
BITI 3133

Neural Network

Lecture 6:

Multi Layer Perceptron

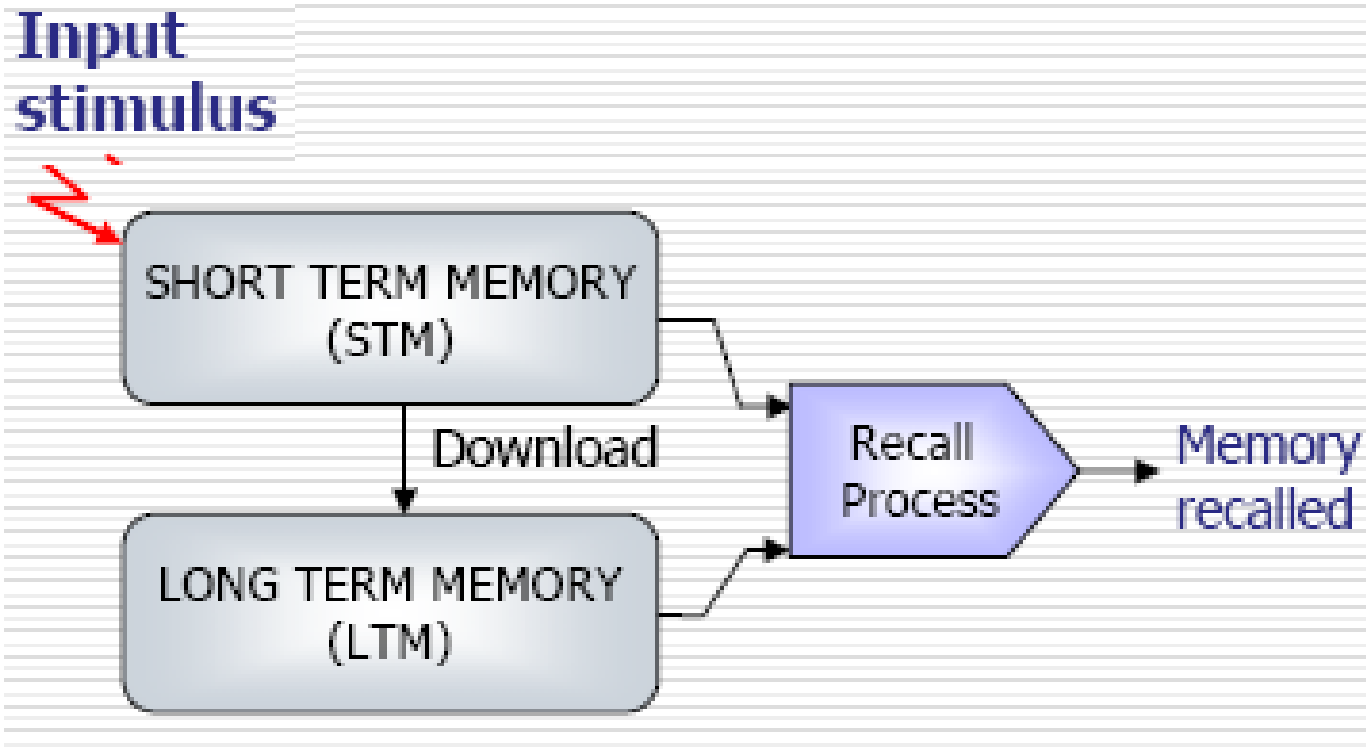
Today's Lecture

- Multi Layer Perceptron
- Backpropagation

Learning Outcomes:

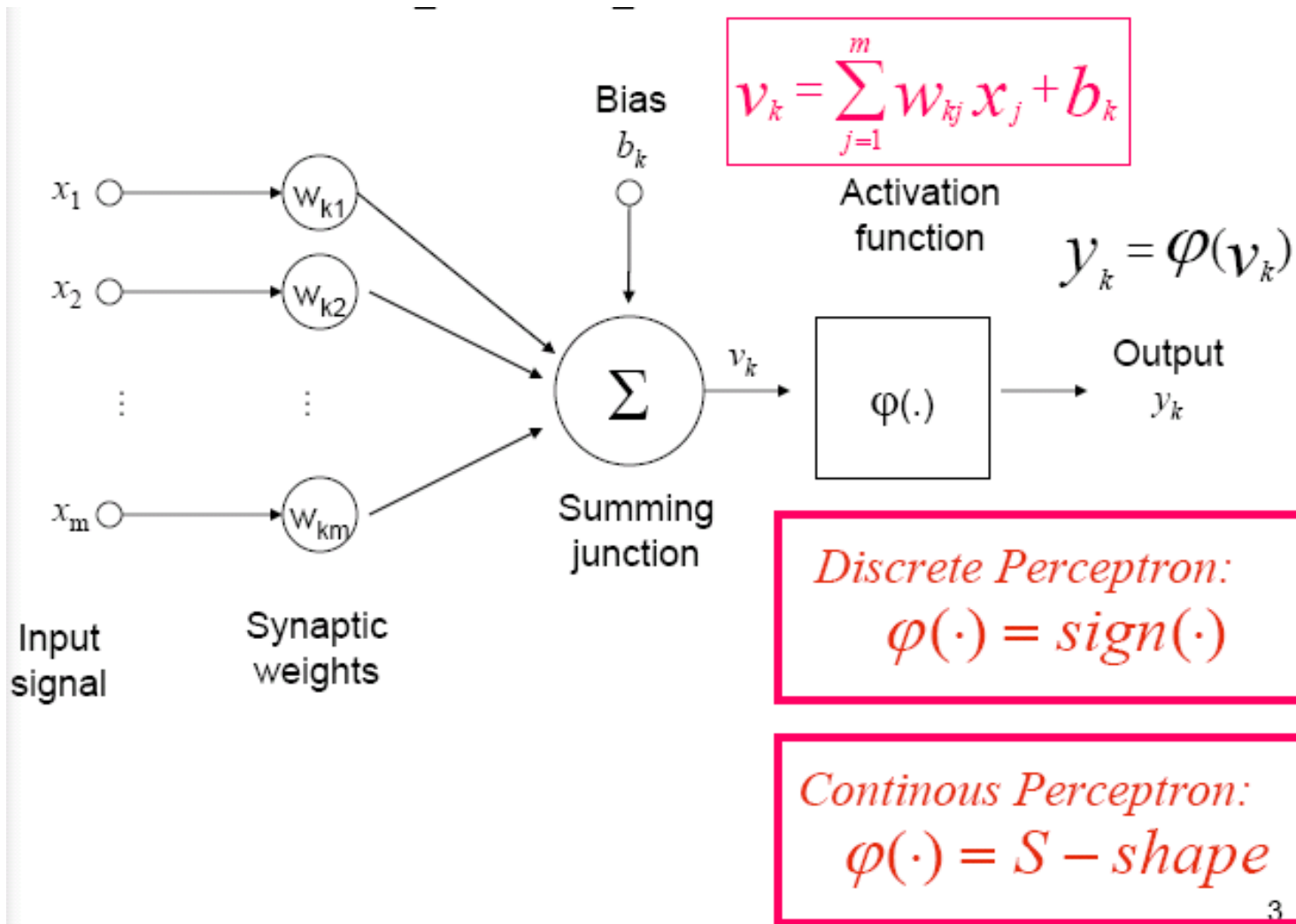
- Understand the concept of multi layer perceptron in Neural Network
- Know how to implement the backpropagation in Neural Network

Information flow in human memory

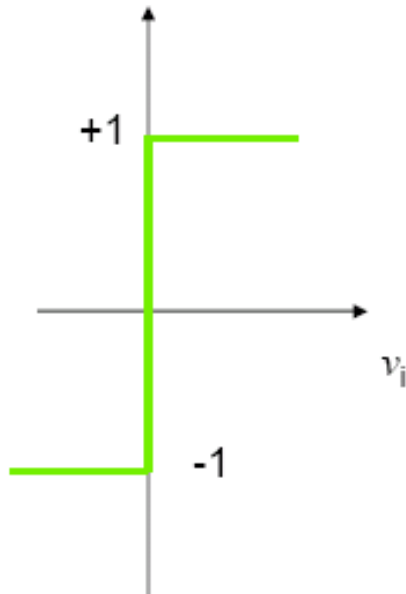


- i. Inputs to the brain are processed into short term memory.
- ii. Information is downloaded into long term memory for more permanent storage: days, months, and years. Capacity of long term memory is very large.
- iii. Memory recalled from the memories. Recall process of these memories are distinct from the memories themselves.

What is Perceptron? (Revisited)

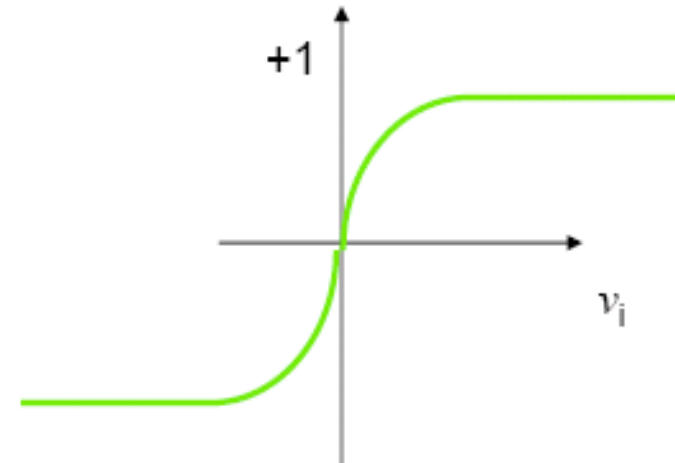


Activation Function of Perceptron



Signum Function
(sign)

Discrete Perceptron:
 $\varphi(\cdot) = \text{sign}(\cdot)$



Continuous Perceptron:

$\varphi(v) = s\text{-shape}$

What can perceptrons represent?

Input 1

Input 2

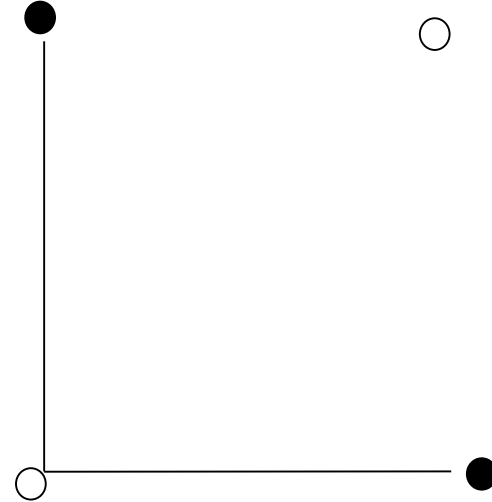
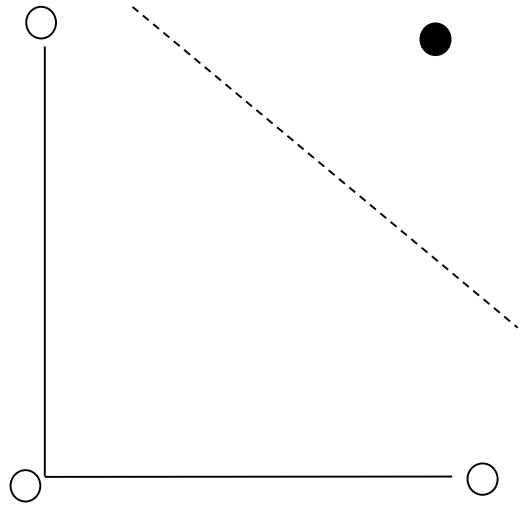
Output

AND			
0	0	1	1
0	1	0	1
0	0	0	1

XOR			
0	0	1	1
0	1	0	1
0	1	1	0

XOR Problem: One and only one can be true but one has to be true.

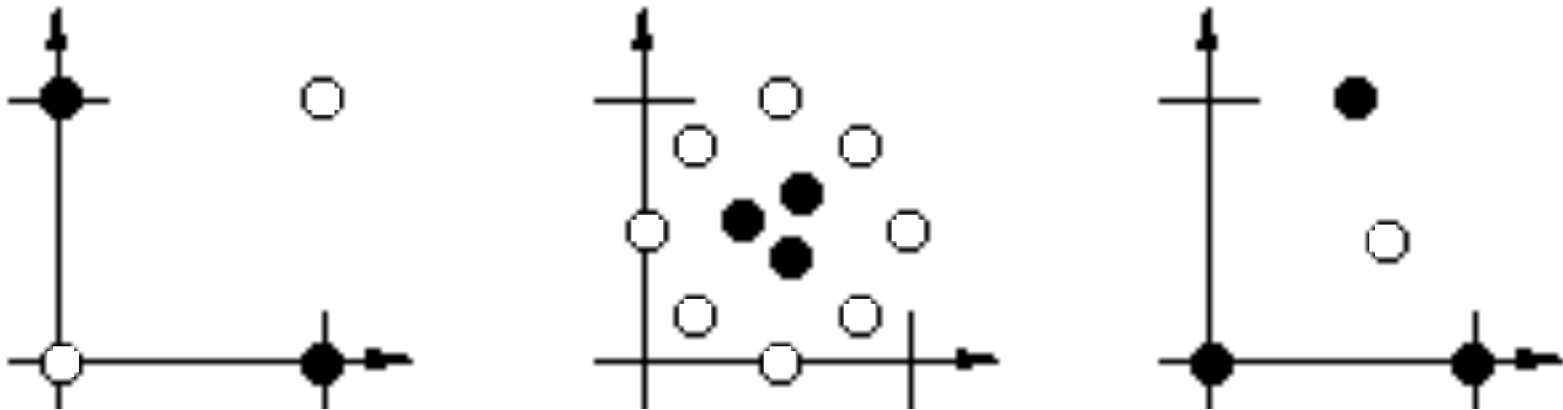
What can perceptrons represent?



- Functions which can be separated in this way are called *Linearly Separable*
- Only linearly separable functions can be represented by a perceptron
- XOR cannot be represented by a perceptron

Perceptron Limitations

Linearly Inseparable Problems



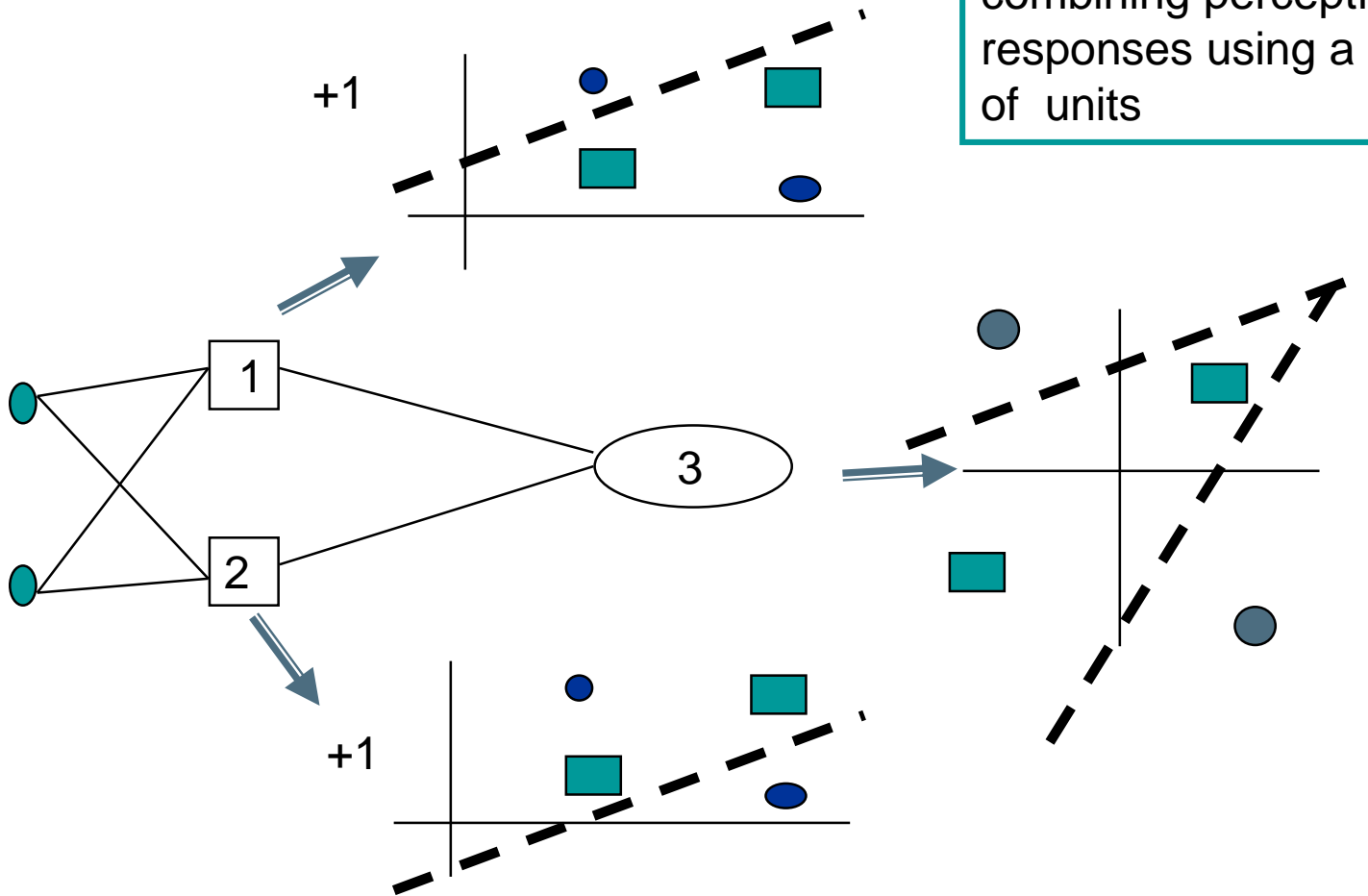
Solution is Multilayer Perceptron

Why MLP

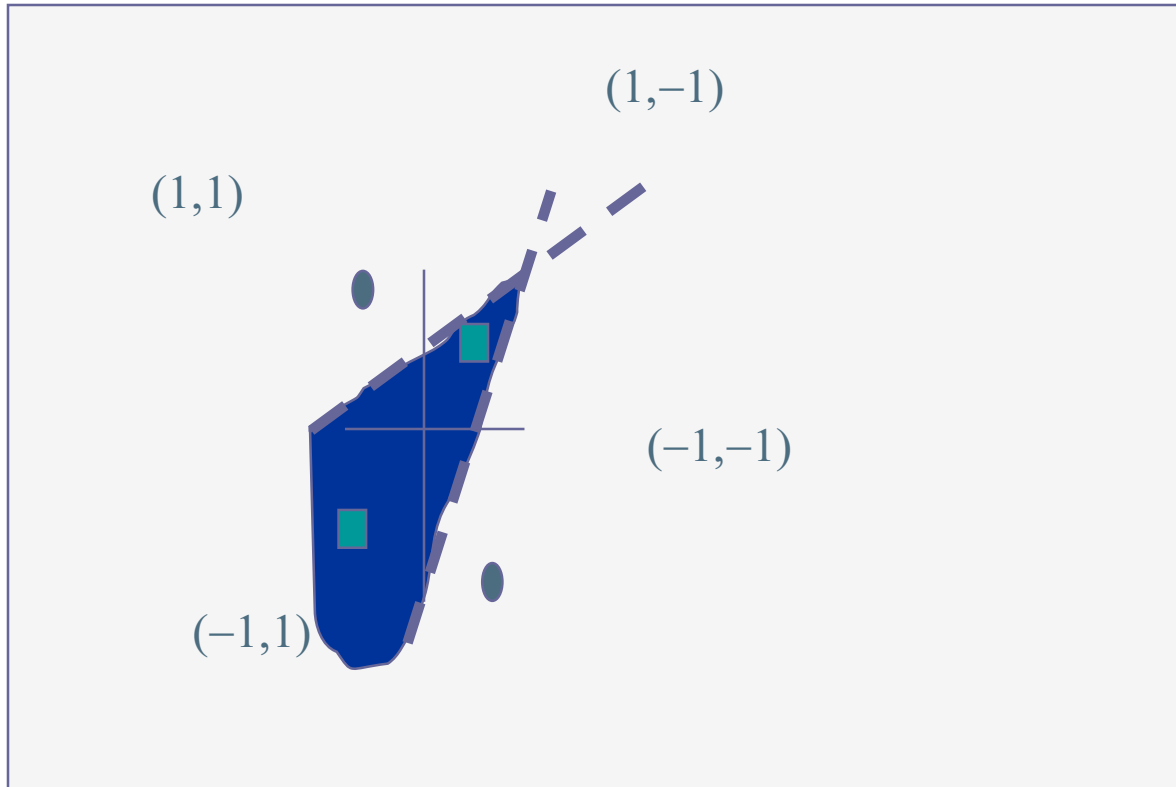
- The single-layer perceptron classifiers discussed previously can only deal with linearly separable sets of patterns.
- The multilayer networks to be introduced here are the most widespread neural network architecture
 - Made useful until the 1980s, because of lack of efficient training algorithms (McClelland and Rumelhart 1986)
 - The introduction of the backpropagation training algorithm.

Why MLP

Minsky & Papert (1969) offered solution to XOR problem by combining perceptron unit responses using a second layer of units



Why MLP

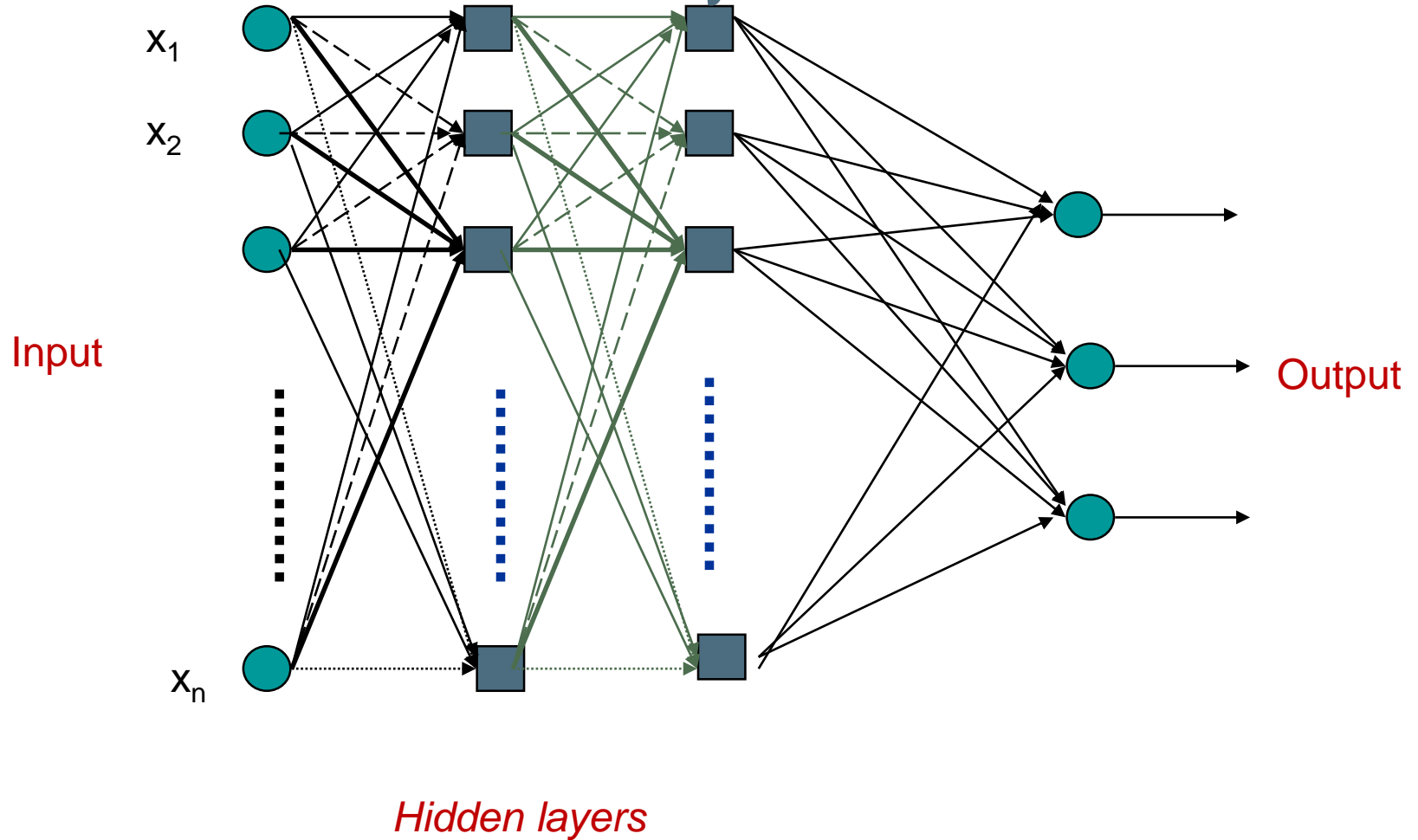


This is a linearly separable problem!

Since for 4 points $\{(-1, 1), (-1, -1), (1, 1), (1, -1)\}$ it is always linearly separable if we want to have three points in a class

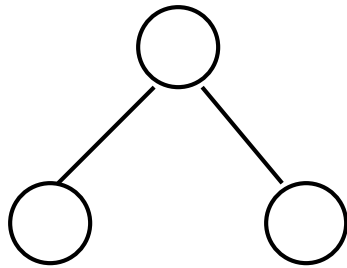
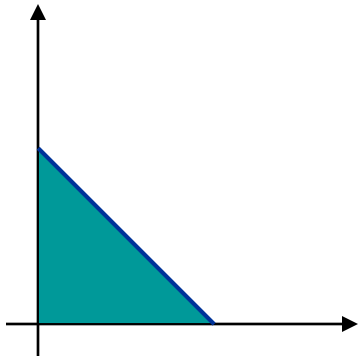
Why MLP

Three-layer networks

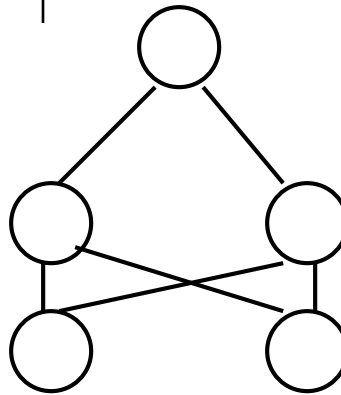
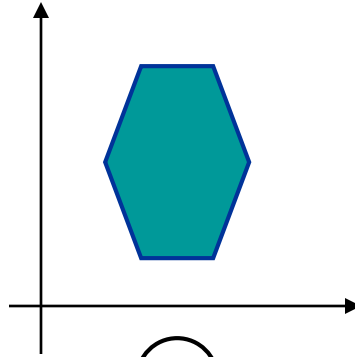


WHY MLP ?

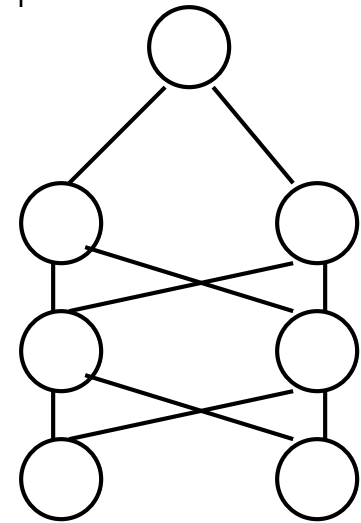
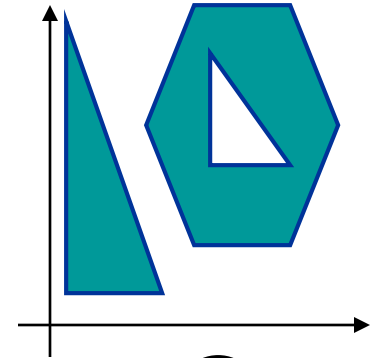
What do each of the layers do?



1st layer draws linear boundaries


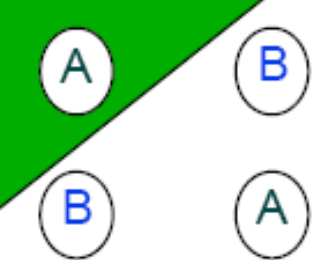
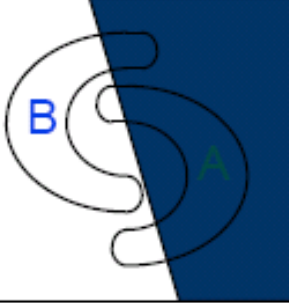

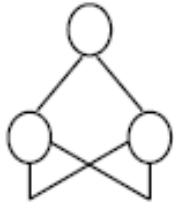
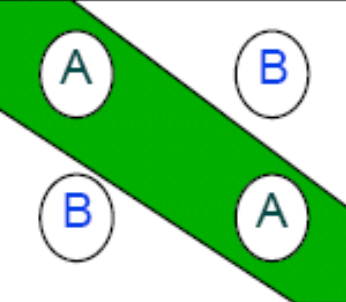
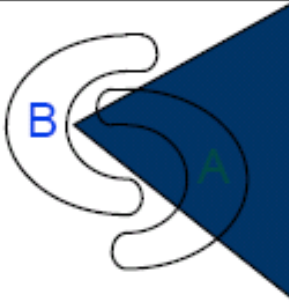

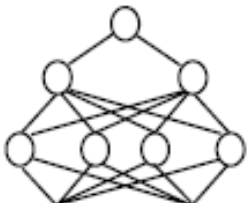
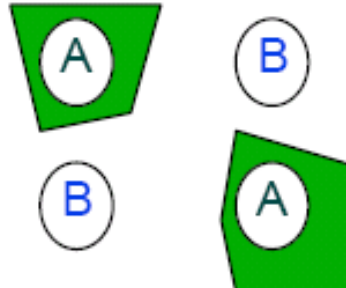

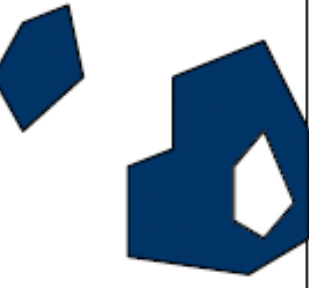


2nd layer combines the boundaries



3rd layer can generate arbitrarily complex boundaries

Different Non-Linearly Separable Problems

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
<p>Single-Layer</p> 	<p>Half Plane Bounded By Hyperplane</p>			
<p>Two-Layer</p> 	<p>Convex Open Or Closed Regions</p>			
<p>Three-Layer</p> 	<p>Arbitrary (Complexity Limited by No. of Nodes)</p>			

Multi Layer perceptron

Multi layer Perceptron in Brief

- Multilayer perceptrons (MLPs) are **feedforward neural networks** trained with the standard **backpropagation** algorithm.
- They are supervised networks so they require a desired response to be trained.
- They learn how to transform input data into a desired response, so they are widely used for pattern classification.
- With one or two hidden layers, they can approximate virtually any input-output map.
- Most neural network applications involve MLPs.

Basic of Multi Layer Perceptron

- In a MLP, there are at least three layers :
 - 1 Input , 1 - ∞ Hidden , 1 Output
- **In short:**
 - The **input layer receives data** about something. Each input neuron presents a piece of data such as pixels to an image of a letter.
 - The **hidden layers do some calculations.**
 - The **output layers** provide an answer in percentage. Each output neuron **represents a different possibility.**

Basic of Multi Layer Perceptron.. cont

- The input layer is receiving the data about something. Eg : For character recognition there would need to be one neuron for each pixel in the image of the character. Obviously there can be only one input layer. Input is usually normalized to a double between 0.0 and 1.0.
- The hidden layer is a layer used for calculations. This is what makes up the "brain" of a MLP. There can be as many hidden layers with any number of neurons. Typically one hidden layer is usually enough, though to get the best performance(learning rate vs. correctness) different configurations could be tested.
- The output layer is the results of what the network has calculated. For character recognition that could only see the letters of the alphabet there would need to be 32 output neurons (one for each letter). A neurons output value is a normalized double between 0.0 and 1.0. 0 is false, 1 is true. It could also be thought of as a percentage of correctness for what that neuron is to represent. In the Optical Character Recognition network, if a neuron representing the capital letter T was 0.85, that would mean the network thinks the input is an 85% chance T, yet also the neuron representing the letter F might be 0.64, meaning the network thinks the input is an 64% chance F.

Multi Layer Perceptron

The Multi-Layer-Perceptron was first introduced by M. Minsky and S. Papert in 1969

Type:

Feedforward

Neuron layers:

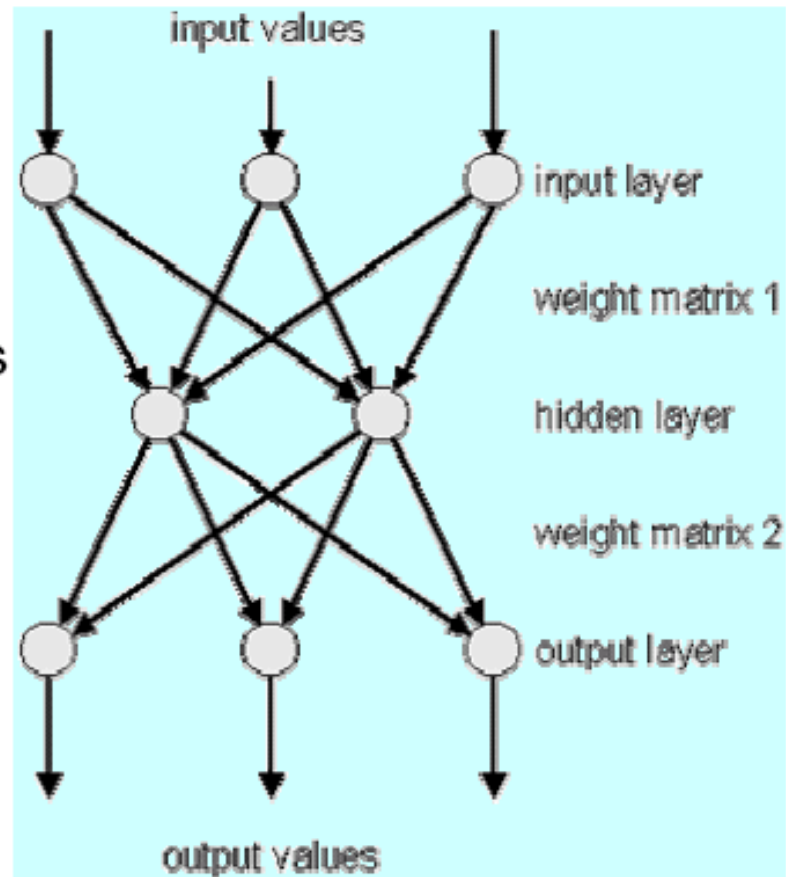
1 input layer

1 or more hidden layers

1 output layer

Learning Method:

Supervised



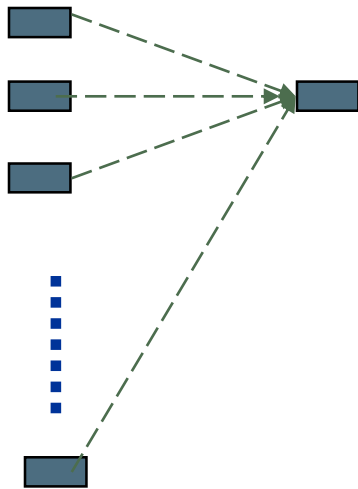
Terminology

- Arrows indicate the direction of data flow.
- The first layer, termed **input layer**, just contains the input vector and does not perform any computations.
- The second layer, termed **hidden layer**, receives input from the input layer and sends its output to the **output layer**.
- After applying their activation function, the neurons in the output layer contain the output vector.

Why MLP

Properties of architecture

- No connections within a layer
- No direct connections between input and output layers
- Fully connected between layers
- Often more than 3 layers
- Number of output units need not equal number of input units
- Number of hidden units per layer can be more or less than input or output units

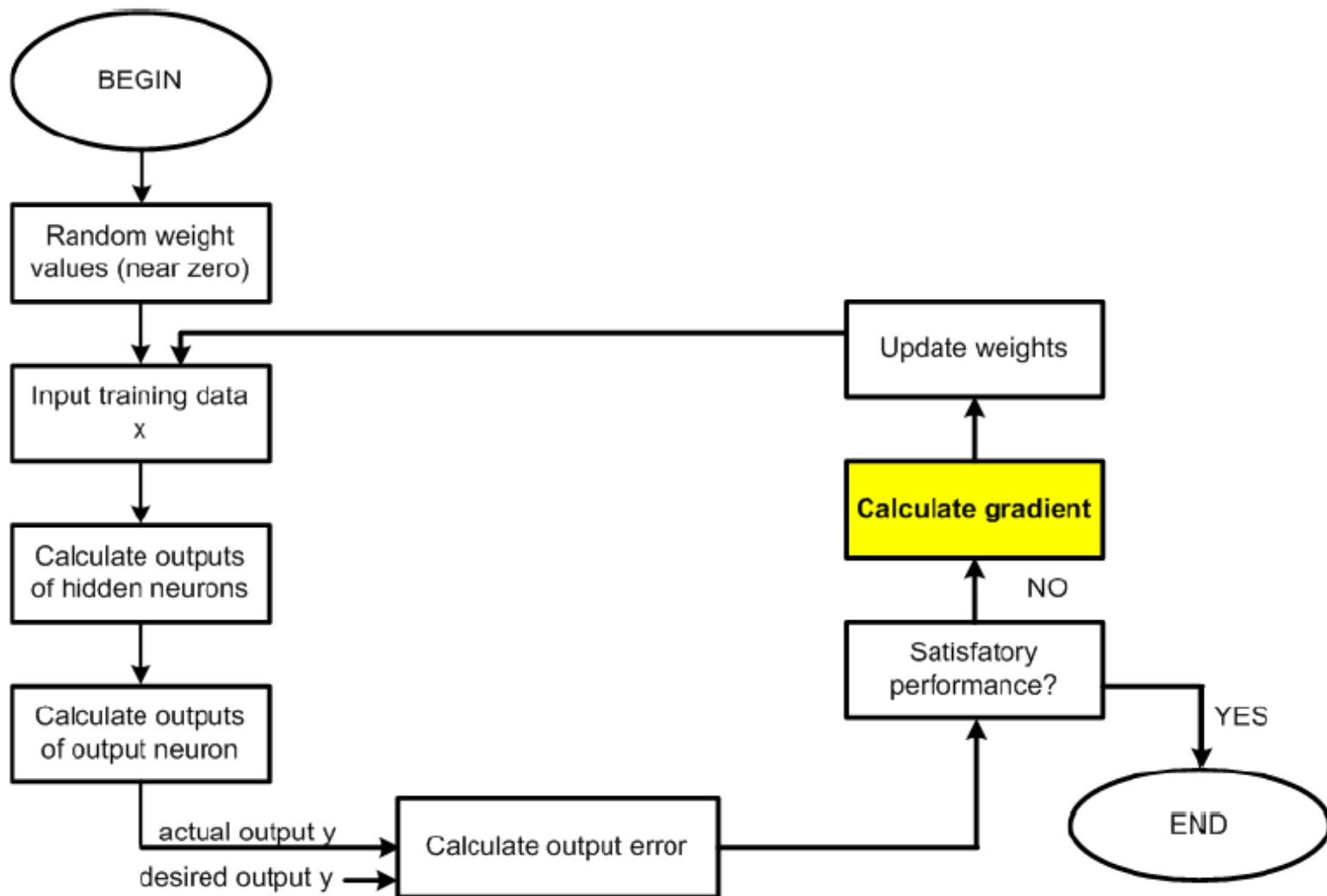


Each unit is a perceptron

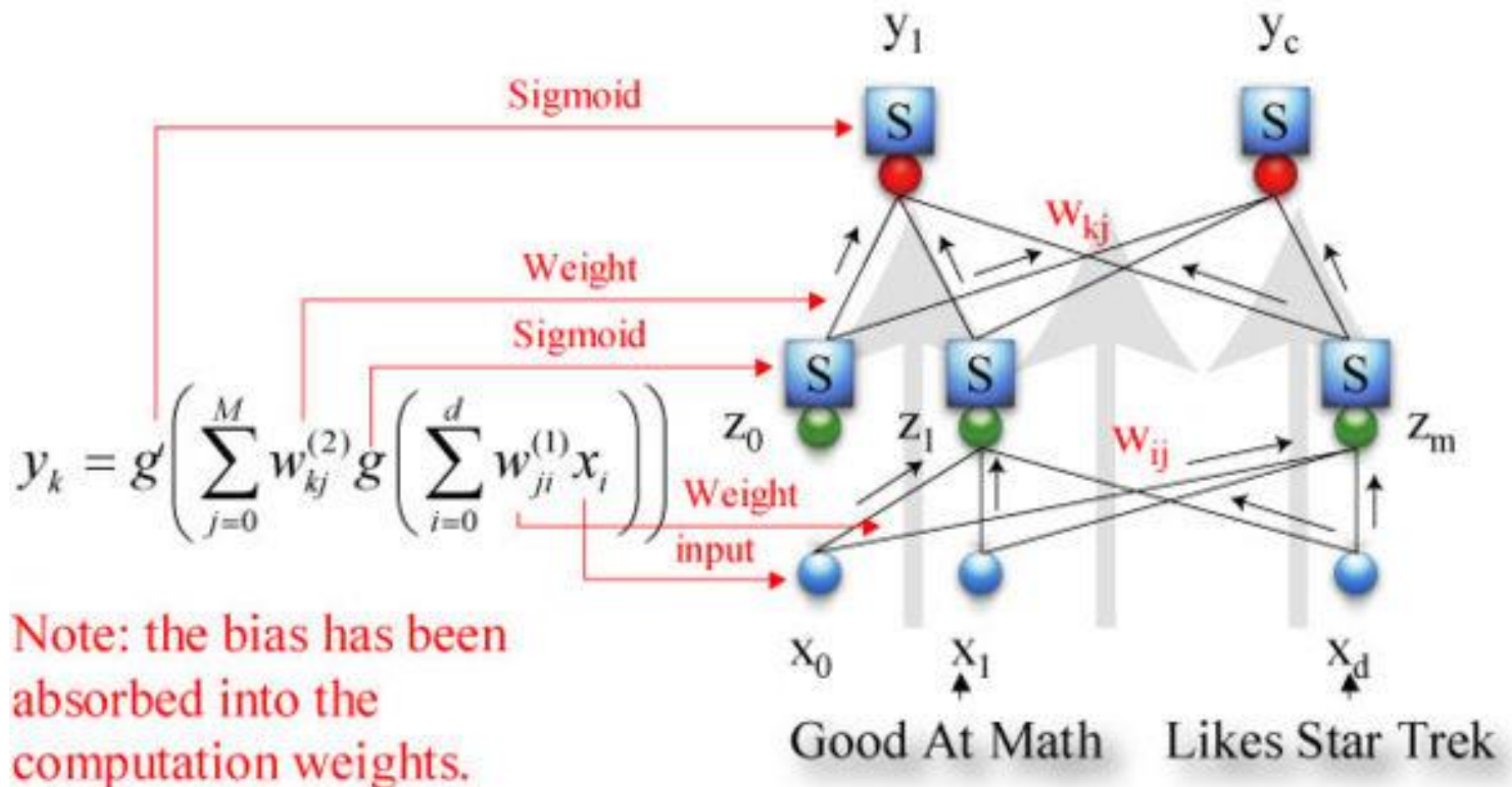
$$y_i = f \left(\sum_{j=1}^m w_{ij} x_j + b_i \right)$$

Often include bias as an extra weight

Multi Layer Perceptron Understanding



How A Multi-Layer Perceptron Calculates An Answer (each neuron)?



Calculation

- The calculations happen inside each individual neuron which each of it needs :
 - Output values of every neuron in the previous layer,
 - Value of every dendrite(weight) connecting the current neuron to every corresponding previous neuron,
 - Activation Function
- The value of every dendrite(weight) in a MLP are randomized double values between 0.0 and 1.0 and need to be altered during learning
- The input layer does not have any calculations performed on it, therefore the input to the input layer neurons is also their output.

Calculation.. cont

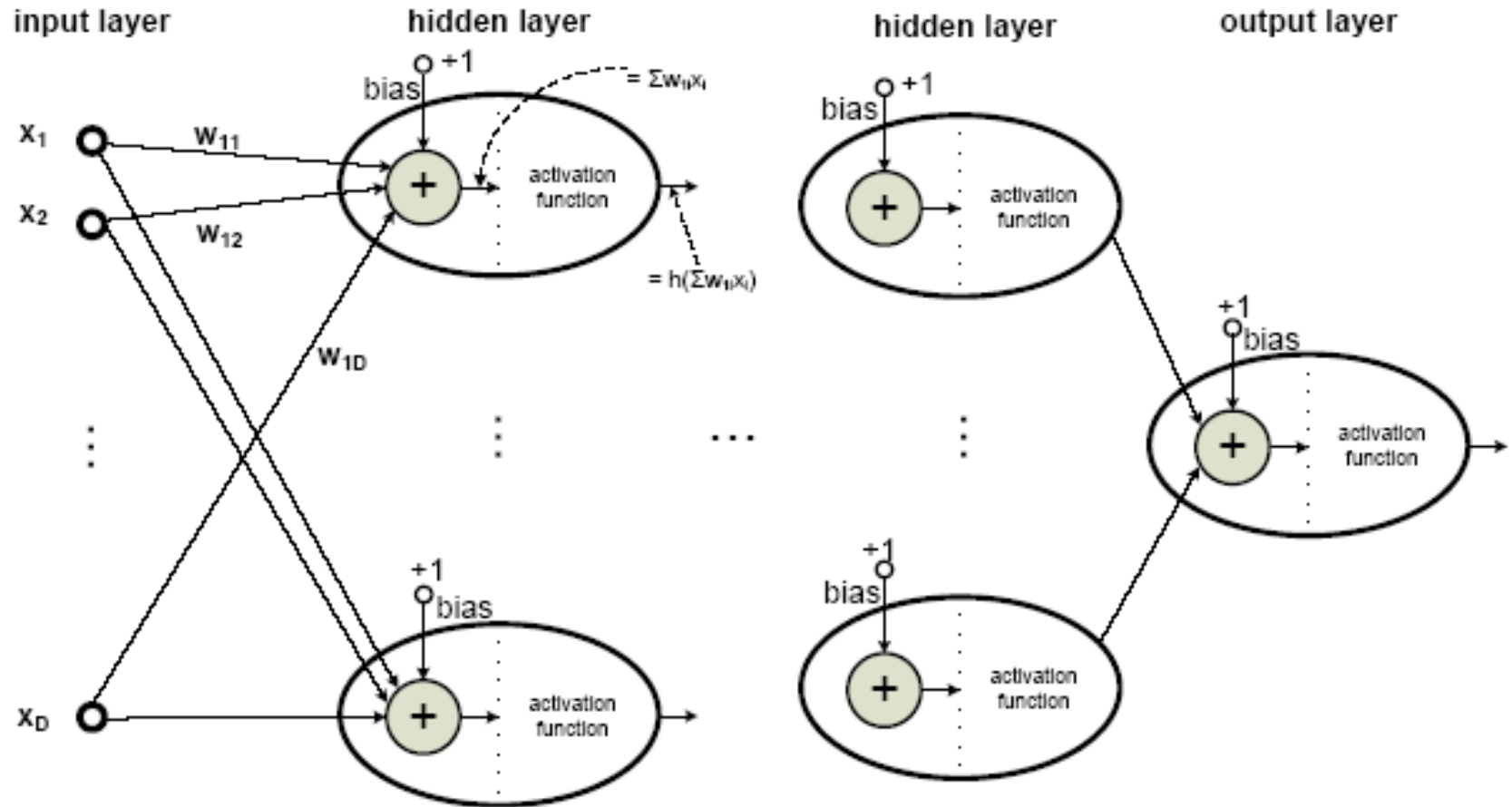
- Starting with the first hidden layer and using the input layers, for each neuron in the layer sum all of the previous layers output values * their corresponding weights.
- Pass the sum through an activation function. Continue to the next layer. Do this for the output layer as well.
- The values of each neuron in the output layer is the answer from the network.

- This can be represented as follows:

$$v = \text{Sigmoid}(\Sigma(p_v * p_w))$$

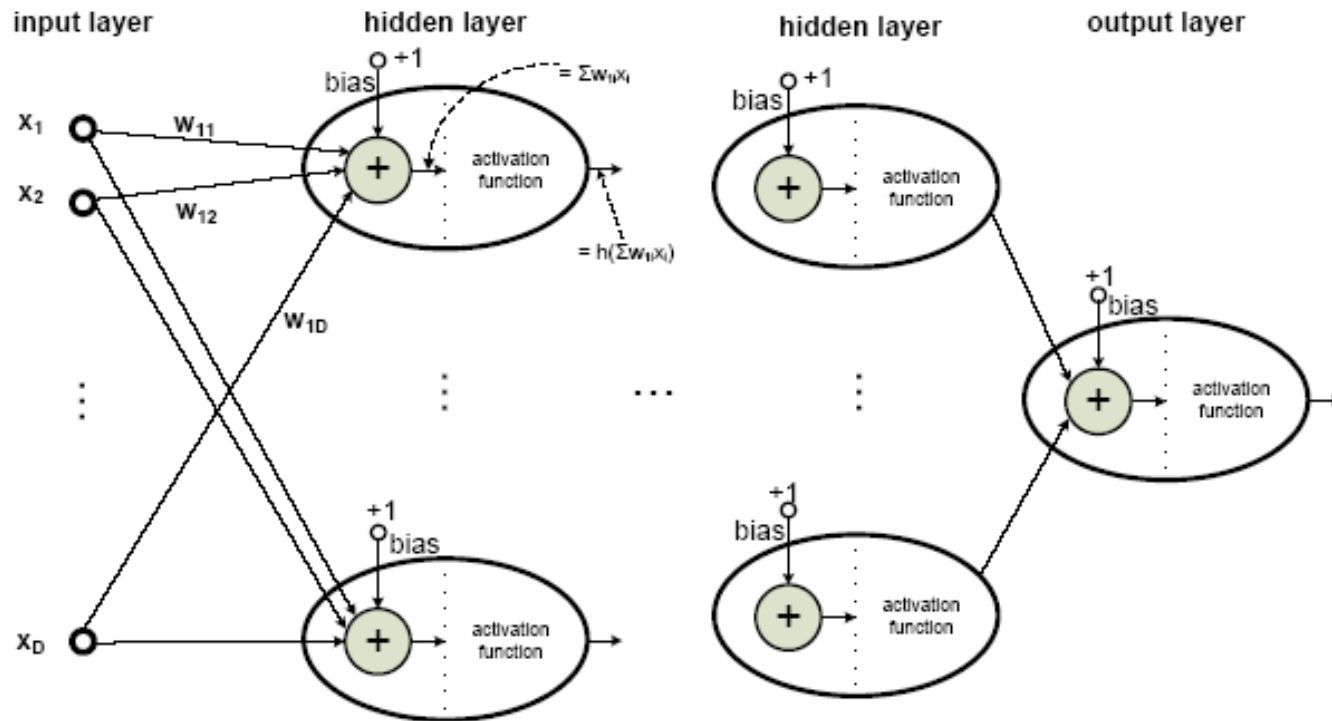
- v = output value of the neuron
- Σ = calculus for [summation]
- p_v = a previous neurons output value
- p_w = the dendrite or weight value connecting to the previous neuron
- Sigmoid = activation function

Why do we need a bias?



Calculation... cont

- There is a problem with this structure though. If all the inputs to the network are 0, when the network is training, it will not be able to learn what all zero inputs are. For this reason there must be a pseudo input added to each layer excluding the output layer. This **bias** acts as a neuron with a value that is always 1 or -1 and it has a dendrite (weight) attached to it and is calculated in with the summation.



How A Multi-Layer Perceptron Learns??

Back-Propagation

Backpropagation

Intro to BP

- **BACKWARDS.**
- Supervised Error Back-propagation Training
 - The mechanism of backward error transmission (delta learning rule) is used to modify the synaptic weights of the internal (hidden) and output layers
- **[The mapping error can be propagated into hidden layers]**
 - Can implement arbitrary complex/output mappings or decision surfaces to separate pattern classes

Architecture: Backpropagation Network

The Backpropagation Net was first introduced by G.E. Hinton, E. Rumelhart and R.J. Williams in 1986

Type:

Feedforward

Neuron layers:

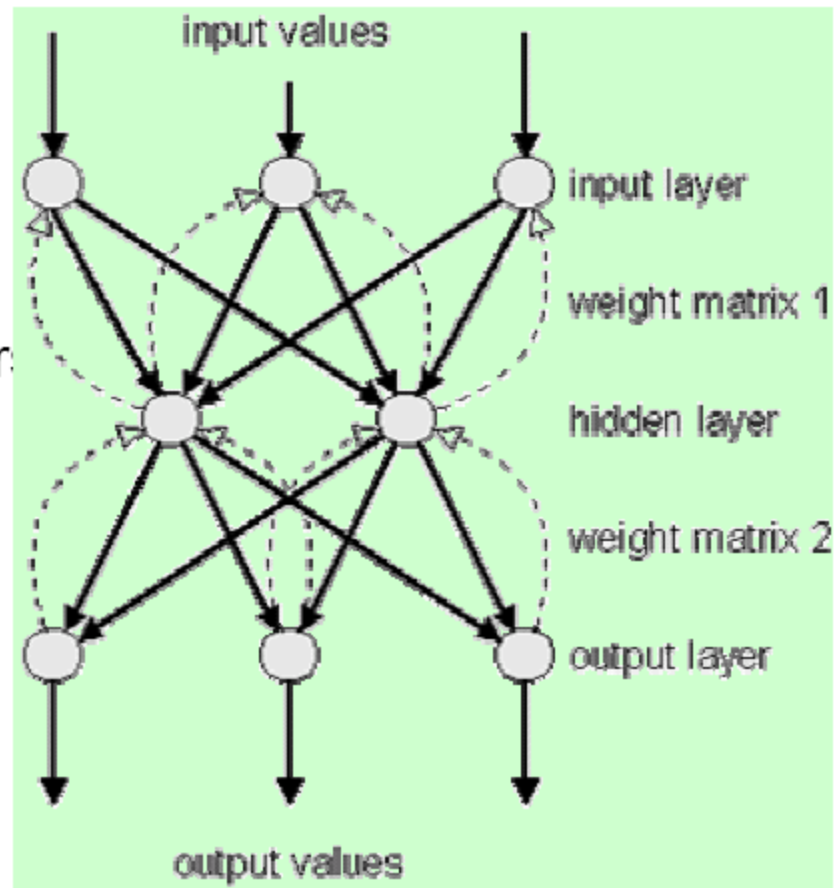
1 input layer

1 or more hidden layer

1 output layer

Learning Method:

Supervised



Backpropagation Preparation

■ Training Set

- A collection of input-output patterns that are used to train the network

■ Testing Set

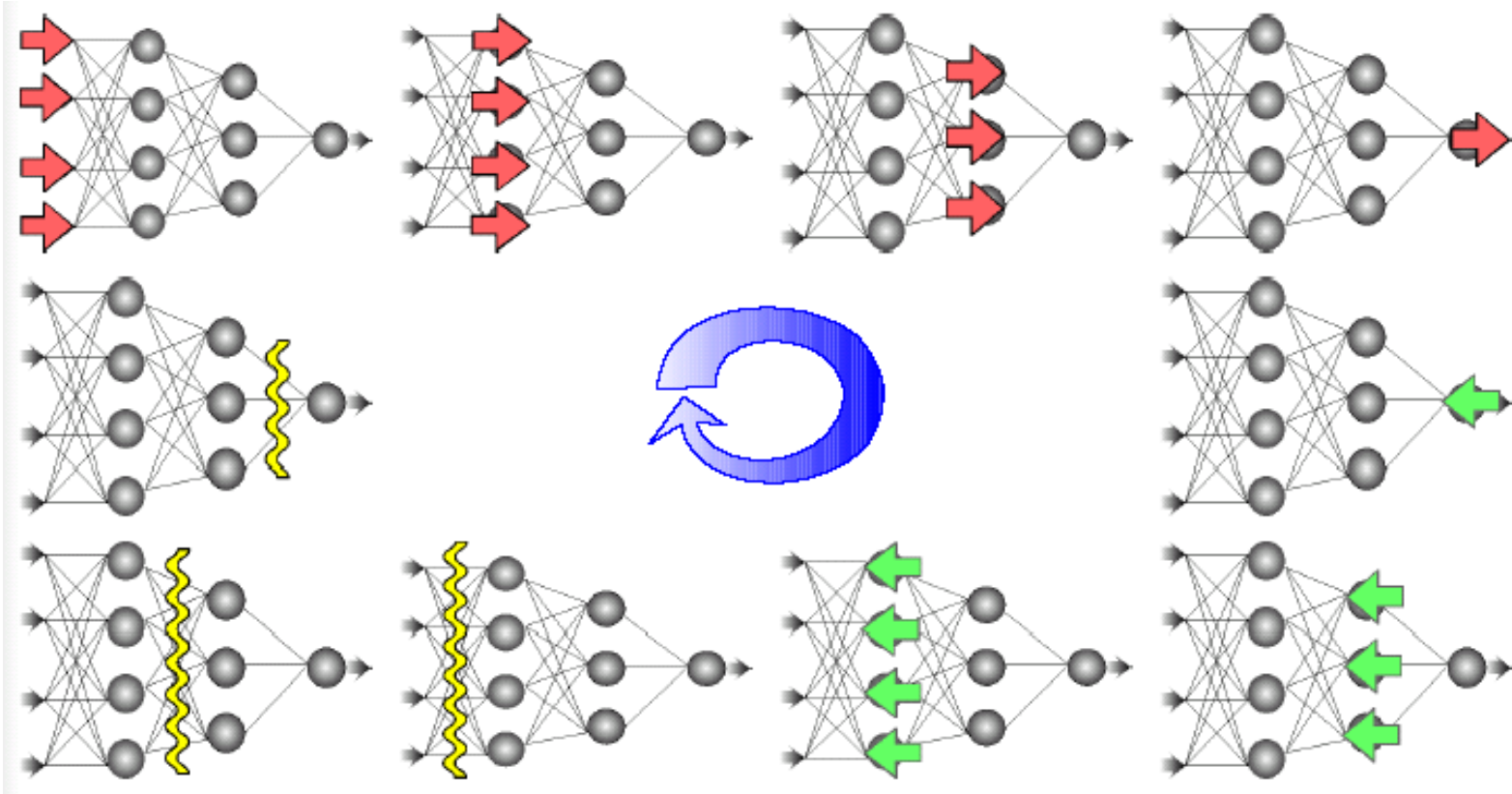
- A collection of input-output patterns that are used to assess network performance

■ Learning Rate- α

- A scalar parameter, analogous to step size in numerical integration, used to set the rate of adjustments

Backpropagation Training Cycle

1- Feedforward of the input training pattern



3 - Adjustment of the weights 2 - Backpropagation of the associated error

Backpropagation Learning Algo

- Initialise weights and threshold.
 - Set all weights and thresholds to small random values.
- Present input and desired output
 - Present input $\mathbf{X}_p = x_0, x_1, x_2, \dots, x_{n-1}$ and target output $\mathbf{T}_p = t_0, t_1, \dots, t_{m-1}$ where n is the number of input nodes and m is the number of output nodes. Set w_0 to be $-\phi$, the bias, and x_0 to be always 1. For pattern association, \mathbf{X}_p and \mathbf{T}_p represent the patterns to be associated. For classification, \mathbf{T}_p is set to zero except for one element set to 1 that corresponds to the class that \mathbf{X}_p is in.
- Calculate the actual output
 - *Each layer calculates the following:*
 - $fy_{pj} = f[w_0x_0 + w_1x_1 + \dots + w_nx_n]$
 - This is then passes this to the next layer as an input. The final layer outputs values o_{pj} .
- Adapts weights
 - Starting from the output we now work backwards.
 - $w_{ij}(t+1) = w_{ij}(t) + \tilde{n}b_{pj}o_{pj}$, where \tilde{n} is a gain term and b_{pj} is an error term for pattern p on node j .
 - For output units : $b_{pj} = ko_{pj}(1 - o_{pj})(t - o_{pj})$
 - For hidden units : $b_{pj} = ko_{pj}(1 - o_{pj})[(b_{p0}w_{j0} + b_{p1}w_{j1} + \dots + b_{pk}w_{jk})]$
 - where the sum(in the [brackets]) is over the k nodes in the layer above node j .

■ Adapts weights

- Starting from the output we now work backwards.
- $w_{ij}(t+1) = w_{ij}(t) + \tilde{n}b_{pj}o_{pj}$, where \tilde{n} is a gain term and b_{pj} is an error term for pattern p on node j .
- For output units : $b_{pj} = ko_{pj}(1 - o_{pj})(t - o_{pj})$
- For hidden units : $b_{pj} = ko_{pj}(1 - o_{pj})[(b_{p0}w_{j0} + b_{p1}w_{j1} + \dots + b_{pk}w_{jk})]$
- where the sum(in the [brackets]) is over the k nodes in the layer above node j .

Algorithm (after Lippmann, 1987) The back-propagation algorithm has five steps.

- (a) *Initialise weights.* Set all weights to small random values.
- (b) *Present inputs and desired outputs (training pairs).* Present an input vector \mathbf{u} and specify the desired outputs \mathbf{d} . If the net is used as a classifier then all desired outputs are typically set to zero, except one (set to 1). The input could be new on each trial, or samples from a training set could be presented cyclically until weights stabilise.
- (c) *Calculate actual outputs.* Calculate the outputs by successive use of $\mathbf{y} = \mathbf{f}(\mathbf{w}^T \mathbf{u})$ where \mathbf{f} is a vector of activation functions.
- (d) *Adapt weights.* Start at the output neurons and work back to the first hidden layer. Adjust weights by

$$w_{ji} = w_{ji} + \eta \delta_j y_i \quad (55)$$

In this equation w_{ji} is the weight from hidden neuron i (or an input node) to neuron j , y_i is the output of neuron i (or an input), η is the learning rate parameter, and δ_j is the gradient; if neuron j is an output neuron then it is defined by (53), and if neuron j is hidden then it is defined by (54). Convergence is sometimes faster if a momentum term is added according to (78).

- (e) *Go to step 2.*

Learning Process

- If the system output is different with desired output , the error rate should be calculated and the weight will be altered.
- The training process happens on a per-input basis. If the network is trained on one set of data, such as the letter "A", it will not be trained if it comes across the letter "B". All the neural network will know is how much "B" is like "A".
- Training may have to be done hundreds or even thousands of times on each kind of input until the error rate of the network is within a satisfactory range. Usually less than 1%.
- The process as follows:
 - ❑ Compute resulting output
 - ❑ Compute ERROR for neurons in output layer
 - ❑ Compute ERROR for all other neurons
 - ❑ Compute CHANGE for all weights

Error Calculation

1. Error for the output layer.

- Error is equal to the desired output minus the actual output.

$$\beta = d - o$$

Where:

- β = Error
- d = Desired output
- o = Actual output

2. Error for all other neurons.

- Take the weights and outputs and errors of each neuron in the right-side layer.
- Sum the product of them all. Also include $1 - \text{output value}$ for the slope of the line.

$$\beta_j = \sum (w_k * o_k (1 - o_k) \beta_k)$$

Where:

- β_j = Error for current neuron
- w_k = Weight for neuron in right-side layer
- o_k = Output for neuron in right-side layer
- β_k = Error for neuron in right-side layer
- j = Current Neuron
- k = Neuron in right-side layer
- Σ = calculus for[summation]

3. Weight Changes Calculation

- For a weight of a neuron. Take the neuron in the left-side layer the weight is connected to Multiply it with the slope and the error of the current neuron Multiply in a learning rate (How fast the network will learn) .20-.25 seem to be optimal
 - $\Delta w_j = r * o_i * o_j (1 - o_j) \beta_j$
 - $w_j = w_j + \Delta w_j$
 - Where:
 - Δw_j = Change in weight for current neuron
 - w_j = Weight of the current neuron
 - r = Learning rate of the network (how fast the network learns) .20-.25 are good.
 - o_i = Output for neuron in the left-side layer
 - o_j = Output for neuron in the right-side layer
 - β_j = Error for current neuron
 - j = Current Neuron
 - i = Neuron in left-side layer

4. Putting It All Together

$$w_j = w_j + \Delta w_j$$

$$\Delta w_j = r * o_i * o_j (1 - o_j) \beta_j$$

$$\beta_j = \sum (w_k * o_k (1 - o_k) \beta_k)$$

$$\beta_z = d - o$$

BackPropagation in Detail

BP Training Algorithm

Initialize weights.

(Set to small random values).

While stopping condition is false, do Steps 2–9.

Step 2. For each training pair, do Steps 3–8.

Feedforward:

Step 3. Each input unit ($X_i, i = 1, \dots, n$) receives input signal x_i and broadcasts this signal to all units in the layer above (the hidden units).

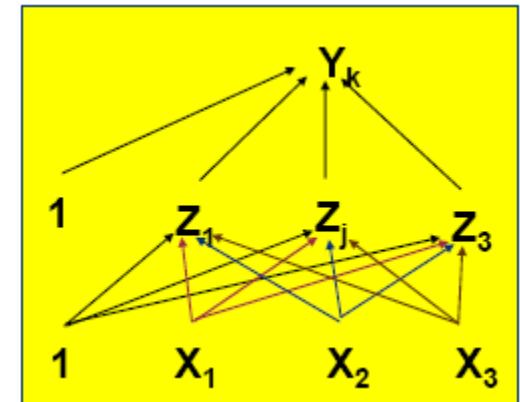
Step 4. Each hidden unit ($Z_j, j = 1, \dots, p$) sums its weighted input signals,

$$z_in_j = v_{0j} + \sum_{i=1}^n x_i v_{ij},$$

applies its activation function to compute its output signal,

$$z_j = f(z_in_j),$$

and sends this signal to all units in the layer above (output units).



Initialize weights.

(Set to small

While stopping c

Step 2. For e

Feedf

Step 4

Step 4

Step 5. Each output unit ($Y_k, k = 1, \dots, m$) sums its weighted input signals,

$$y_in_k = w_{0k} + \sum_{j=1}^p z_j w_{jk}$$

and applies its activation function to compute its output signal,

$$y_k = f(y_in_k).$$

Backpropagation of error:

Step 6. Each output unit ($Y_k, k = 1, \dots, m$) receives a target pattern corresponding to the input training pattern, computes its error information term,

$$\delta_k = (t_k - y_k) f'(y_in_k),$$

for sigmoid $f(1-f)$

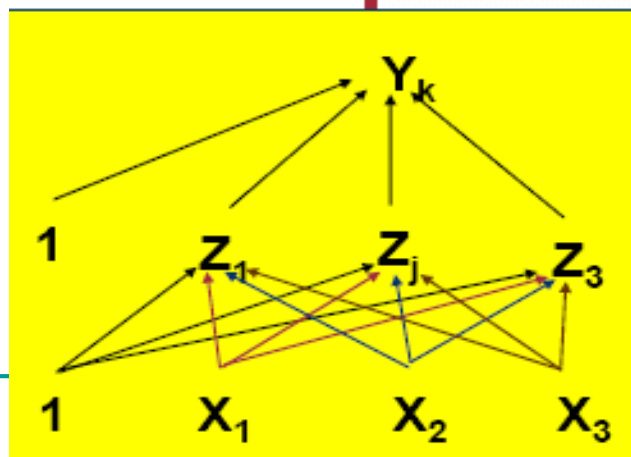
calculates its weight correction term (used to update w_{jk} later),

$$\Delta w_{jk} = \alpha \delta_k z_j.$$

calculates its bias correction term (used to update w_{0k} later),

$$\Delta w_{0k} = \alpha \delta_k,$$

and sends δ_k to units in the layer below.



Step 7. Each hidden unit ($Z_j, j = 1, \dots, p$) sums its delta inputs (from units in the layer above),

$$\delta_in_j = \sum_{k=1}^m \delta_k w_{jk},$$

multiplies by the derivative of its activation function to calculate its error information term,

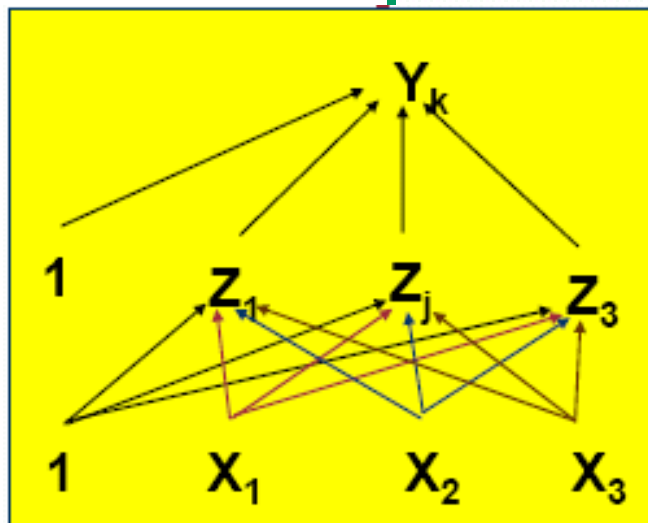
$$\delta_j = \delta_in_j f'(z_in_j),$$

calculates its weight correction term (used to update v_{ij} later).

$$\Delta v_{ij} = \alpha \delta_j x_i,$$

and calculates its bias correction term (used to update v_{0j} later),

$$\Delta v_{0j} = \alpha \delta_j.$$



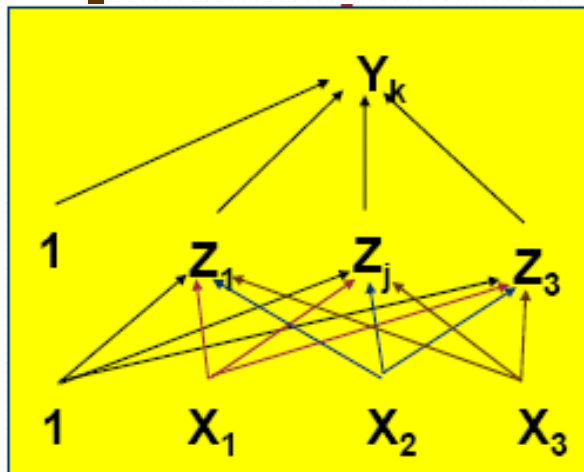
Update weights and biases:

Step 8. Each output unit ($Y_k, k = 1, \dots, m$) updates its bias and weights ($j = 0, \dots, p$):

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}.$$

Each hidden unit ($Z_j, j = 1, \dots, p$) updates its bias and weights ($i = 0, \dots, n$):

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}.$$



Step 9. Test stopping condition.

Example

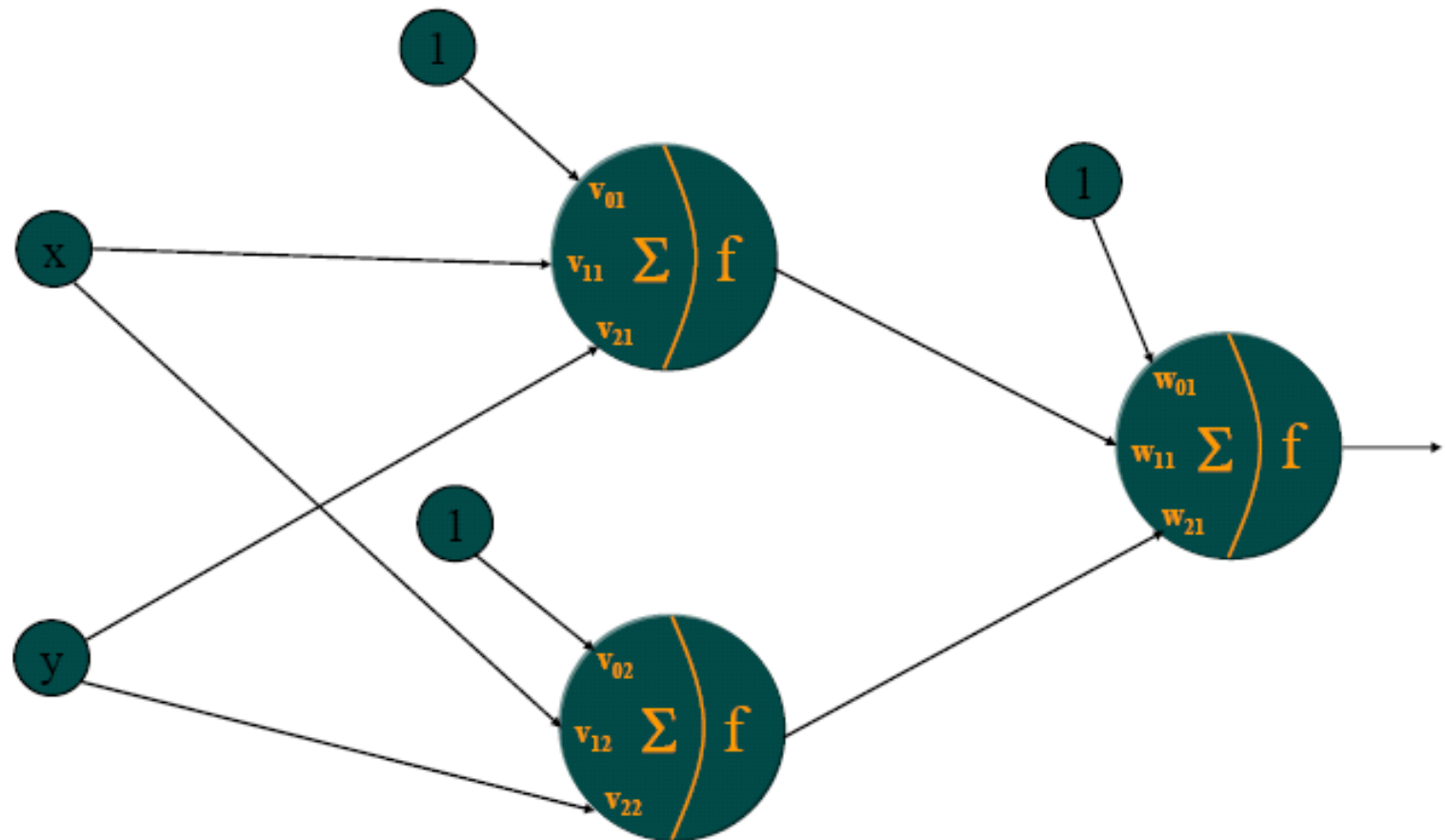
Example 1

- [The XOR function could not be solved by a single layer perceptron network]

- The function is:

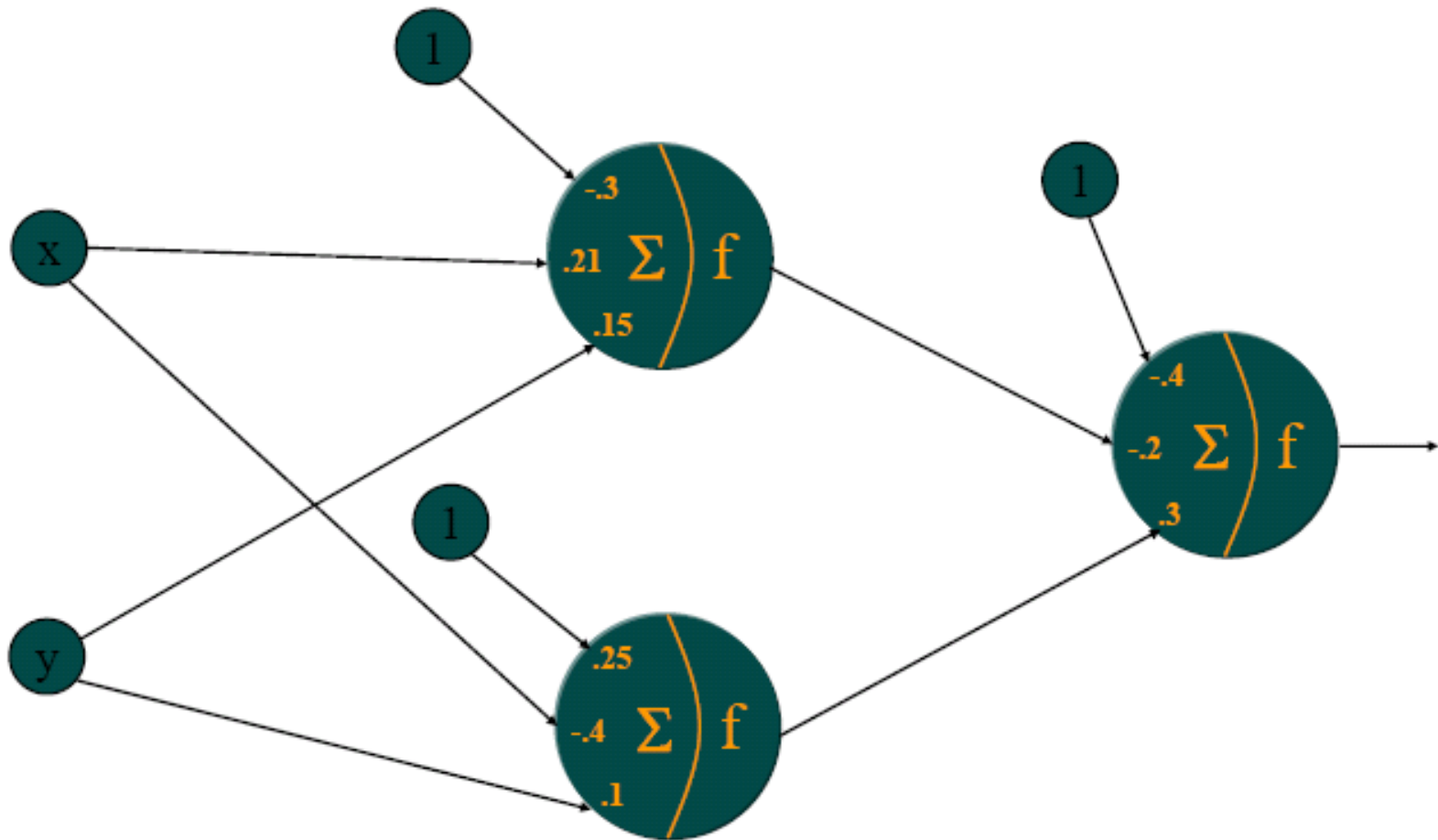
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

XOR Architecture

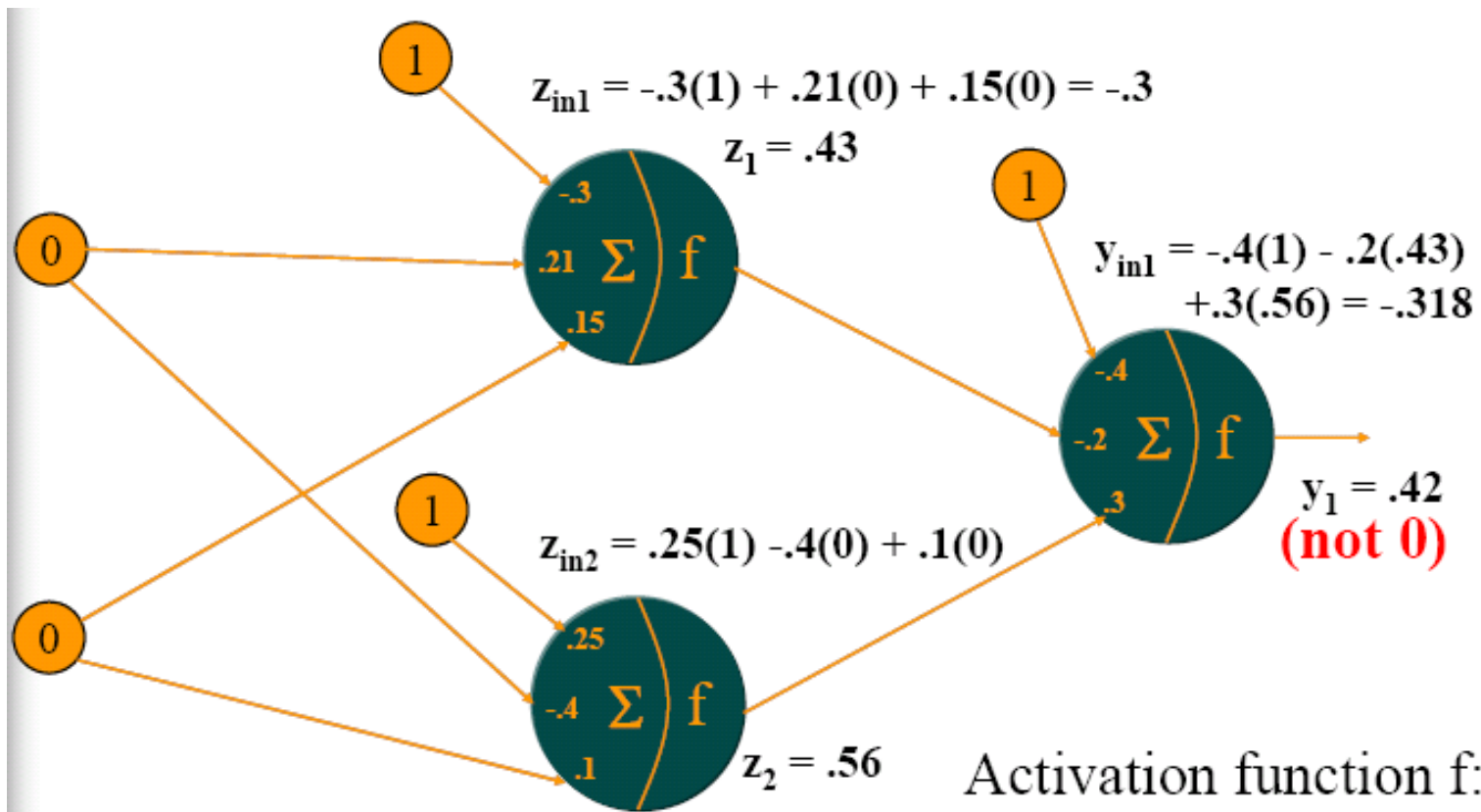


Initial Weights

Randomly assign small weight values:



Feedforward – 1st Pass

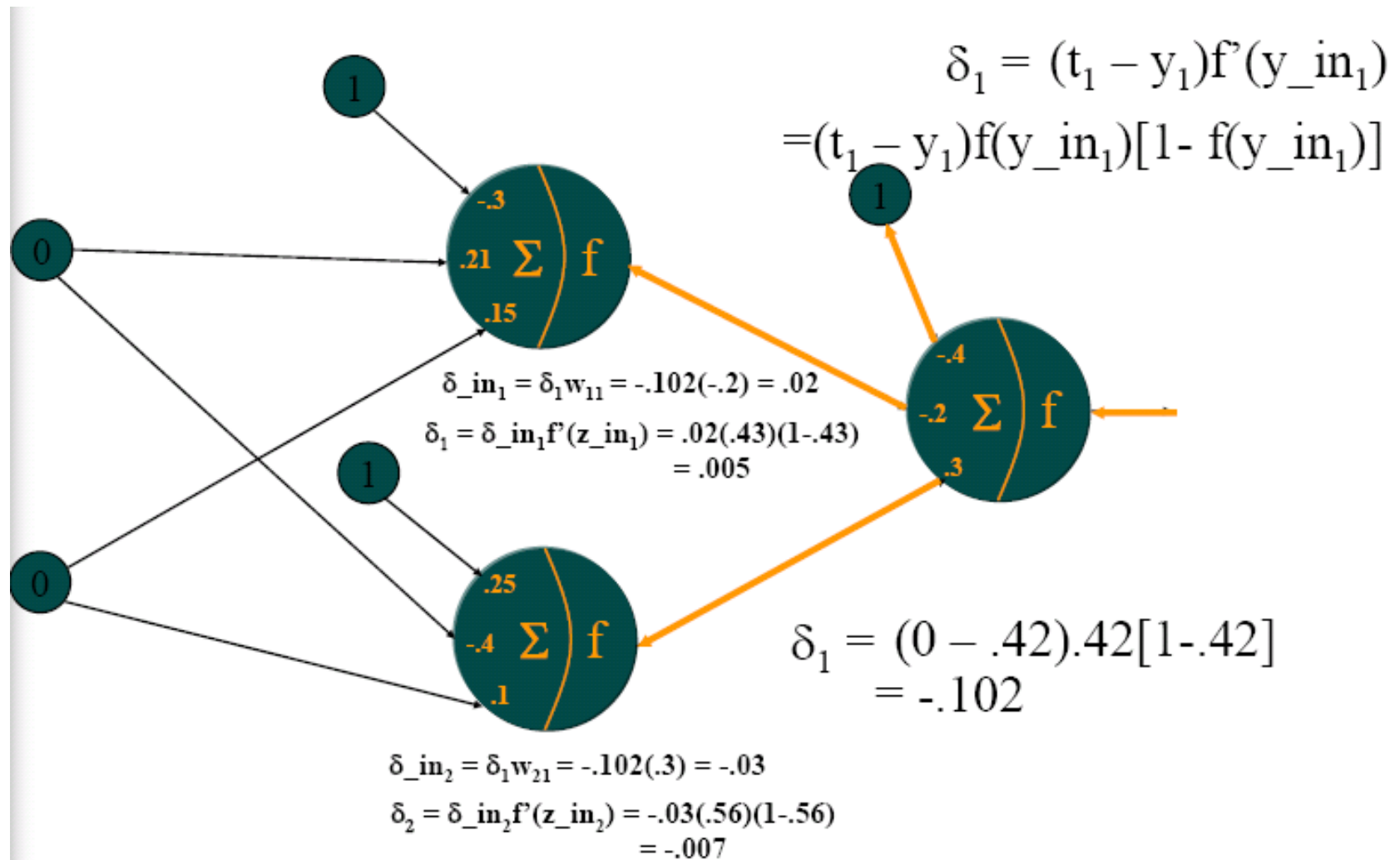


Training Case: (0 0 0)

Activation function f :

$$f(x) = \frac{1}{1 + e^{-x}}$$

Backpropagate



Deep Learning Neural Networks

- Deep-learning networks are distinguished from the more commonplace single-hidden-layer neural networks by their depth; that is, the number of node layers through which data passes in a multistep process of pattern recognition.
- Traditional machine learning relies on shallow nets, composed of one input and one output layer, and at most one hidden layer in between. More than three layers (including input and output) qualifies as “deep” learning. So deep is a strictly defined, technical term that means more than one hidden layer.
- In deep-learning networks, each layer of nodes trains on a distinct set of features based on the previous layer’s output.
- The further we advance into the neural net, the more complex the features our nodes can recognize, since they aggregate and recombine features from the previous layer.