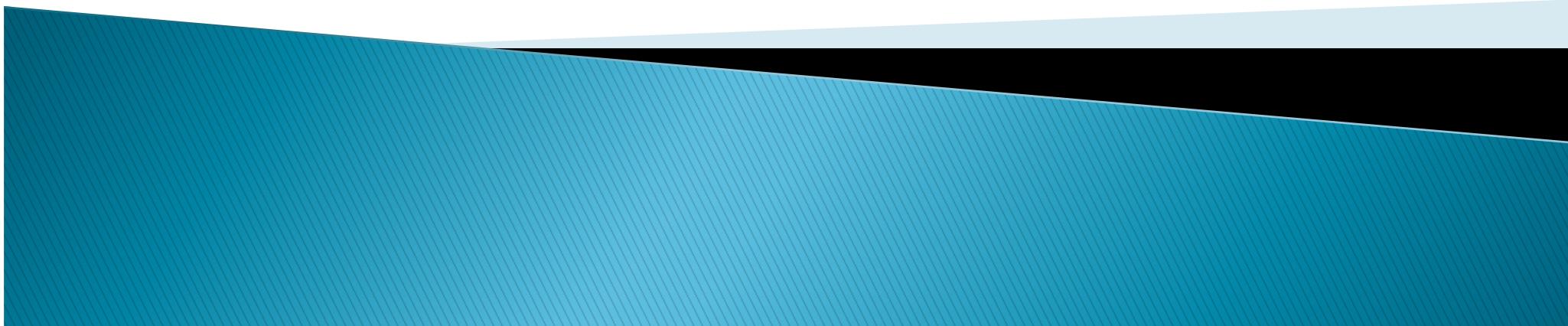


# Artificial Neural Networks



# Learning Outcomes

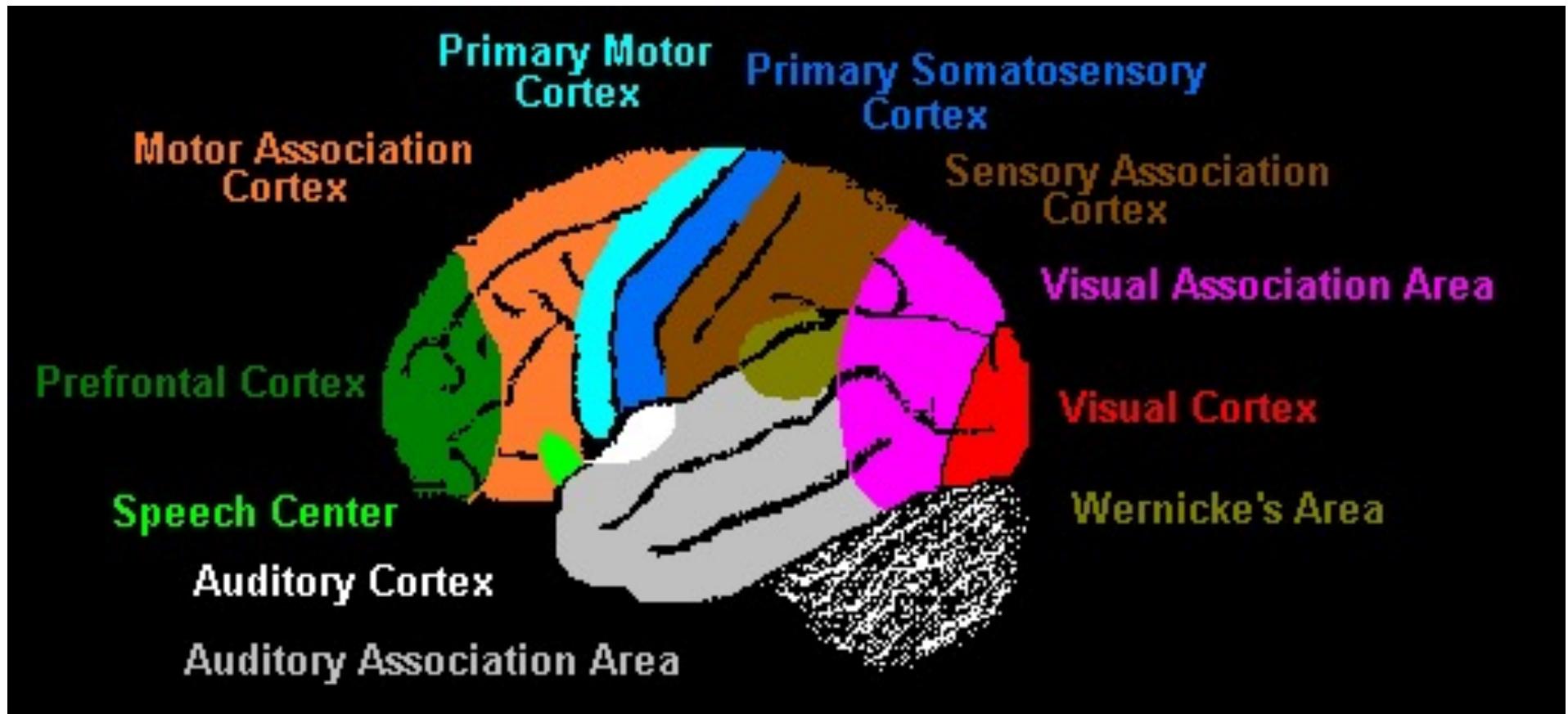
- ▶ Examine the basic principles of artificial neural networks.
- ▶ Discuss the operation of the Multi Layer Perceptron through the use of suitable examples.
- ▶ Discuss the derivation of the weight update formula through the use of *backpropagation*.



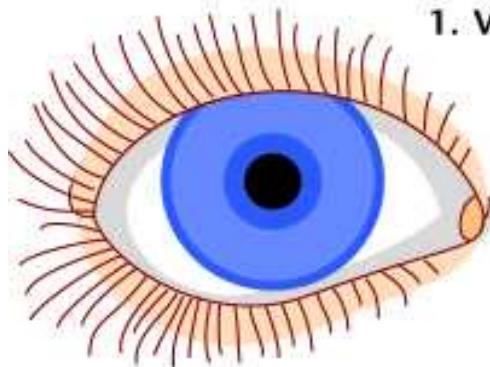
# Neural Networks

- ▶ Biologically inspired family of algorithms that is inspired by the human brain
- ▶ Neural Networks are used for *classification*, *clustering* and *numeric prediction* tasks.
- ▶ Most popular types are
  - Multi Layer Perceptron (MLP) used for classification
  - Radial Basis Function (RBF) used for classification and numeric prediction
  - Self Organizing Map (SOM) used for clustering
  - Convolutional Neural Network (CNN) used for image classification
  - Long Short Term Memory (LSTM) used for modelling time series

# Sensory Areas of the Human Brain



# Brain performs classifications, predictions and associations

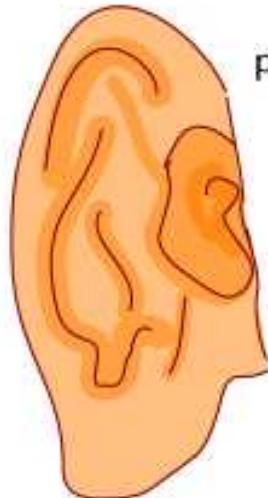


## 1. Vision and seeing

Hue, brightness  
saturation (purity)

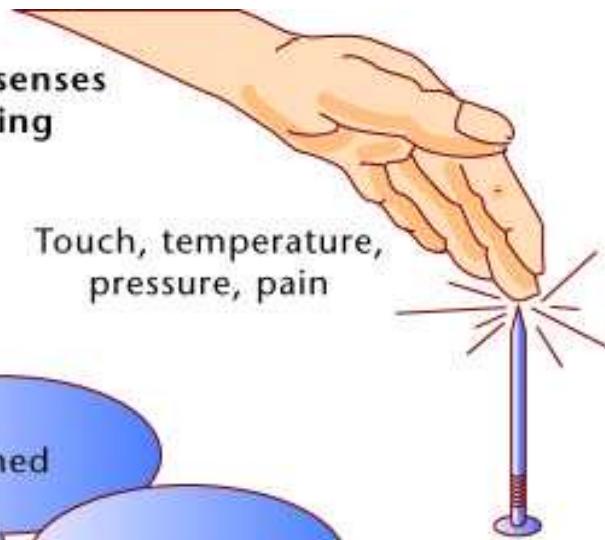
## 2. Audition and hearing

Volume,  
pitch (frequency),  
tonal quality  
(timbre or  
purity)



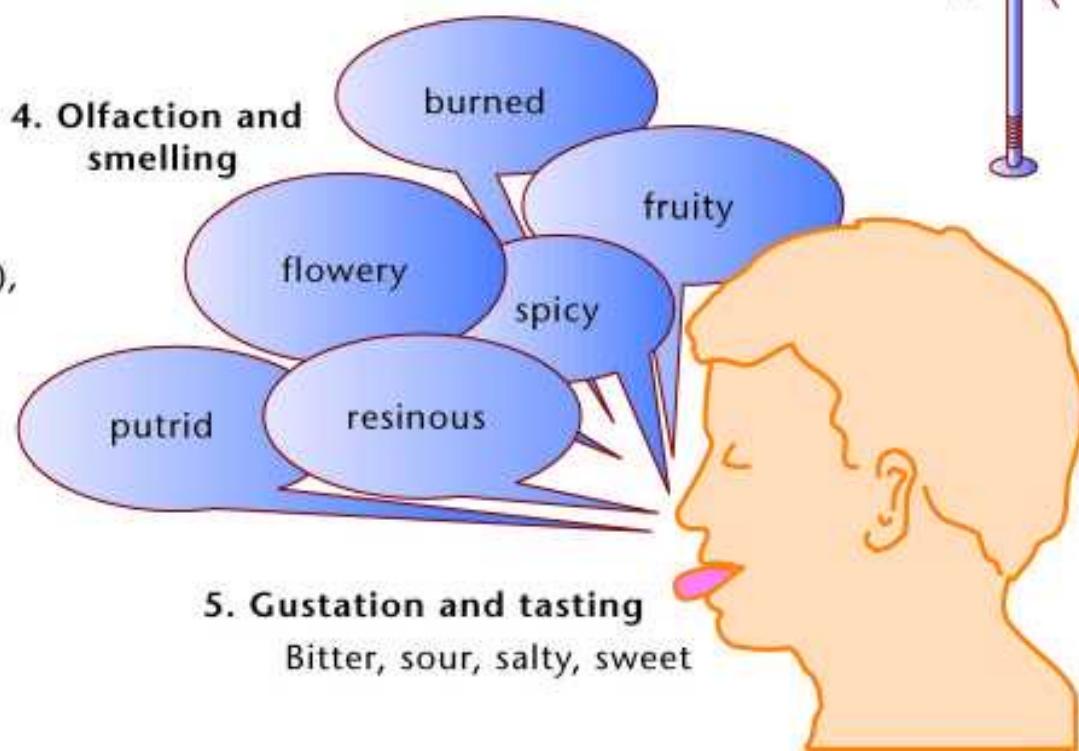
## 3. The skin senses and feeling

Touch, temperature,  
pressure, pain



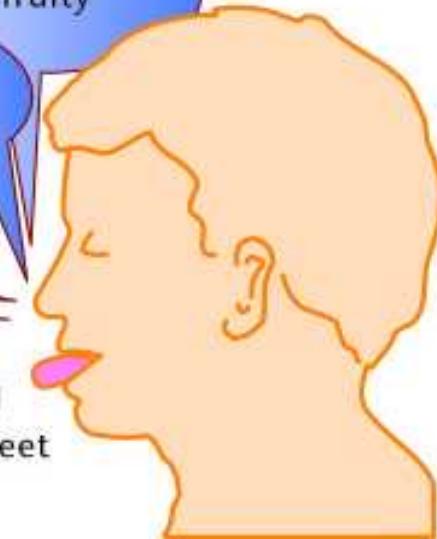
## 4. Olfaction and smelling

burned  
fruity  
flowery  
spicy  
putrid  
resinous

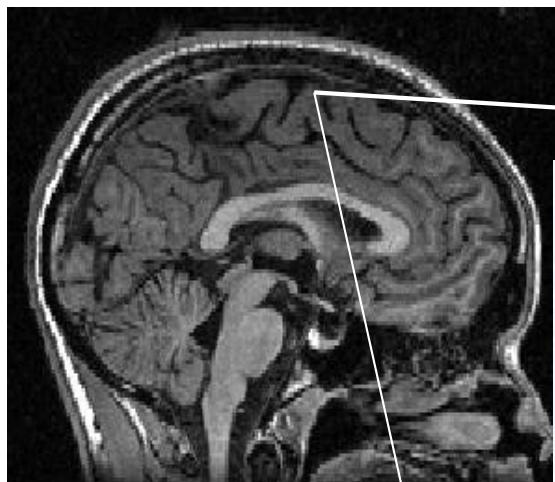


## 5. Gustation and tasting

Bitter, sour, salty, sweet

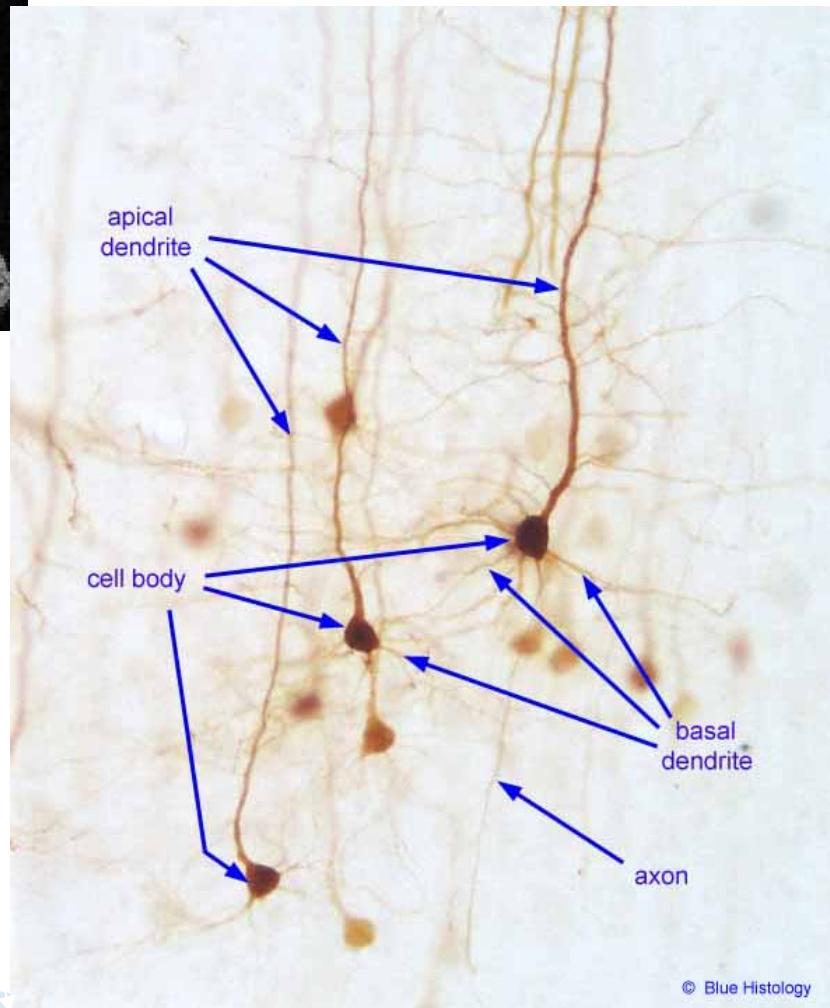


# Neural Networks

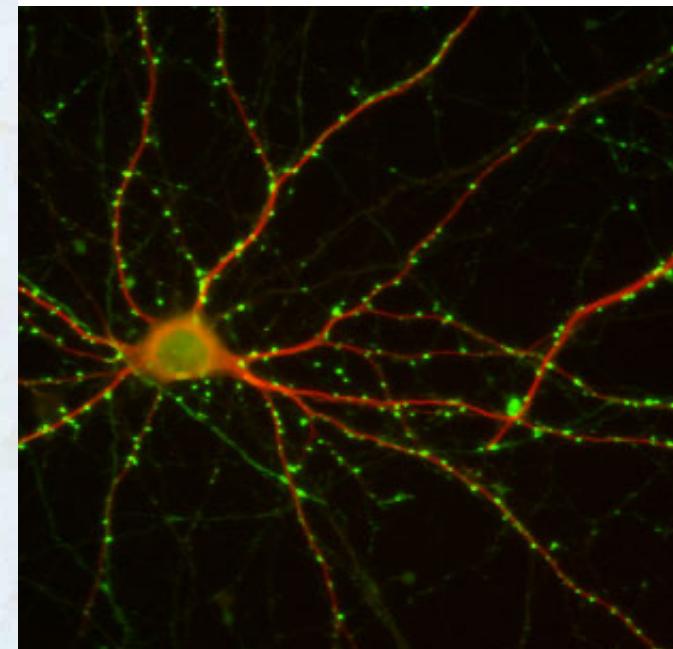


Huge complexity

$10^{11}$  of neurons  
in the brain

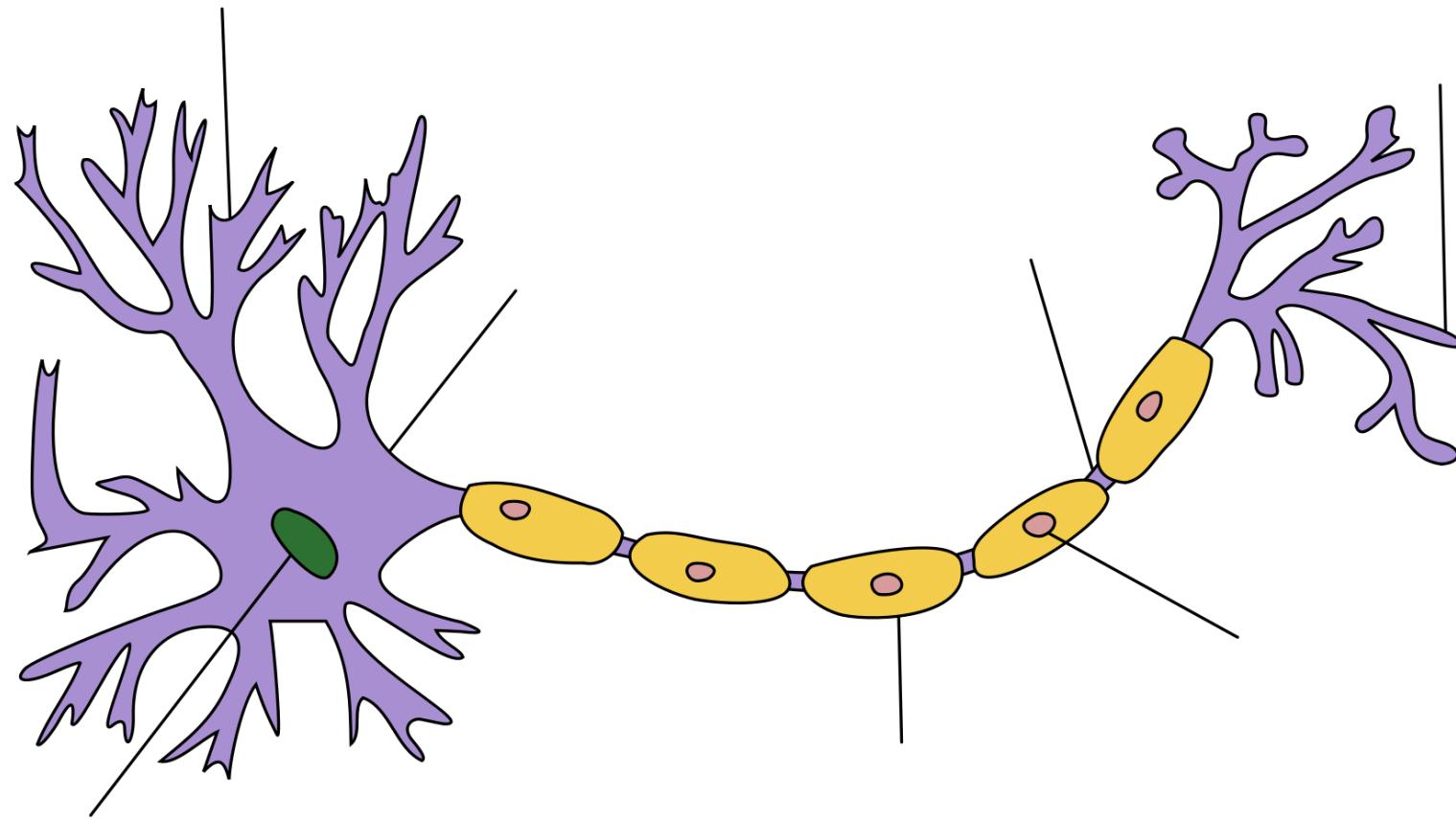


© Blue Histology



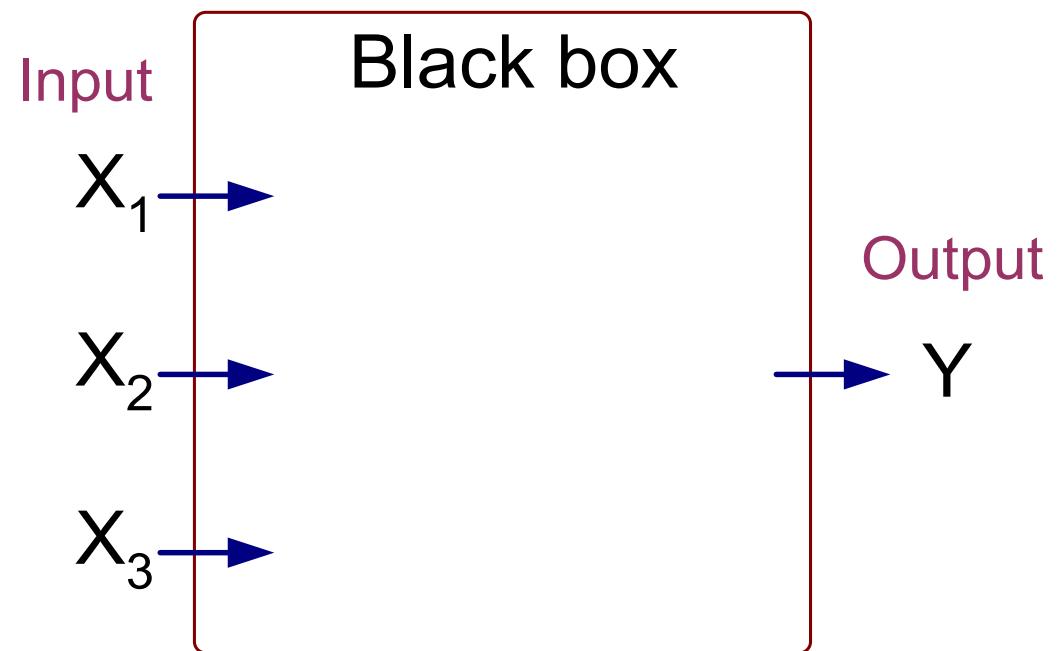
Huge connectivity:  
each neuron sends  
and receives  $10^4$  of  
synapses (contacts)

# Neuron Interconnectivity



# Artificial Neural Networks (ANN)

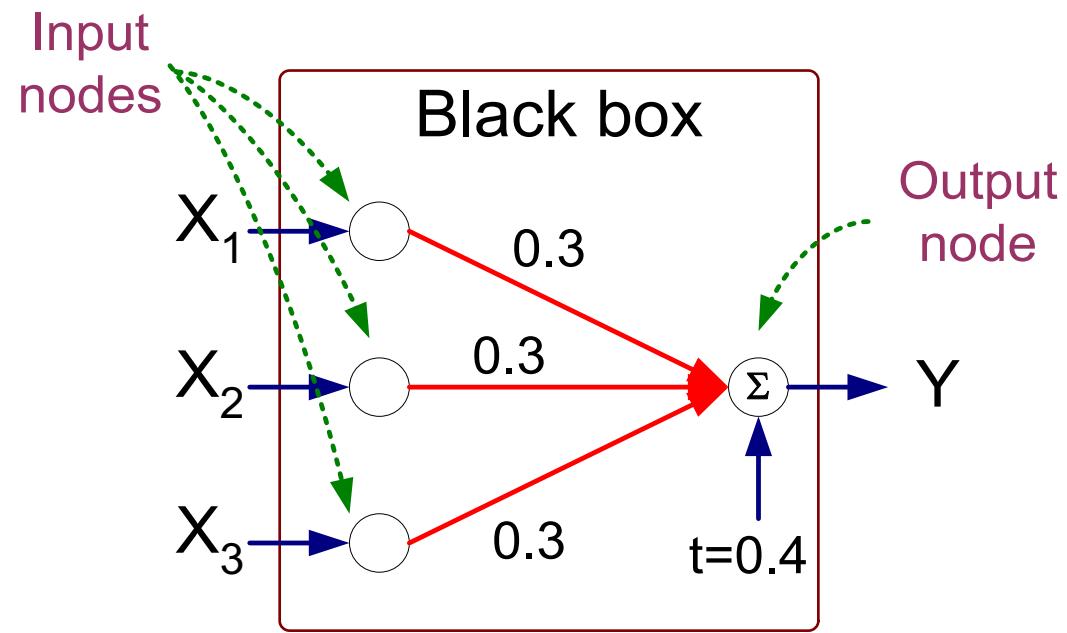
$X_1$	$X_2$	$X_3$	$Y$
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



Output  $Y$  is 1 if at least two of the three inputs are equal to 1.

# Artificial Neural Networks (ANN)

$X_1$	$X_2$	$X_3$	$Y$
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0

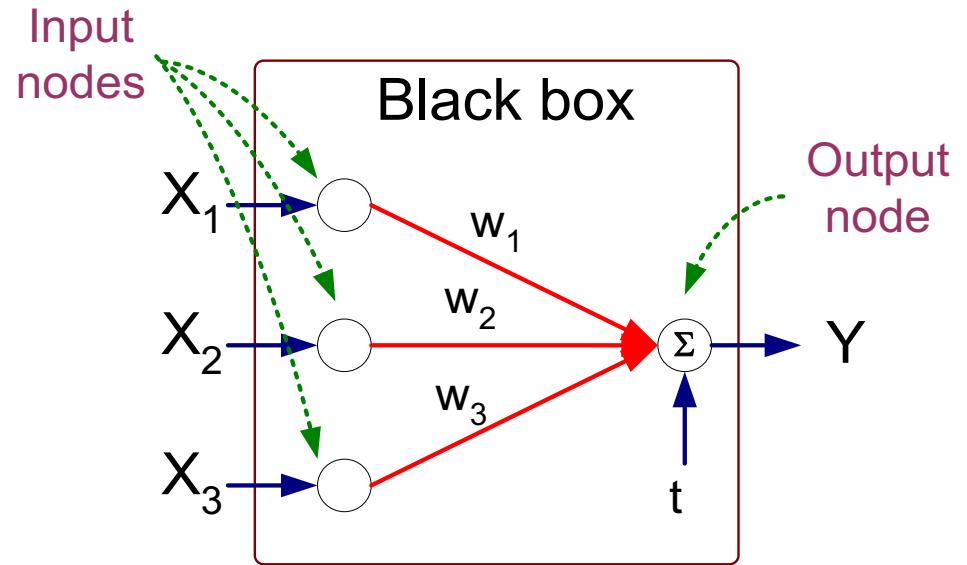


$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

where  $I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$

# Artificial Neural Networks (ANN)

- ▶ Model is an assembly of inter-connected nodes and weighted links
- ▶ Output node sums up each of its input value according to the weights of its links
- ▶ Compare output node against some threshold t



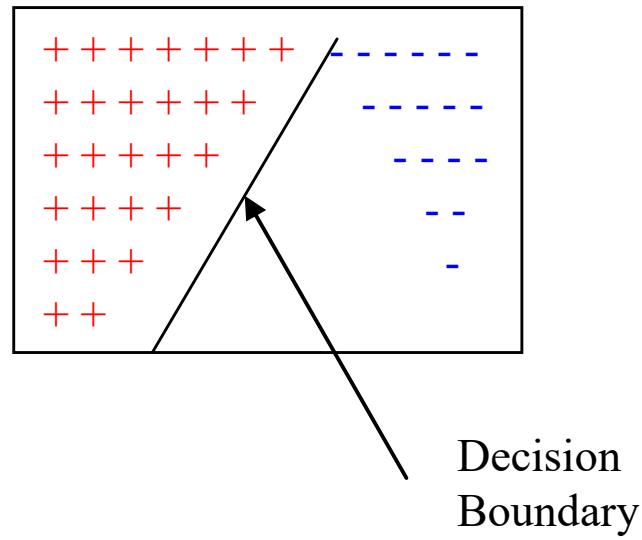
## Perceptron Model

$$Y = I(\sum_i w_i X_i - t) \quad \text{or}$$

$$Y = \text{sign}(\sum_i w_i X_i - t)$$

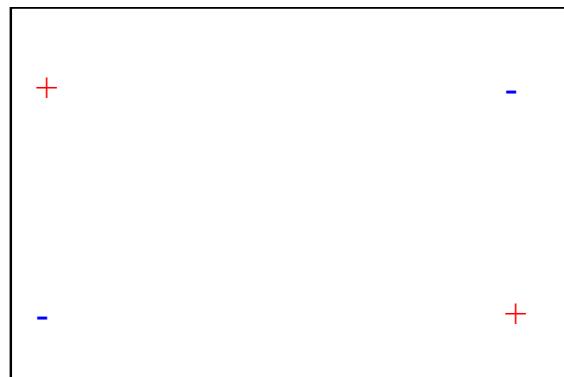
# Limitations of Simple Perceptrons

- ▶ Simple perceptrons can be used to classify problems which are linearly separable
- ▶ For such problems a single line can be drawn which separates the two classes with zero (or near zero) error



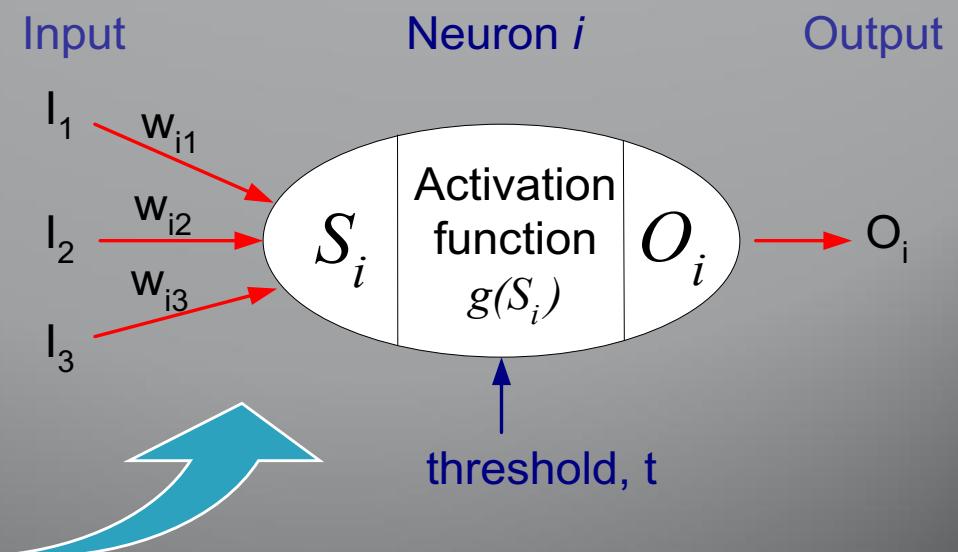
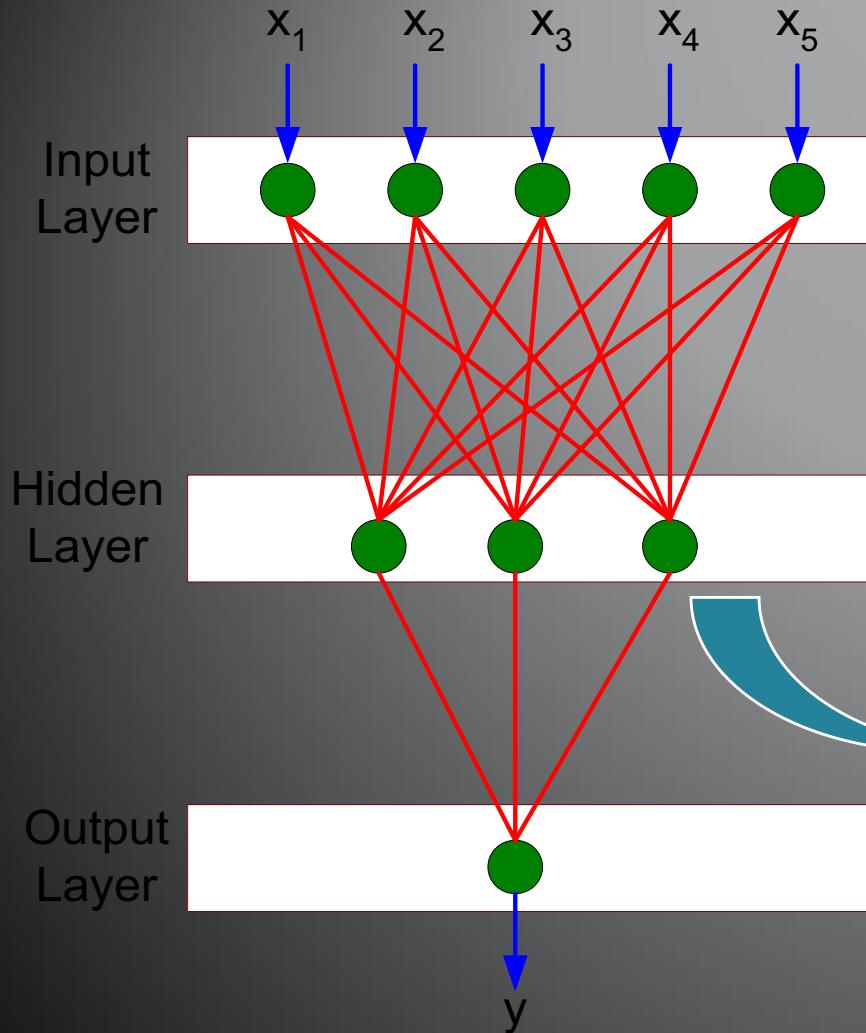
# Limitations of Simple Perceptrons

- ▶ However simple perceptrons cannot solve non linear classification problems such as the XOR problem



- ▶ These types of problems can only be solved by adding another layer (called the hidden layer) of neurons to the network

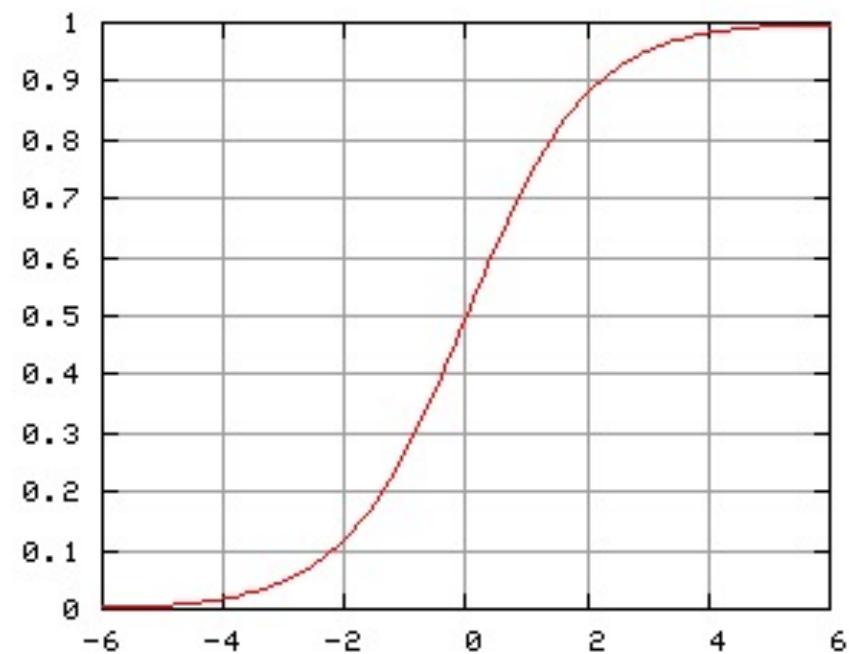
# General Structure of ANN



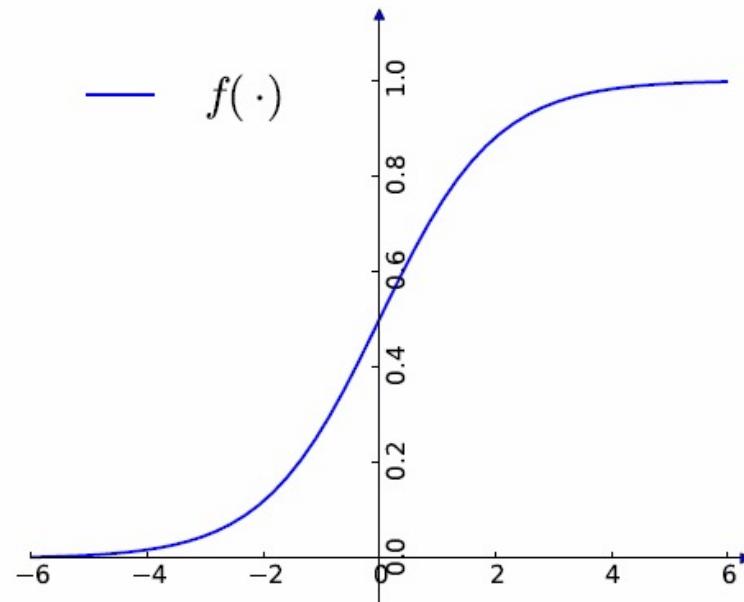
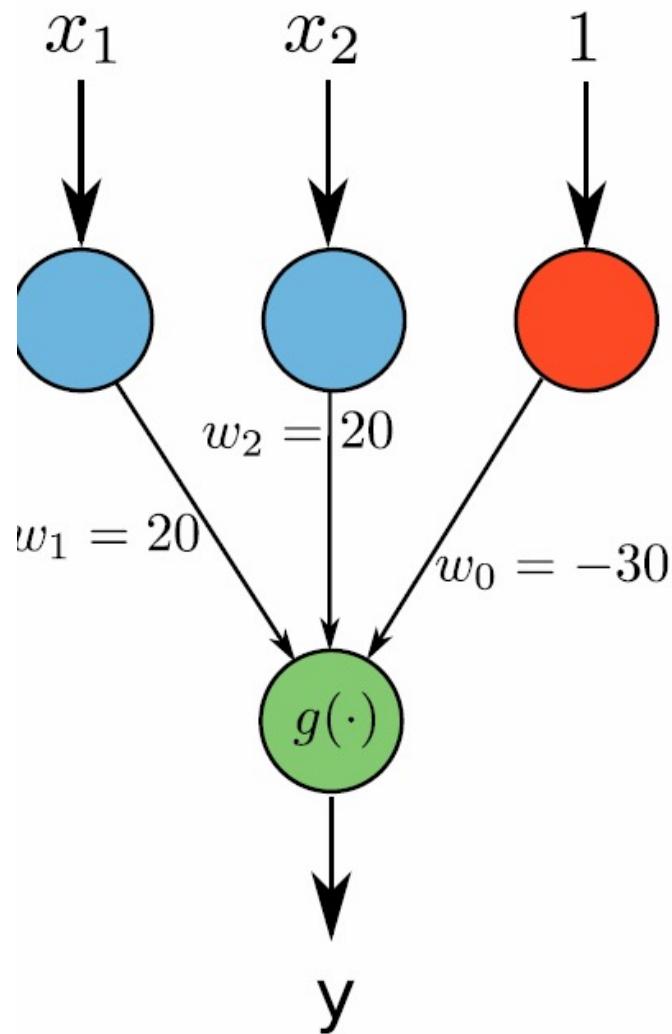
Training ANN means learning  
the weights of the neurons

# Activation Functions

- ▶ Just as with human neurons, the neurons in the hidden layer are activated only when the input they receive is above a certain threshold value
- ▶ To model this situation the sigmoid function is commonly used as an activation function
- ▶  $\sigma(x) = \frac{1}{1+e^{-x}}$



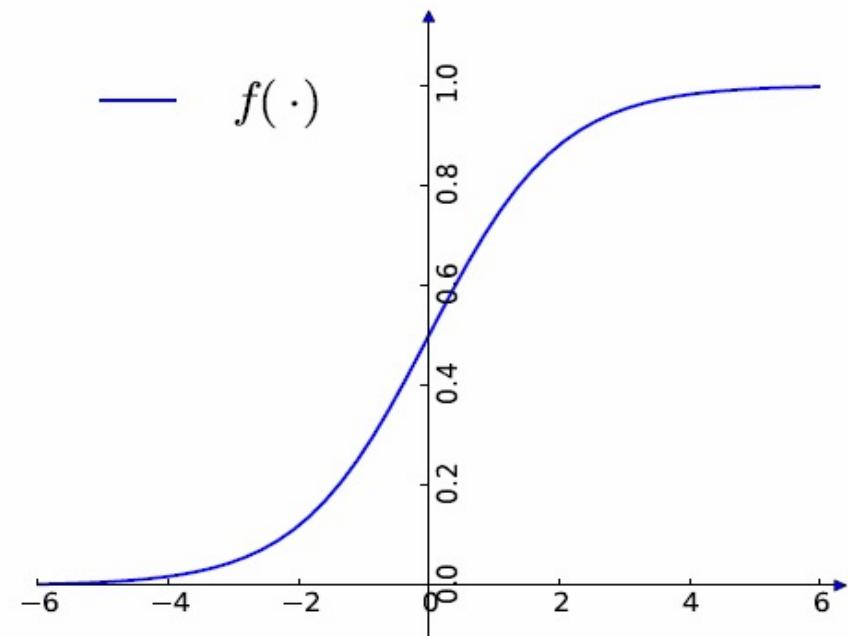
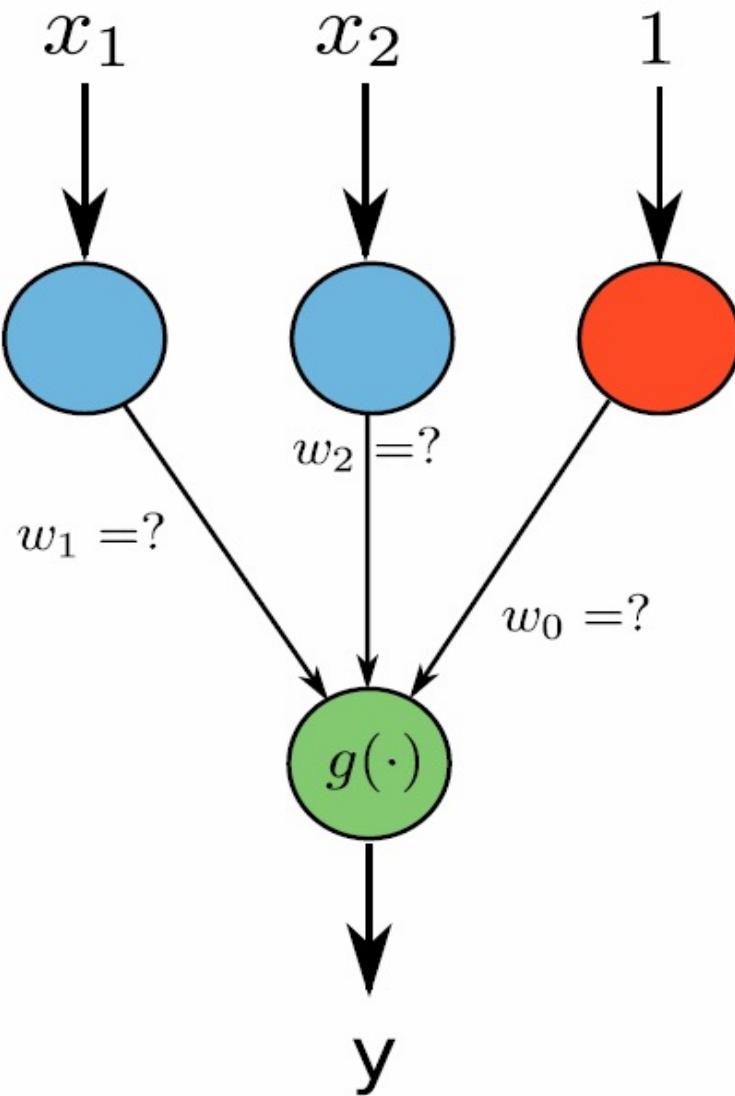
# Solving the Logical AND Problem



$x_1$	$x_2$	$y$	desired
0	0	$f(-30) \approx 0$	0
0	1	$f(-10) \approx 0$	0
1	0	$f(-10) \approx 0$	0
1	1	$f(10) \approx 1$	1

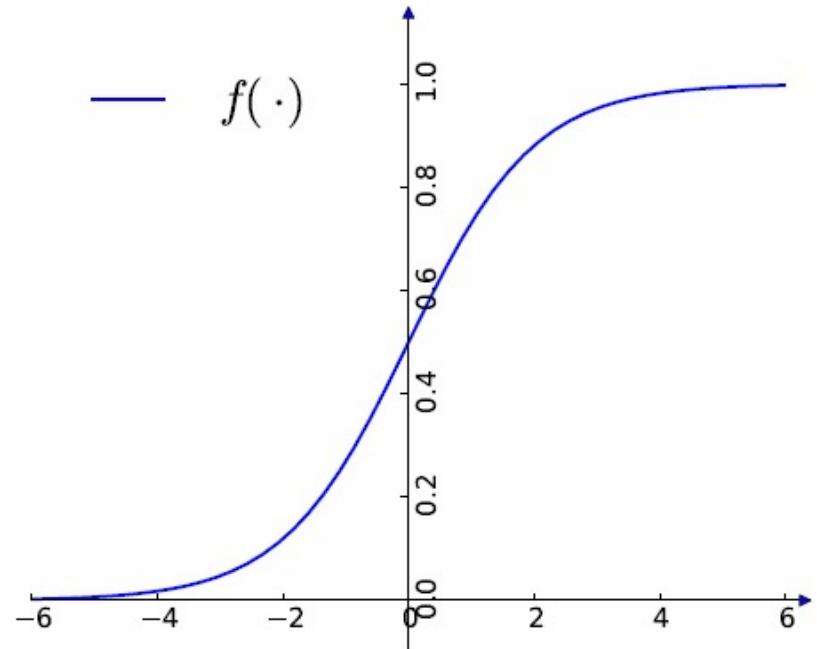
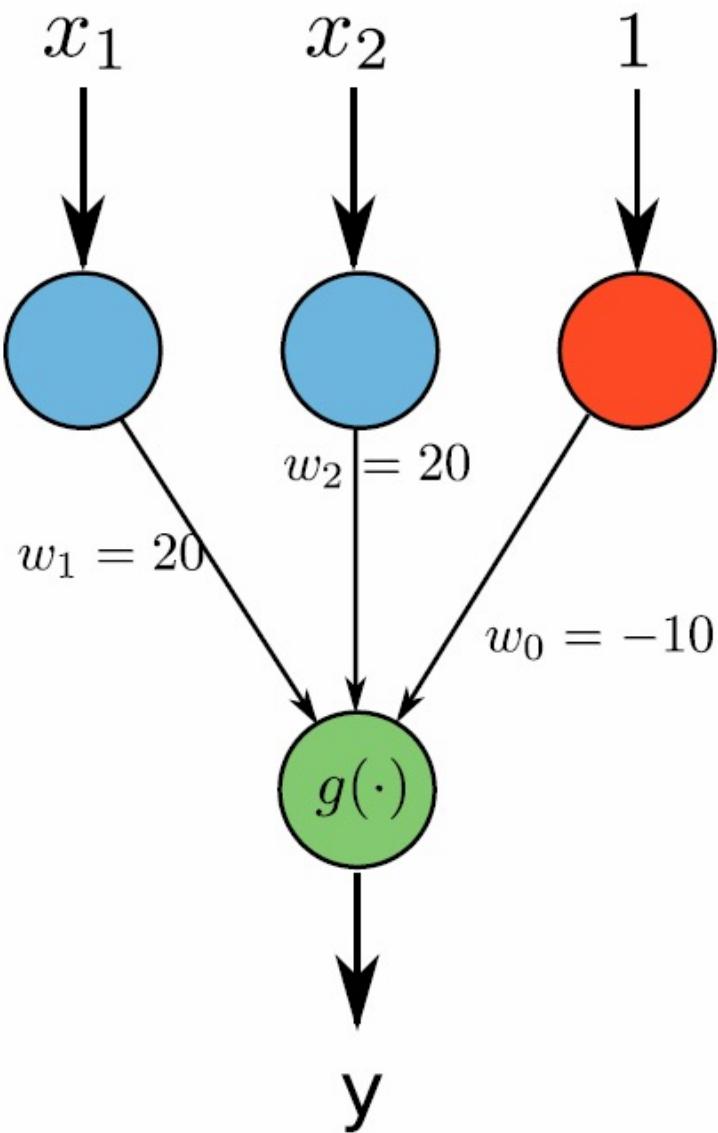
In this and the next 4 slides the functions  $f$  and the sigmoid are one and the same

# Solving the Logical OR Problem



$x_1$	$x_2$	$y$	desired
0	0	?	0
0	1	?	1
1	0	?	1
1	1	?	1

# Solving the Logical OR Problem



$x_1$	$x_2$	$y$	desired
0	0	$f(-10) \approx 0$	0
0	1	$f(10) \approx 1$	1
1	0	$f(10) \approx 1$	1
1	1	$f(30) \approx 1$	1

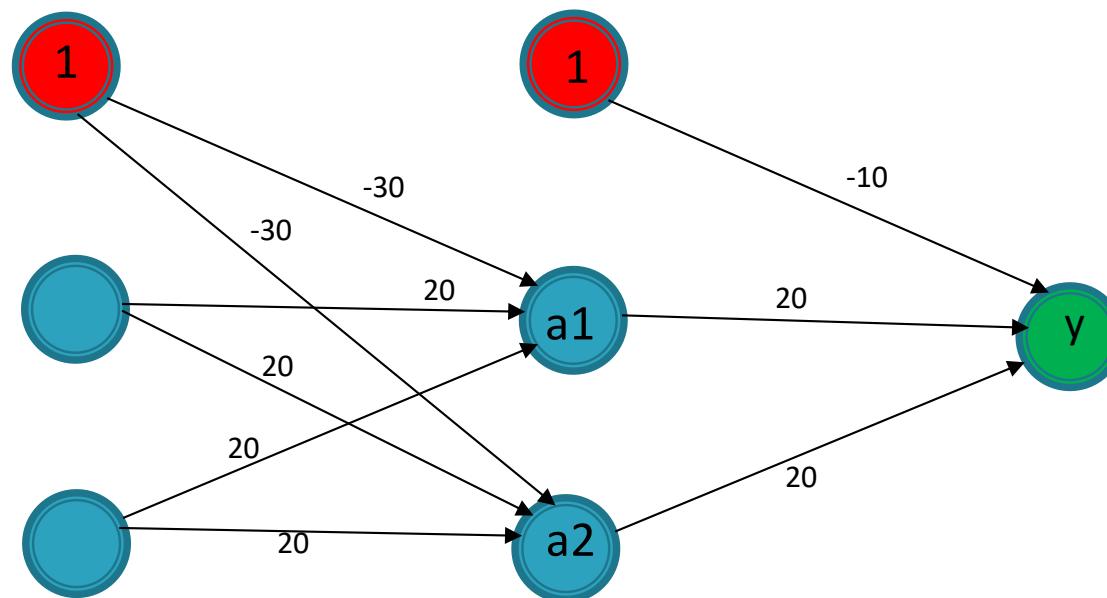
# Solving the Logical XNOR Problem

- ▶ The XNOR problem is more difficult than the logical AND problem.
- ▶ It cannot be solved by a single neuron as it is a 2 stage process
- ▶  $(X_1 \text{ XNOR } X_2) = a_1 \text{ OR } a_2$  where  $a_1 = (X_1 \text{ AND } X_2)$  and  $a_2 = (\text{NOT } X_1 \text{ AND NOT } X_2)$
- ▶ This can be seen from the following truth table

X1	X2	a1	a2	a1 OR a2	X1 XNOR X2
0	0	0	1	1	1
0	1	0	0	0	0
1	0	0	0	0	0
1	1	1	0	1	1

# Neural Net for Solving the Logical XNOR Problem

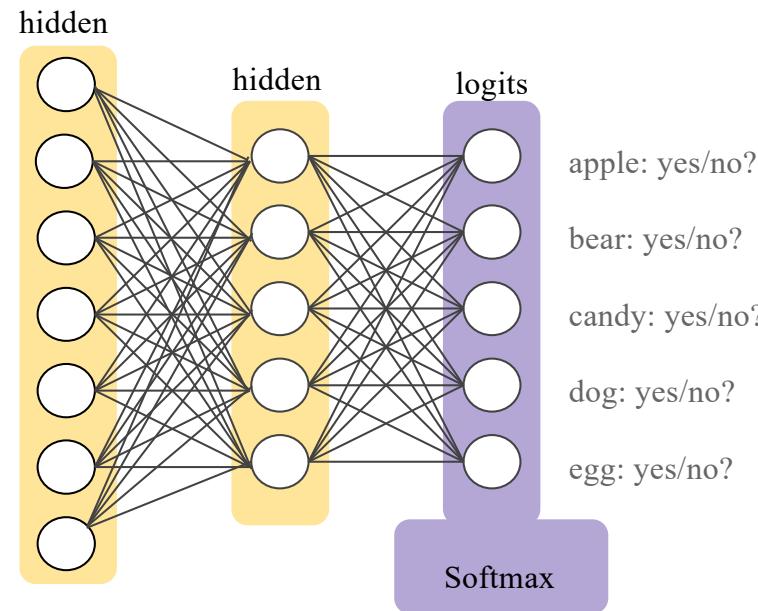
- ▶  $a_1$  and  $a_2$  can be computed in parallel and so 2 neurons can be assigned to do the computation in the hidden (intermediate layer).



# Solving classification problems with Softmax

- Classification problems involving more than two classes are solved through the Softmax function which is implemented as an additional layer

$$p(y = j|\mathbf{x}) = \frac{e^{(\mathbf{w}_j^T \mathbf{x} + b_j)}}{\sum_{k \in K} e^{(\mathbf{w}_k^T \mathbf{x} + b_k)}}$$

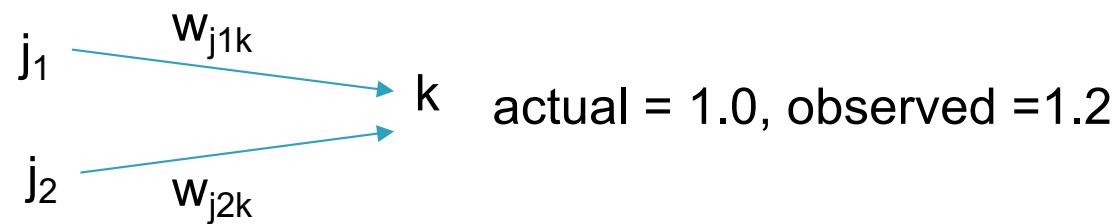


# General Algorithm for learning ANN

- ▶ Initialize the weights ( $w_0, w_1, \dots, w_k$ )
- ▶ Compute the error at each output node (k), and the hidden node (j) connected to it.
- ▶ Now adjust the weights  $w_{jk}$  such that  
 $w_{jk}(\text{new}) = w_{jk}(\text{current}) + \Delta w_{jk}$   
where  $\Delta w_{jk} = r \text{Error}(k) O_j$   
 $r$  = learning rate parameter ( $0 < r < 1$ )  
 $\text{Error}(k)$  = the computed error at node k  
 $O$  = output of node j

# Algorithm for learning ANN

- ▶ Thus it can be seen that the observed errors are used to adjust the weights so that the overall error is minimized
- ▶ For example if the desired output at node k is 1 and the actual output is 1.2, then the error =  $(1 - 1.2) = -.2$ , so we need to decrease the weight of all incoming links starting from all nodes (e.g.  $j_1, j_2$ ) that feed into node k



- ▶ The weight adjustment process is done iteratively until the error is below some specified threshold – this will involve scanning the data many times over

# Backpropagation learning

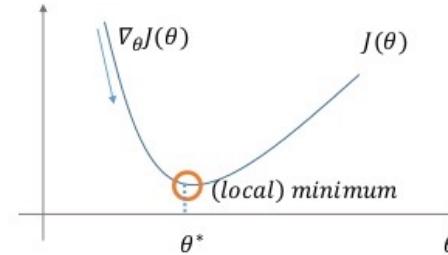
- ▶ A rigorous derivation of the weight update expression using the method of *gradient descent* available from: [backprop.pdf](#)
- ▶ Gradient descent is commonly used for a minimizing a function

## Gradient Descent

- Gradient descent is a way to minimize an objective function  $J(\theta)$ 
  - $J(\theta)$ : Objective function
  - $\theta \in R^d$ : Model's parameters
  - $\eta$  : Learning rate. This determines the size of the steps we take to reach a (local) minimum.

Update equation

$$\theta = \theta - \eta * \nabla_{\theta} J(\theta)$$



# The Loss function in Backpropagation Learning

- ▶ Backpropagation uses gradient descent with Loss as the objective function to minimize.
- ▶ The Loss L is defined as follows:
  - ▶  $L = \frac{\sum_{i=1}^k (E_i - P_i)^2}{k}$ , thus the loss value is the average squared difference between the expected value  $E_i$  (i.e. the actual class value) at the output node i and the predicted value  $P_i$  at that node.
  - ▶ The loss is therefore a measure of error but is not exactly the same as classification error
  - ▶ Python supports the computation of the loss during the training phase of the MLPClassifier – see the sklearn documentation online

# Major Parameters for Multi Layer Perceptrons

1. *Learning rate* – this determines the size of the “steps taken” in the weight adjustment process – larger steps means learning takes place quicker but accuracy may suffer
2. *Number of epochs* – the number of times that the training dataset is scanned – larger the value the more accurate the model (generally 100 or more)
3. The *number of hidden neurons* used – generally chosen as  $(\text{attributes} + \text{classes})/2$
4. *Momentum* – some implementations add a term called the momentum to the current weight – this is a small fraction of the update value from the previous iteration; the momentum makes the learning process smoother

# Neural Networks – Strengths

- ▶ Neural Nets work well with datasets containing noise
- ▶ Have consistently good accuracy rates across several domains
- ▶ Can be used for both supervised (classification and numeric prediction) as well as unsupervised learning



# Neural Networks – Weaknesses

- ▶ Lack the ability to explain their behaviour (unlike Decision Trees and Naïve Bayes)
- ▶ In some cases, overtraining can cause over fitting
- ▶ With large datasets training time can be large – very much larger than the Decision Tree and Naïve Bayes methods

# Neural Network Applications

- ▶ In general can be used for classification as well as for numeric prediction
- ▶ For classification has been used for recognizing both printed and handwritten digits
- ▶ For numeric prediction has been used for forecasting time series such as weather data (temperature, pressure, wind speed, etc), stock market prices, etc.
- ▶ Neural Network handwritten recognition:

<http://myselph.de/neuralNet.html>

<http://www.sund.de/netze/applets/BPN/bpn2/ochre.html>

- ▶ Neural Nets play Pong:

<http://www.youtube.com/watch?v=LD6OgKEj5JE>