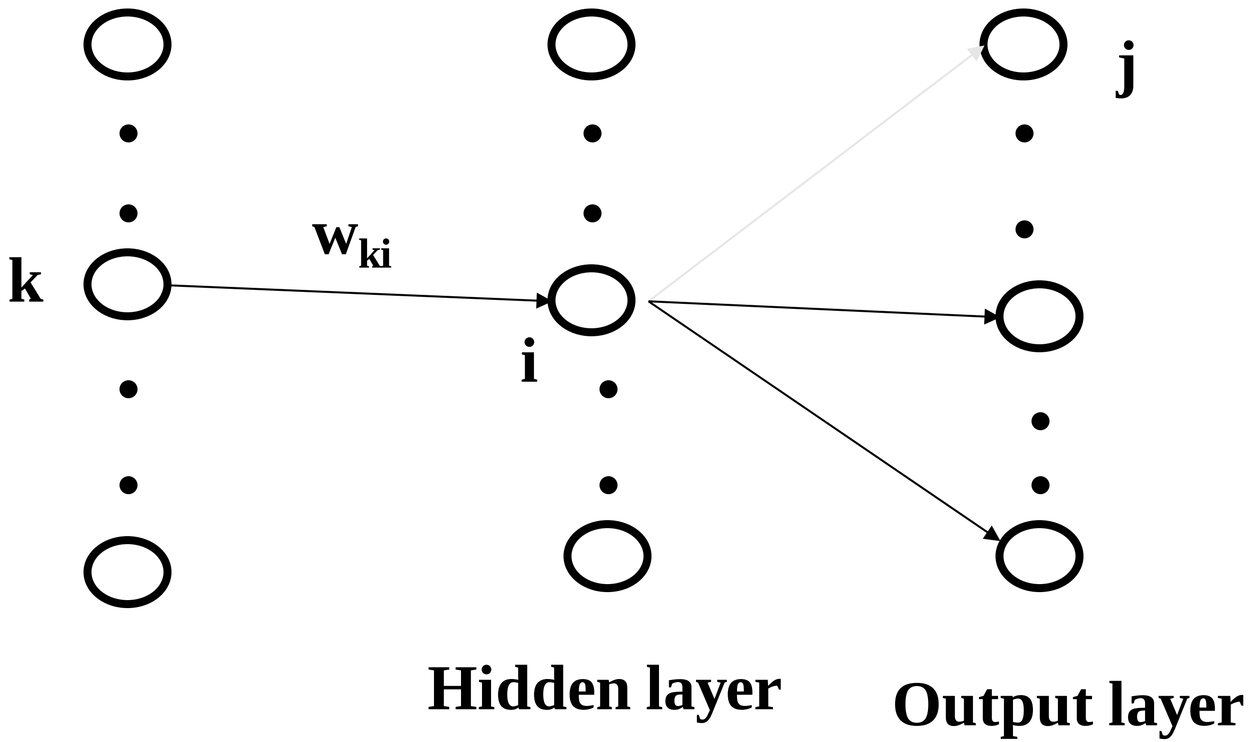
The formula for hidden layer weights update is different

because

the training examples provide target values only for the network outputs, and no target values are directly available to indicate the error of hidden unit's values

NEURAL NETWORKS

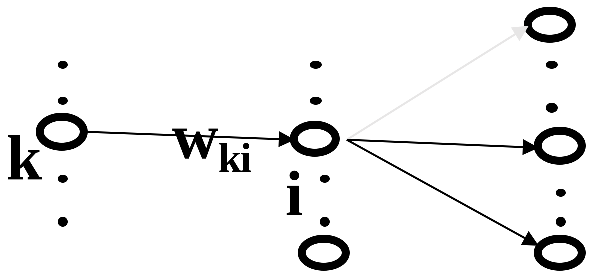
Multi Layer Perceptron: Training of Hidden Layer Weights



NEURAL NETWORKS

Multi Layer Perceptron: Training of Hidden Layer Weights

Adjustment of k^{th} weight of node "i"



$$\Delta w_{ki} = -c\ (\partial Error / \partial w_{ki})$$

Since Error is not a direct function of weight w_{ki}, therefore we use chain rule

$$\partial Error / \partial w_{ki} = (\partial Error / \partial O_i)\ .\ (\partial O_i / \partial w_{ki})$$

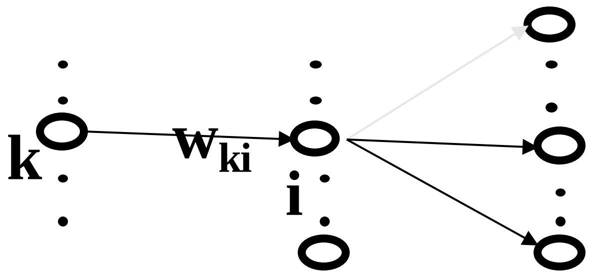
$\partial Error / \partial O_i$ = rate of change of error w.r.t output of node i

$$= \partial \textstyle\sum_j Error_j / \partial O_i$$

NEURAL NETWORKS

Multi Layer Perceptron: Training of Hidden Layer Weights

Since each $Error_j$ is independent of other $Error_j$ (each has its own independent weight set)



Hence

$$\partial \sum_j Error_j / \partial O_i = \sum_j (\partial Error_j / \partial O_i)$$

Again use chain rule we have

$$= \sum_j [(\partial Error_j / \partial act_j) . (\partial act_j / \partial O_i)]$$

NEURAL NETWORKS

Multi Layer Perceptron: Training of Hidden Layer Weights

$$\partial \sum_j \text{Error}_j / \partial O_i = \sum_j [(\partial \text{Error}_j / \partial act_j) . (\partial act_j / \partial O_i)]$$

$$\partial \text{Error}_j / \partial act_j = (\partial \text{Error}_j / \partial O_j) (\partial O_j / \partial act_j)$$

where $\partial \text{Error}_j / \partial O_j = \partial (d_j - O_j)^2 / \partial O_j = -2(d_j - O_j)$

and $\partial O_j / \partial act_j = \partial f(act_j) / \partial act_j = f'(act_j)$

$$(\partial act_j / \partial O_i) = (\partial \sum x_i w_{ij} / \partial O_i)$$

Since $O_i = x_i$

hence $\partial act_j / \partial O_i = w_{ij}$

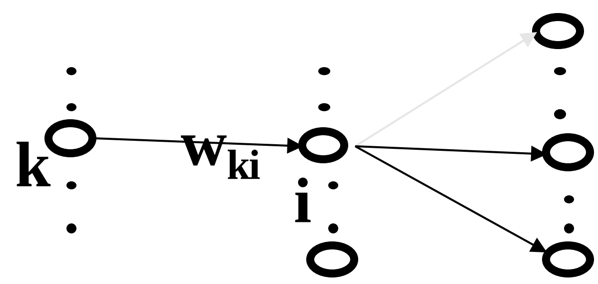
NEURAL NETWORKS

Multi Layer Perceptron: Training of Hidden Layer Weights

So we started with

$\partial Error / \partial w_{ki} = (\partial Error / \partial O_i) \cdot (\partial O_i / \partial w_{ki})$

and we have determined the first part



For the 2nd part

$\partial O_i / \partial w_{ki} = (\partial O_i / \partial act_i)(\partial act_i / \partial w_{ki})$

$(\partial act_i / \partial w_{ki}) = (\partial \sum_k x_k w_{ki} / \partial w_{ki}) = x_k$

$(\partial O_i / \partial act_i) = (\partial f(act)_i / \partial act_i) = f'(act)_i$

Hence $\Delta w_{ki} = -c\,(\partial Error / \partial w_{ki})$

$= -c[-2\sum_j \{(d_j - O_j)\, f'(act_j)\, w_{ij}\} f'(act)_i\, x_k]$

NEURAL NETWORKS

Multi Layer Perceptron: Training

A typical activation function is logistic function (which is a type of sigmoidal function)

$$f(act) = 1/(1 + e^{-\lambda act})$$

If value of λ (squashing parameter) is large we have a unit step function, if it is small we have almost a straight line between two saturation limits

$$f'(act) = f(act)(1 - f(act))$$

NEURAL NETWORKS

Multi Layer Perceptron: Training

This approach is called "gradient descent learning"

Requirement of this approach is that the activation function must be differentiable (i.e. continuous)

The number of input and output neurons are fixed

But the selection of number of hidden layers and the number of neurons in the hidden layers is done by trial and error

NEURAL NETWORKS

Multi Layer Perceptron: Training

The gradient descent is not guaranteed to converge to the global optimum

The algorithm we have discussed is the incremental gradient descent (or stochastic gradient descent) version of the Backpropagation

NEURAL NETWORKS

Multi Layer Perceptron: Face Recognition Example

Images of 20 different people

32 images per person

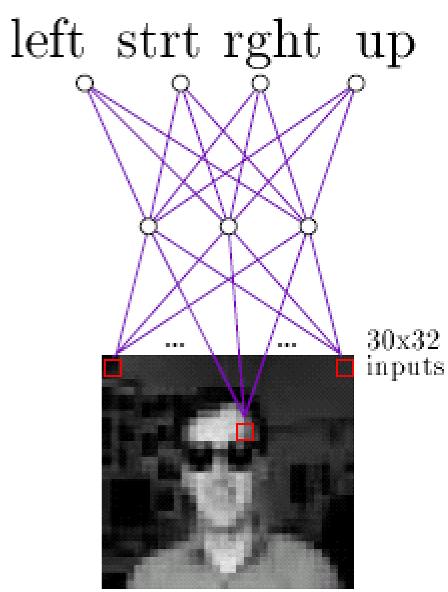
With varying expressions (happy, sad, angry, neutral)
and
looking in various directions (left, right, straight, up)
and
with and without sunglasses

Grayscale images (intensity between 0 to 255) and
size (resolution) of 120 x 128 pixels

NEURAL NETWORKS

Multi Layer Perceptron: Face Recognition Example

Typical input images



NEURAL NETWORKS

Multi Layer Perceptron: Face Recognition Example

An ANN can be trained on any one of a variety of target functions using this image data, e.g.

- **identity of a person**
- **direction in which person is looking**
- **whether or not they are wearing sunglasses**

NEURAL NETWORKS

Multi Layer Perceptron: Face Recognition Example

Design Choices:

Separate the data into

training (260 images) and test sets (364 images)

Input Encoding

- **30 x 32 pixel image**
- **A coarse resolution of the 120 x 128 pixel image**
- **Every 4 x 4 pixels are replaced by their mean value**
- **The pixel intensity is linearly scaled from 0 to 1 so that inputs, hidden units and output units have the same range**

NEURAL NETWORKS

Multi Layer Perceptron: Face Recognition Example

Design Choices:

Output Encoding

- **Learning Task: Direction in which person is looking**
- **Only one neuron could have been used with outputs 0.2, 0.4, 0.6, and 0.8 to encode the four possible values**
- **But we use 4 output neurons, so that measure of confidence in the ANN's decision can be obtained**
- **Output vector:**

  1 for true & 0 for false; e.g. [1, 0, 0, 0]

Multi Layer Perceptron: Face Recognition Example

Design Choices:

Network Structure

- **How many Layers?**
  - **Usually one hidden layer is enough**
- **How many units in the hidden layer**
  - **More than necessary units result in over-fitting**
  - **Less units result in failure of training**
  - **Trial & error: Start with a number and prune the units with the help of a cross-validation set**

Reading Assignment & References

1. Section 10.1.1, 10.2.1 and 10.2.2
from George F. Luger
2. Machine Learning by Tom Mitchell

http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/systems/nevprop/np.c