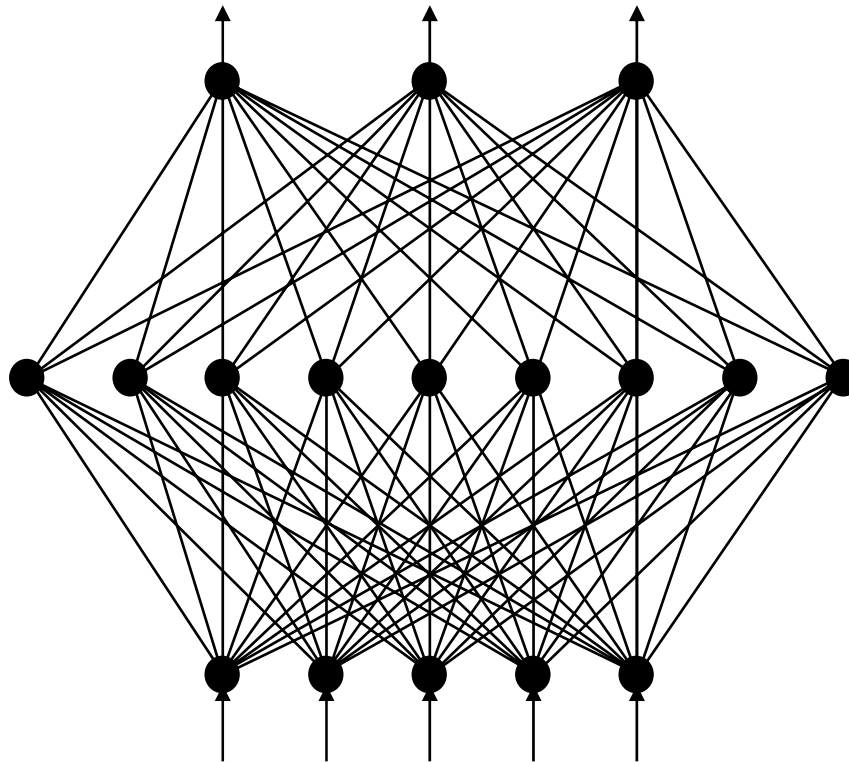
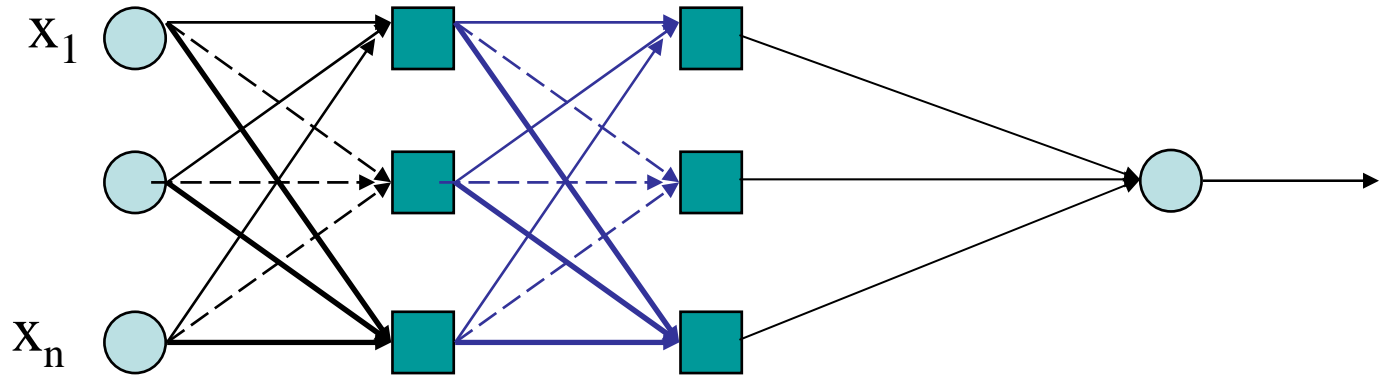


Backpropagation Learning Algorithm





The backpropagation algorithm was used to train the multi layer perceptron MLP

MLP used to describe any general Feedforward (no recurrent connections) Neural Network FNN

However, we will concentrate on nets with units arranged in layers

Architecture of BP Nets

- **Multi-layer, feed-forward networks have the following characteristics:**
 - They must have at least one hidden layer
 - Hidden units must be non-linear units (usually with sigmoid activation functions)
 - Fully connected between units in two consecutive layers, but no connection between units within one layer.
 - For a net with only one hidden layer, each hidden unit receives input from all input units and sends output to all output units
 - 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

Other Feedforward Networks

- **Madaline**
 - Multiple adalines (of a sort) as hidden nodes
- **Adaptive multi-layer networks**
 - Dynamically change the network size (# of hidden nodes)
- **Networks of radial basis function (RBF)**
 - e.g., Gaussian function
 - Perform better than sigmoid function (e.g., interpolation in function approximation)

Introduction to Backpropagation

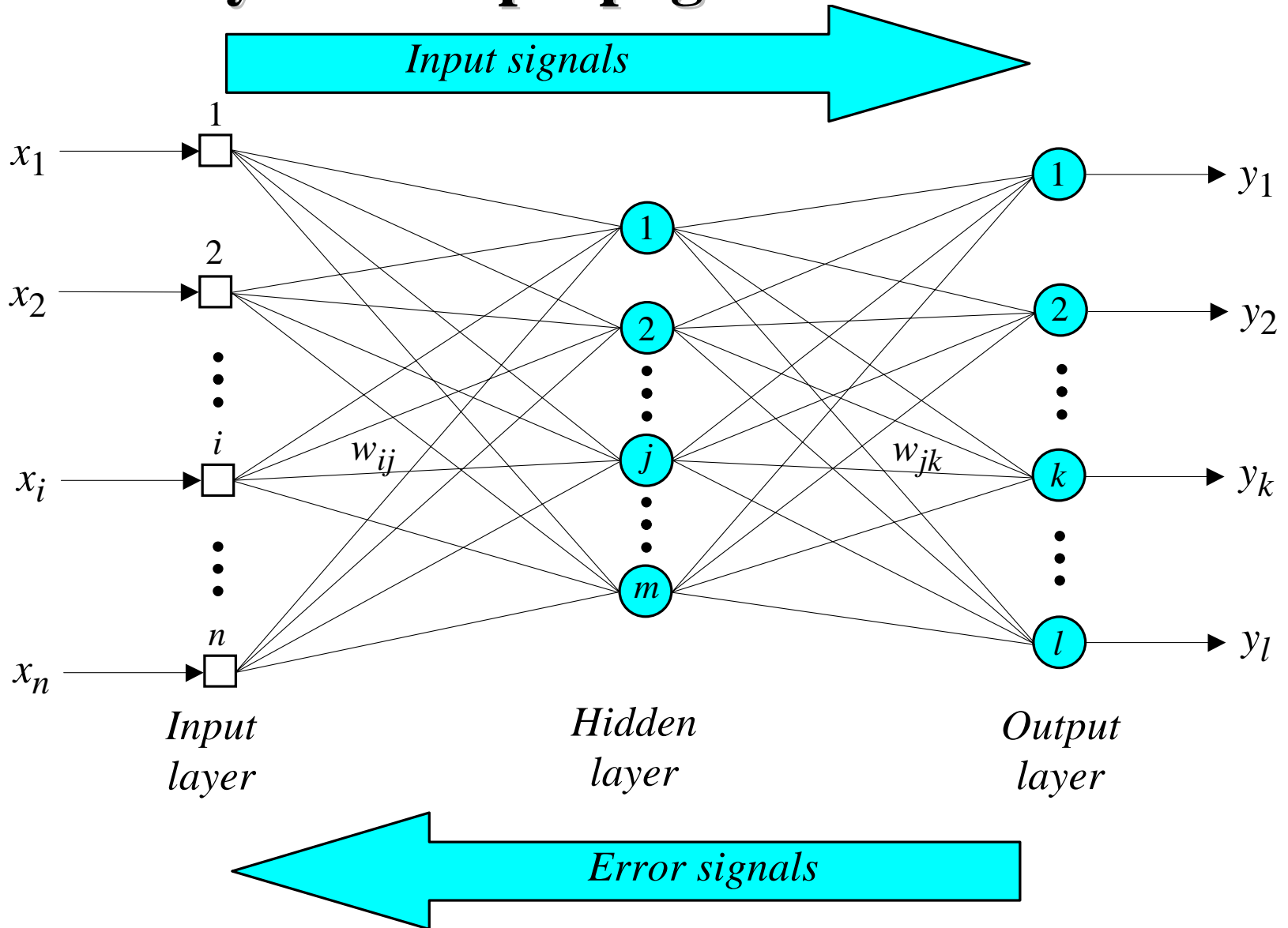
- In 1969 a method for learning in multi-layer network, Backpropagation (or generalized delta rule) , was invented by Bryson and Ho.
- **It is** best-known example of a training algorithm. Uses training data to adjust weights and thresholds of neurons so as to minimize the networks errors of prediction.
- Slower than gradient descent .
- Easiest algorithm to understand
- Backpropagation works by applying the ***gradient descent*** rule to a feedforward network.

- **How many hidden layers and hidden units per layer?**
 - Theoretically, one hidden layer (possibly with many hidden units) is sufficient for any L2 functions
 - There is no theoretical results on minimum necessary # of hidden units (either problem dependent or independent)
 - **Practical rule :**
 - $n = \#$ of input units; $p = \#$ of hidden units
 - For binary/bipolar data: $p = 2n$
 - For real data: $p \gg 2n$
 - Multiple hidden layers with fewer units may be trained faster for similar quality in some applications

Training a BackPropagation Net

- **Feedforward training of input patterns**
 - each input node receives a signal, which is broadcast to all of the hidden units
 - each hidden unit computes its activation which is broadcast to all output nodes
- **Back propagation of errors**
 - each output node compares its activation with the desired output
 - based on these differences, the error is propagated back to all previous nodes *Delta Rule*
- **Adjustment of weights**
 - weights of all links computed simultaneously based on the errors that were propagated back

Three-layer back-propagation neural network



Generalized delta rule

- Delta rule only works for the output layer.
- Backpropagation, or the generalized delta rule, is a way of creating desired values for hidden layers

Description of Training BP Net:

Feedforward Stage

1. Initialize weights with small, random values
2. While stopping condition is not true
 - for each training pair (input/output):
 - each input unit broadcasts its value to all hidden units
 - each hidden unit sums its input signals & applies activation function to compute its output signal
 - each hidden unit sends its signal to the output units
 - each output unit sums its input signals & applies its activation function to compute its output signal

Training BP Net:

Backpropagation stage

3. Each output computes its error term, its own weight correction term and its bias(threshold) correction term & sends it to layer below
4. Each hidden unit sums its delta inputs from above & multiplies by the derivative of its activation function; it also computes its own weight correction term and its bias correction term

Training a Back Prop Net: Adjusting the Weights

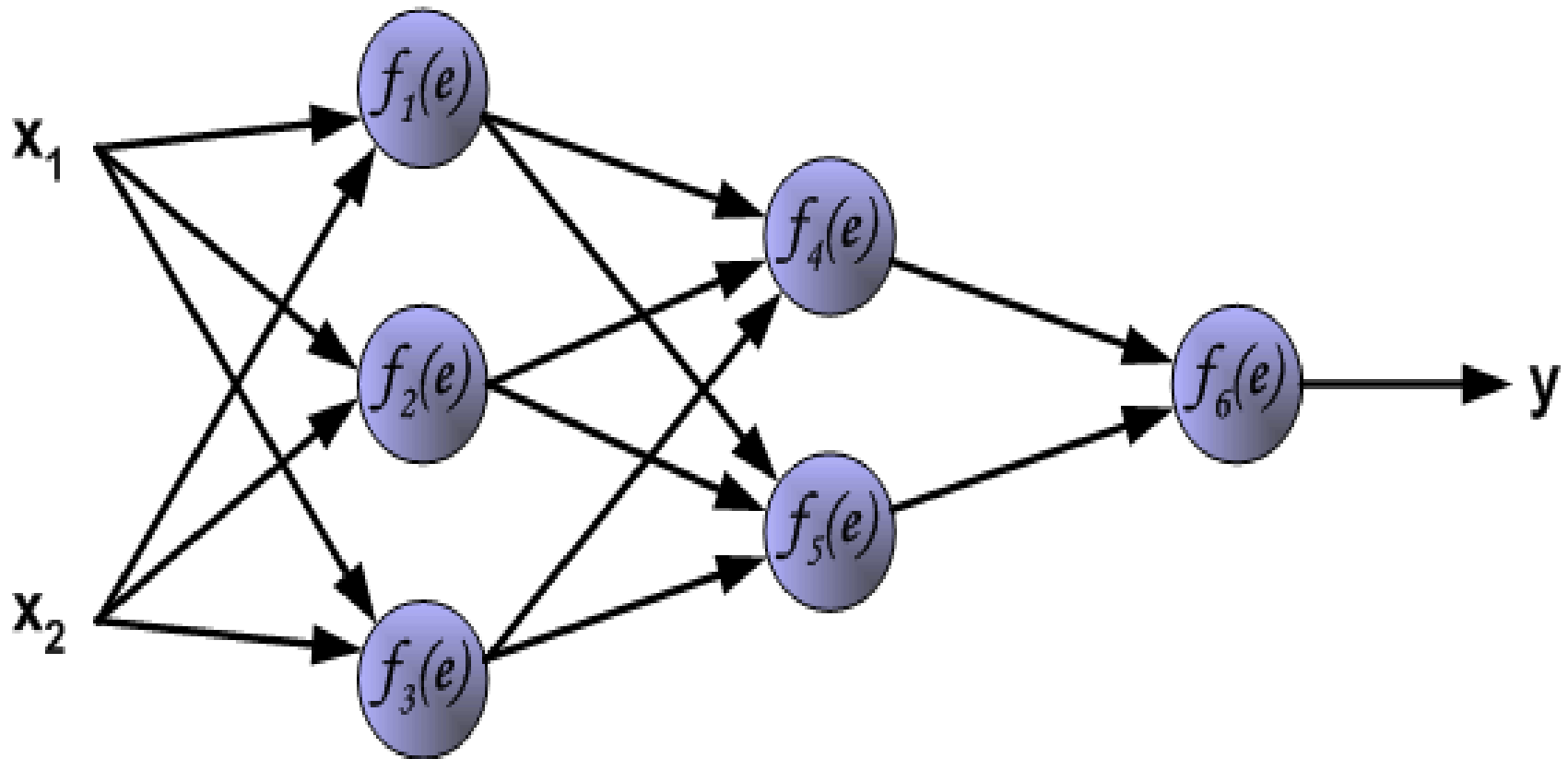
5. Each output unit updates its weights and bias
6. Each hidden unit updates its weights and bias
 - Each training cycle is called an epoch. The weights are updated in each cycle
 - It is not analytically possible to determine where the global minimum is. Eventually the algorithm stops in a low point, which may just be a local minimum.

How long should you train?

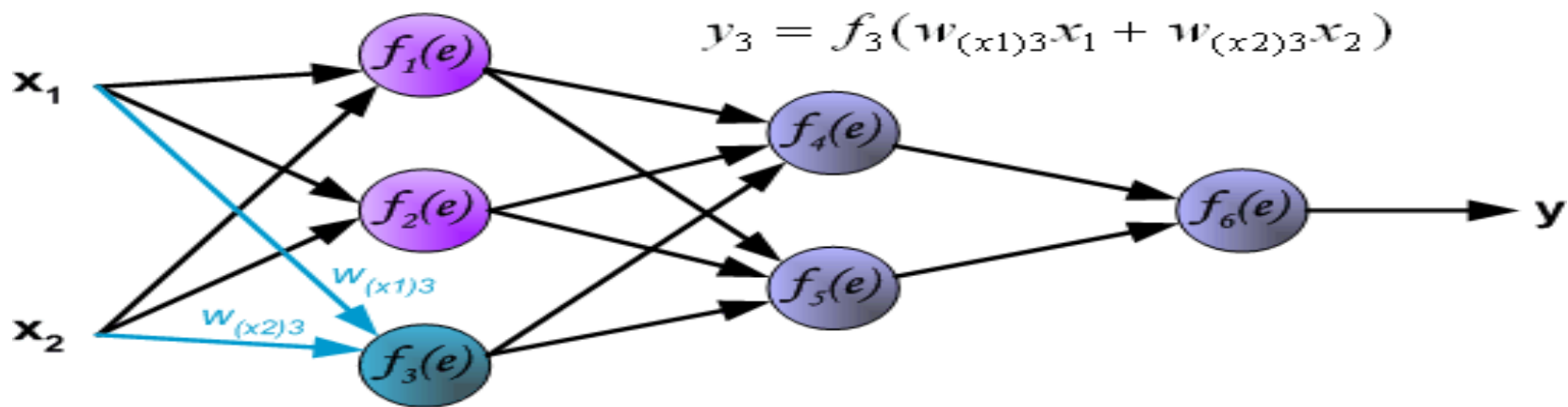
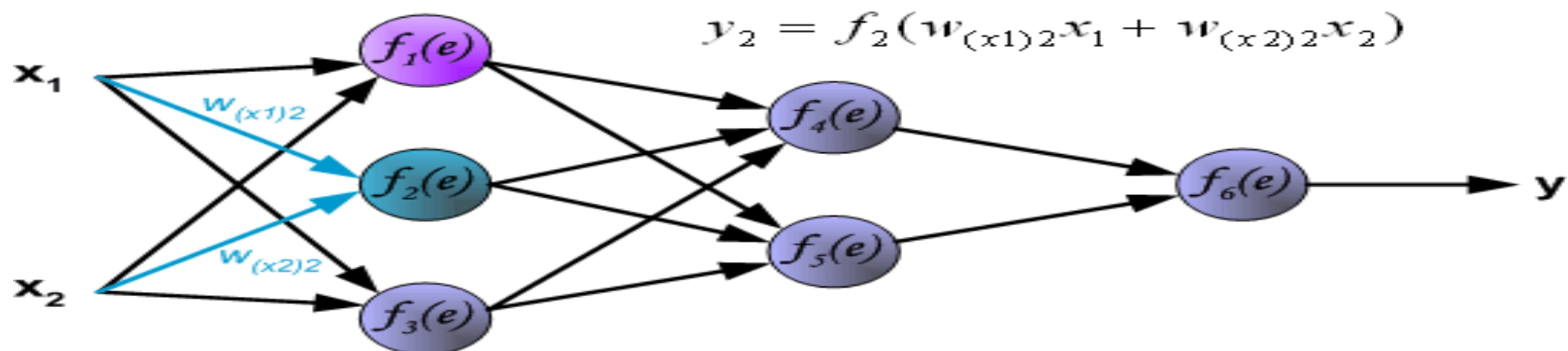
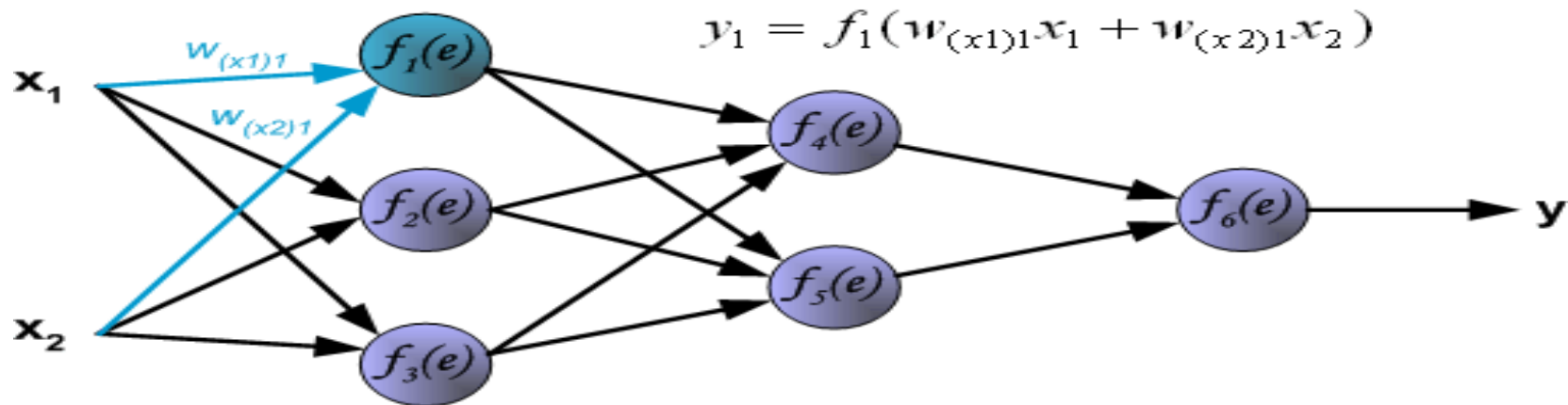
- Goal: balance between correct responses for training patterns & correct responses for new patterns (memorization v. generalization)
- In general, network is trained until it reaches an acceptable error rate (e.g. 95%)
- If train too long, you run the risk of overfitting

Graphical description of training multi-layer neural network using *BP* algorithm

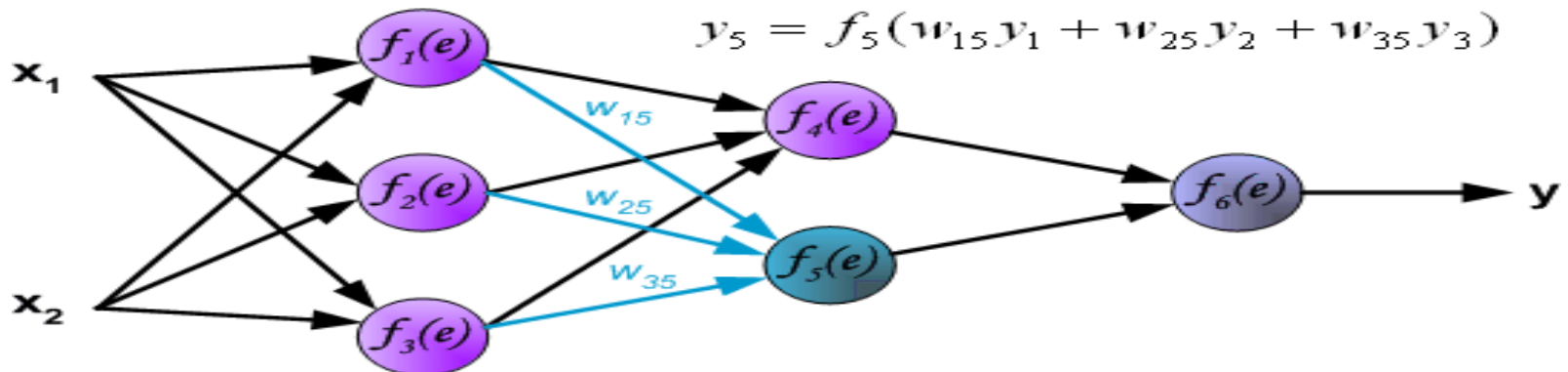
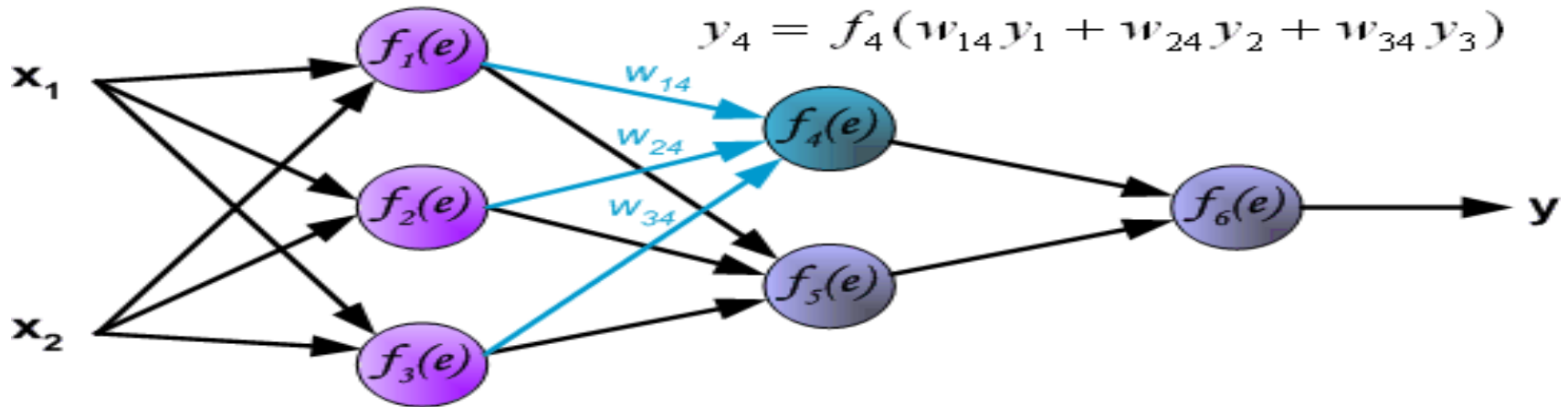
To apply the BP algorithm to the following FNN



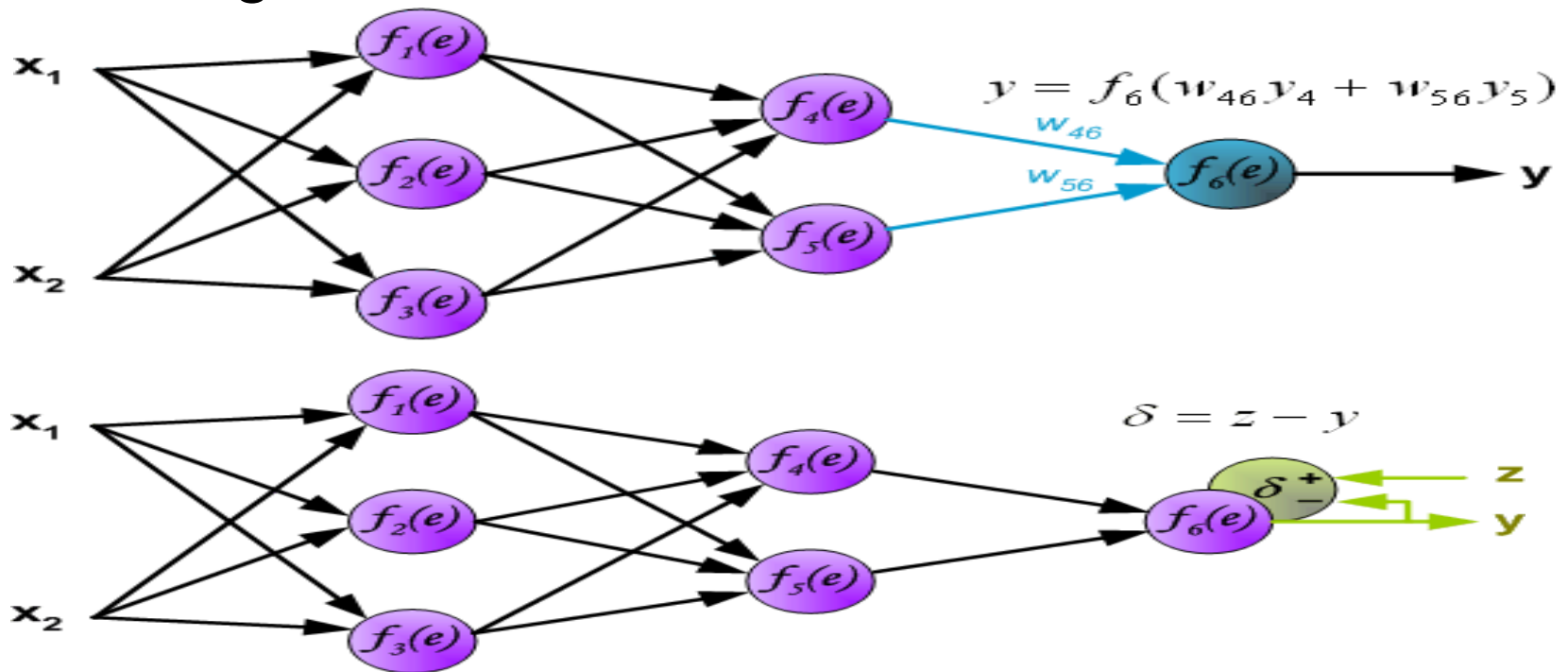
- To teach the neural network we need training data set. The training data set consists of **input signals** (x_1 and x_2) assigned with corresponding target (**desired output**) z .
- The network training is an iterative process. In each iteration weights coefficients of nodes are modified using new data from training data set.
- After this stage we can determine output signals values for each neuron in each network layer.
- Pictures below illustrate how signal is propagating through the network, Symbols $w(x_m)n$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n .



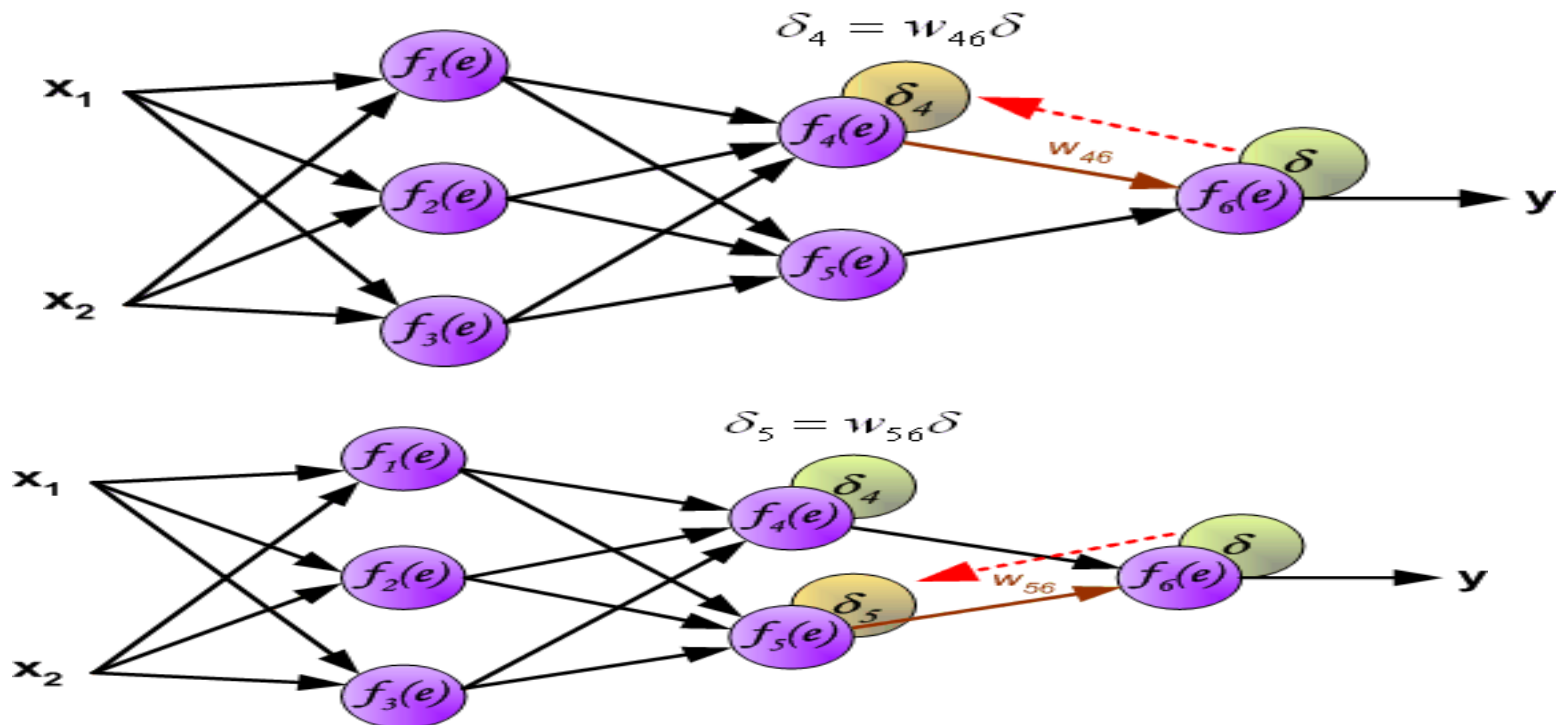
- Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron m and input of neuron n in the next layer.



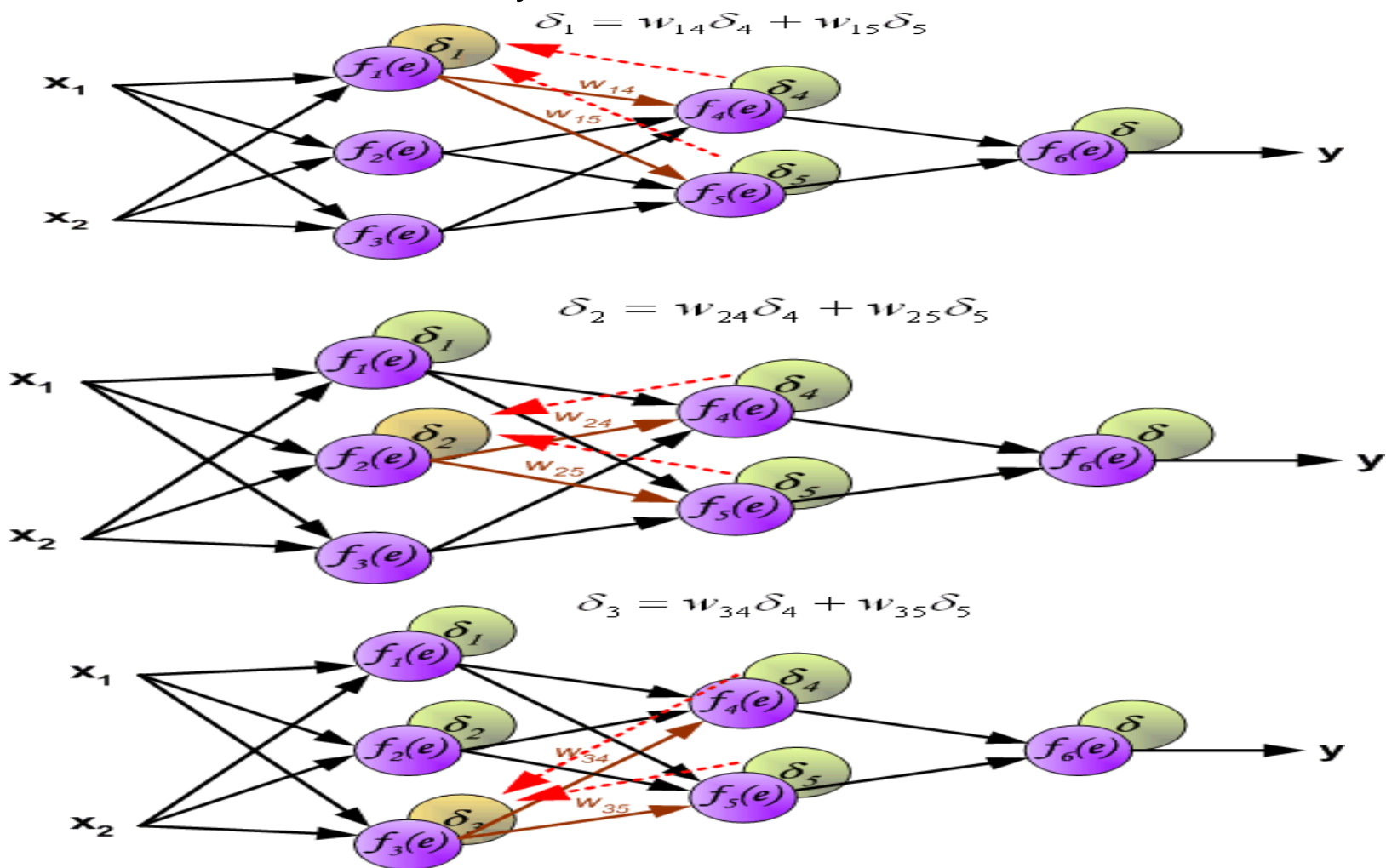
- Propagation of signals through the output layer.
- In the next algorithm step the output signal of the network y is compared with the desired output value (the target), which is found in training data set. The difference is called error



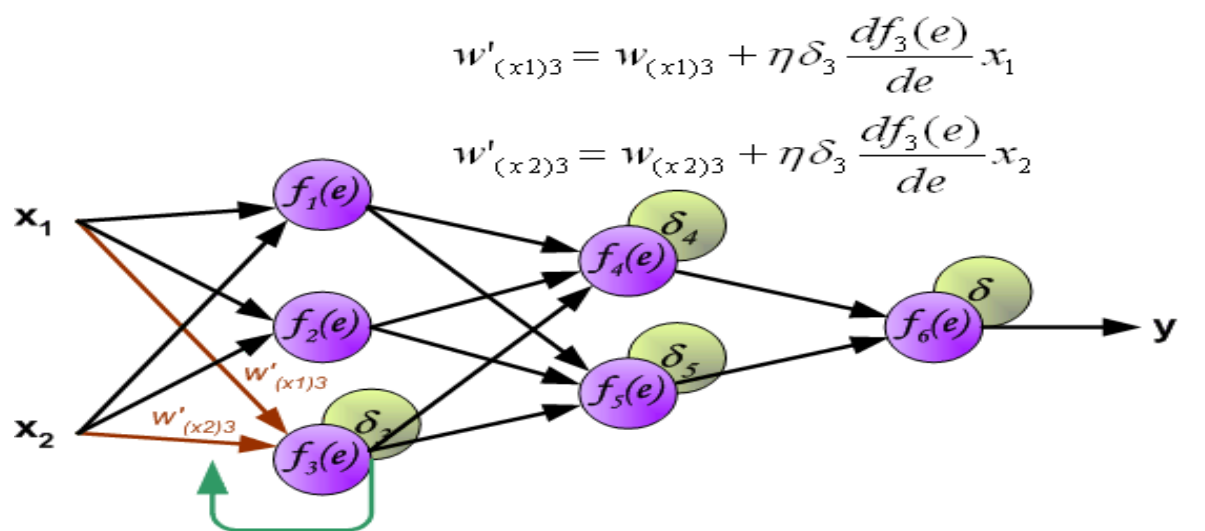
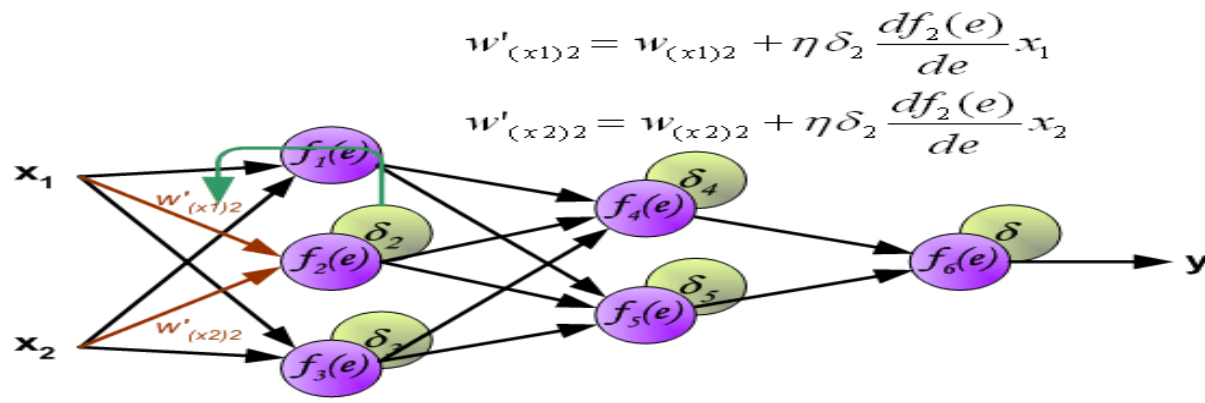
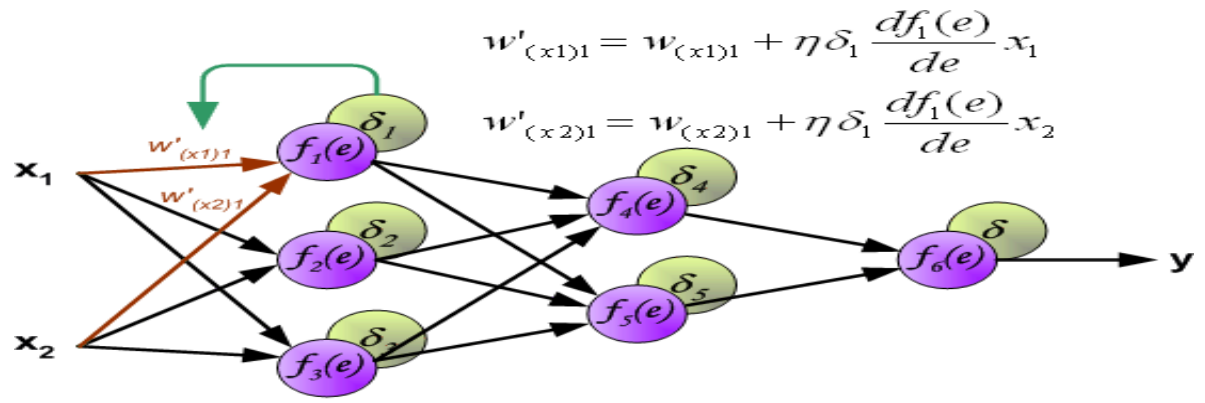
- It is impossible to compute error signal for internal neurons directly, because output values of these neurons are unknown. For many years the effective method for training multiplayer networks has been unknown.
- Only in the middle eighties the backpropagation algorithm has been worked out. The idea is to propagate error signal δ (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.



- The weights' coefficients w_{mn} used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons they are added. The illustration is below:



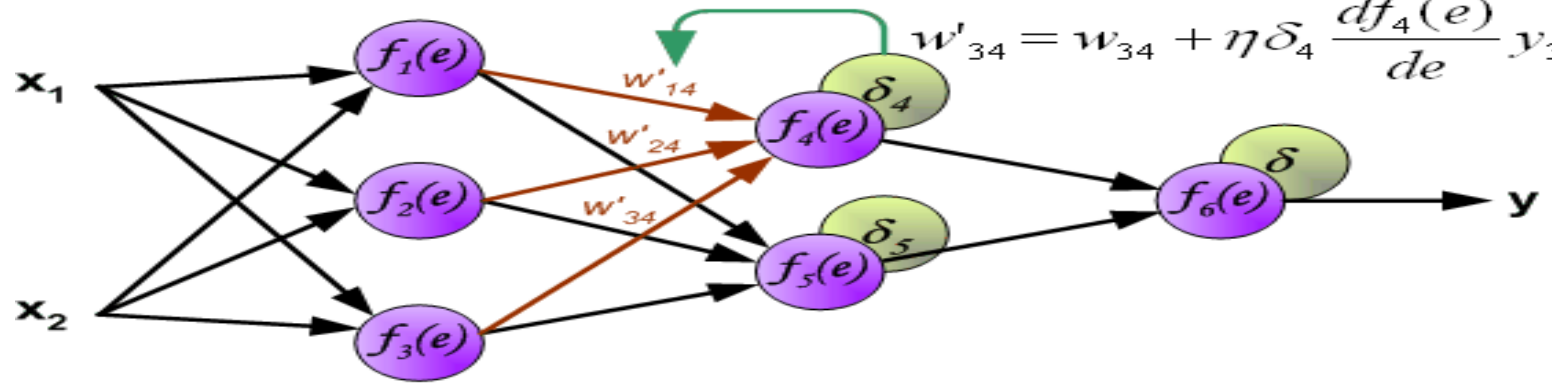
- When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below $df(e)/de$ represents derivative of neuron activation function (which weights are modified)



$$w'_{14} = w_{14} + \eta \delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta \delta_4 \frac{df_4(e)}{de} y_2$$

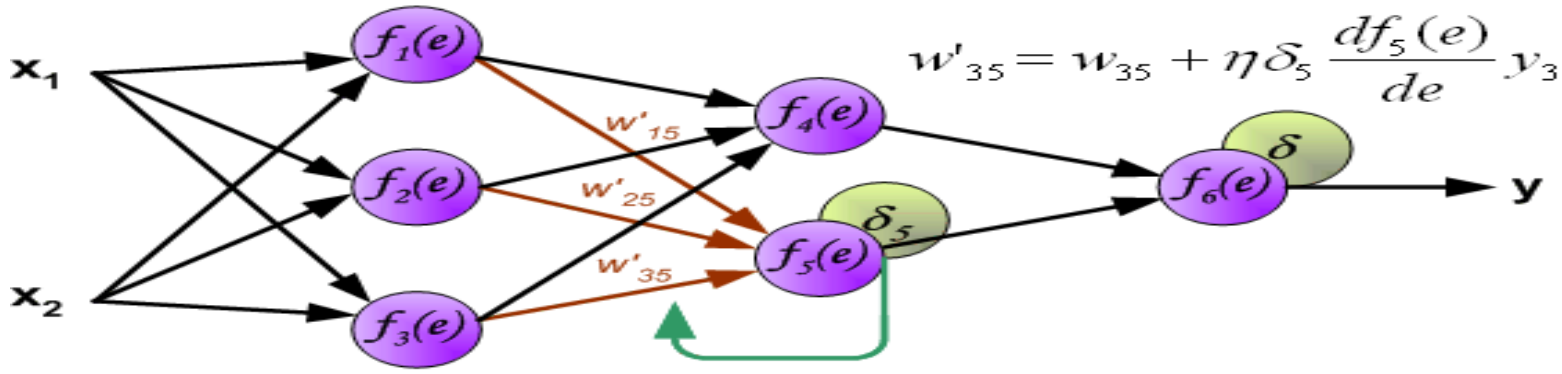
$$w'_{34} = w_{34} + \eta \delta_4 \frac{df_4(e)}{de} y_3$$

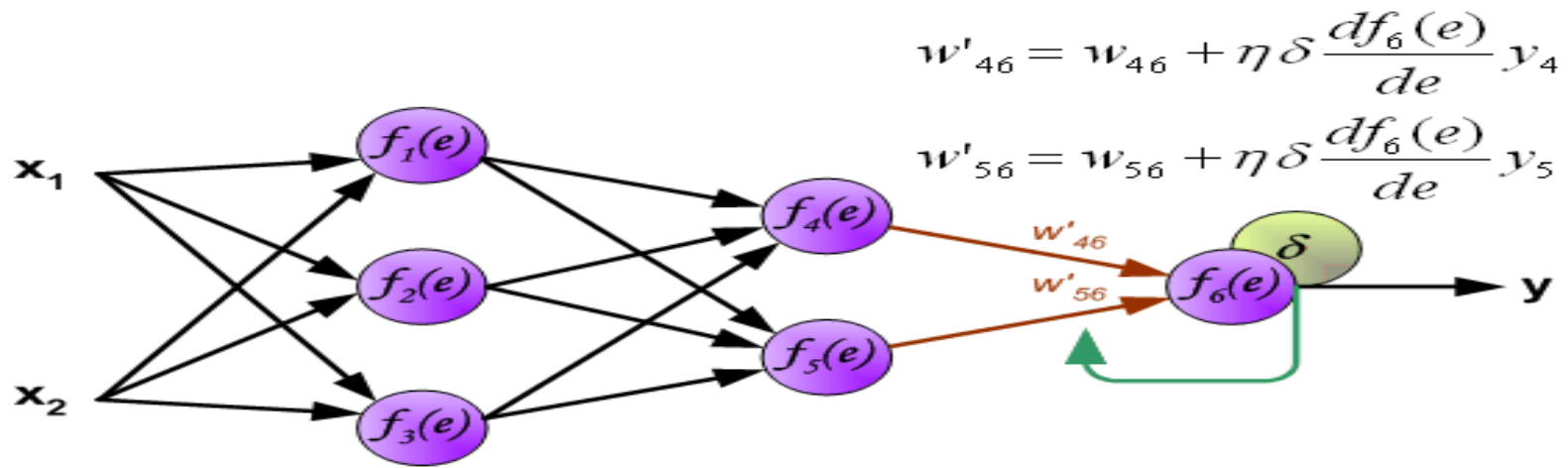


$$w'_{15} = w_{15} + \eta \delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta \delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta \delta_5 \frac{df_5(e)}{de} y_3$$





Coefficient η affects network teaching speed. There are a few techniques to select this parameter. The first method is to start teaching process with large value of the parameter. While weights coefficients are being established the parameter is being decreased gradually.

The second, more complicated, method starts teaching with small parameter value. During the teaching process the parameter is being increased when the teaching is advanced and then decreased again in the final stage.

Training Algorithm 1

- Step 0: Initialize the weights to small random values
- Step 1: Feed the training sample through the network and determine the final output
- Step 2: Compute the error for each output unit, for unit k it is:

The diagram shows the error calculation formula for unit k, $\delta_k = (t_k - y_k)f'(y_{in_k})$, enclosed in a light blue box. Three orange arrows point from labels to parts of the formula: 'Required output' points to t_k , 'Actual output' points to y_k , and 'Derivative of f' points to f' .

$$\delta_k = (t_k - y_k)f'(y_{in_k})$$

Required output

Actual output

Derivative of f

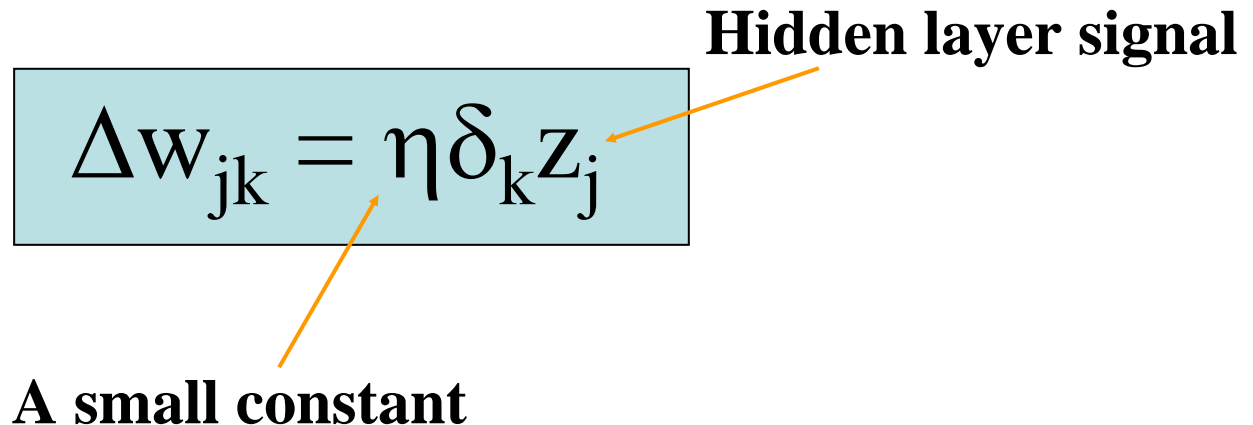
Training Algorithm 2

- Step 3: Calculate the weight correction term for each output unit, for unit k it is:

$$\Delta w_{jk} = \eta \delta_k z_j$$

Hidden layer signal

A small constant

The diagram features a light blue rectangular box containing the mathematical formula $\Delta w_{jk} = \eta \delta_k z_j$. Two orange arrows point from external text to parts of the formula: one points from the text 'A small constant' to the Greek letter η , and another points from the text 'Hidden layer signal' to the variable z_j .

Training Algorithm 3

- Step 4: Propagate the delta terms (errors) back through the weights of the hidden units where the delta input for the j^{th} hidden unit is:

$$\delta_{\text{in}_j} = \sum_{k=1}^m \delta_k w_{jk}$$

The delta term for the j^{th} hidden unit is:

$$\delta_j = \delta_{\text{in}_j} f'(z_{\text{in}_j})$$

where

$$f'(z_{\text{in}_j}) = f(z_{\text{in}_j}) [1 - f(z_{\text{in}_j})]$$

Training Algorithm 4

- Step 5: Calculate the weight correction term for the hidden units:

$$\Delta w_{ij} = \eta \delta_j x_i$$

- Step 6: Update the weights:

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

- Step 7: Test for stopping (maximum cycles, small changes, etc)

Options

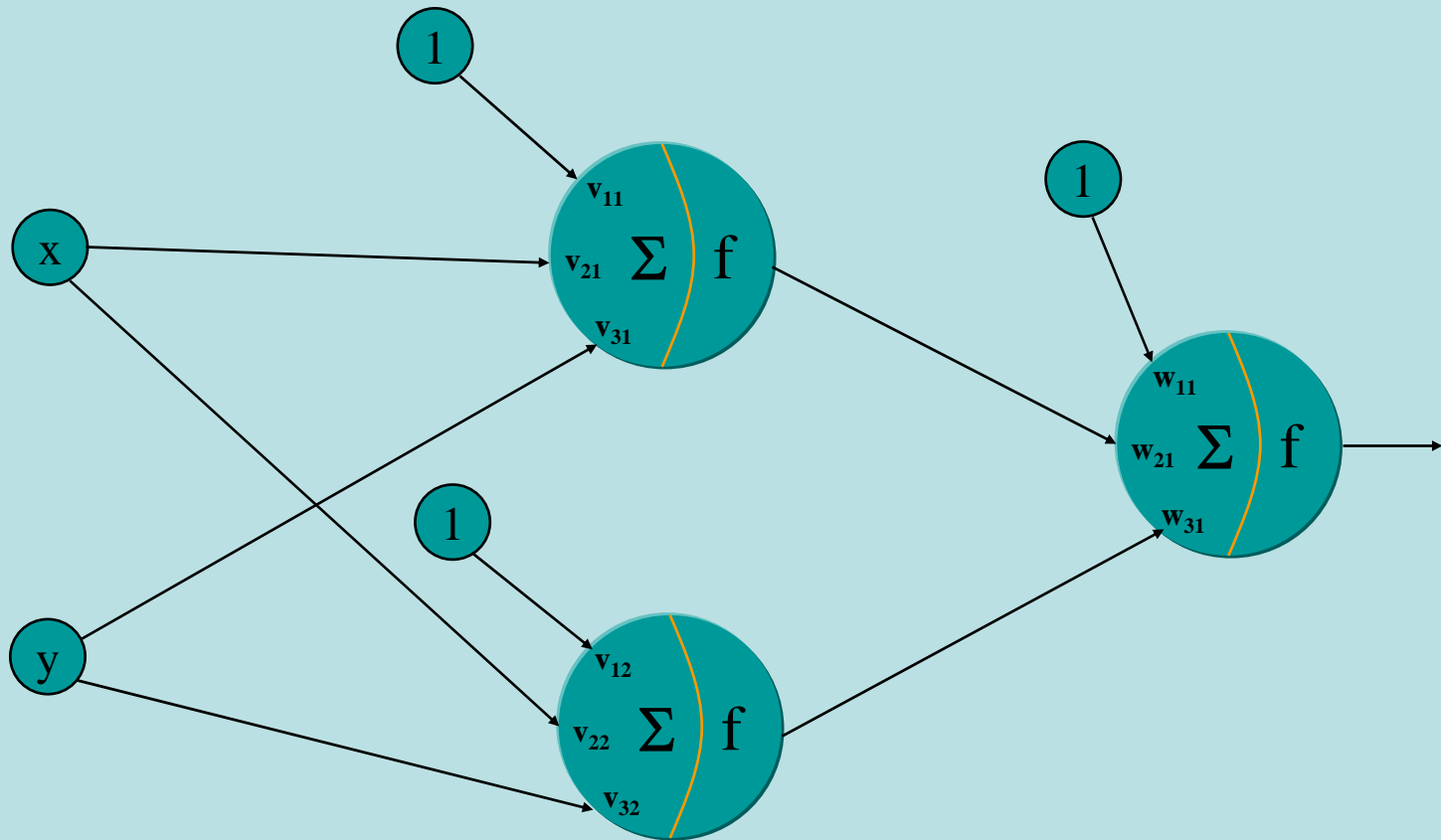
- There are a number of options in the design of a backprop system
 - Initial weights – best to set the initial weights (and all other free parameters) to random numbers inside a small range of values (say: -0.5 to 0.5)
 - Number of cycles – tend to be quite large for backprop systems
 - Number of neurons in the hidden layer – as few as possible

Example

- The XOR function could not be solved by a single layer perceptron network
- The function is:

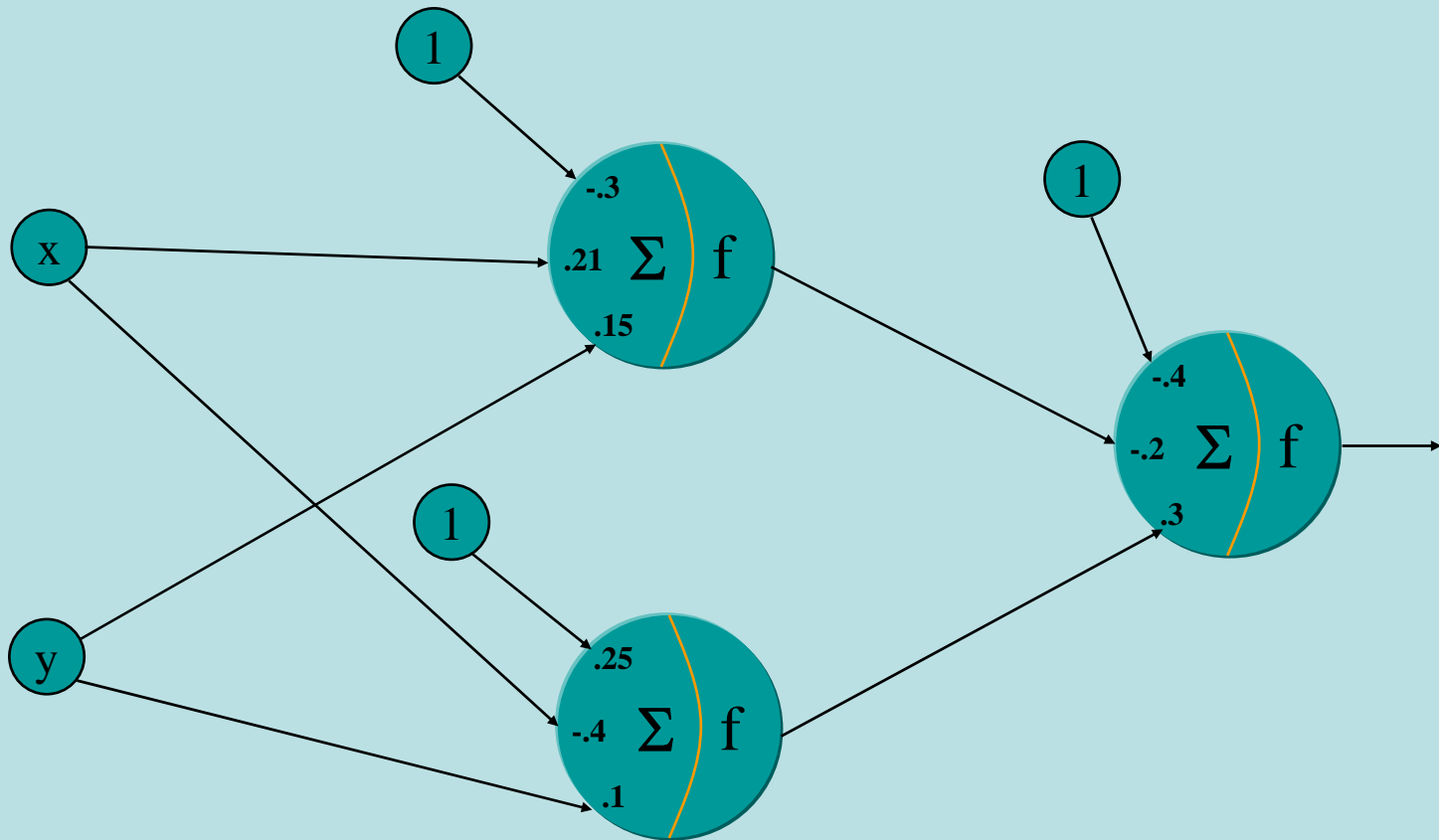
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

XOR Architecture

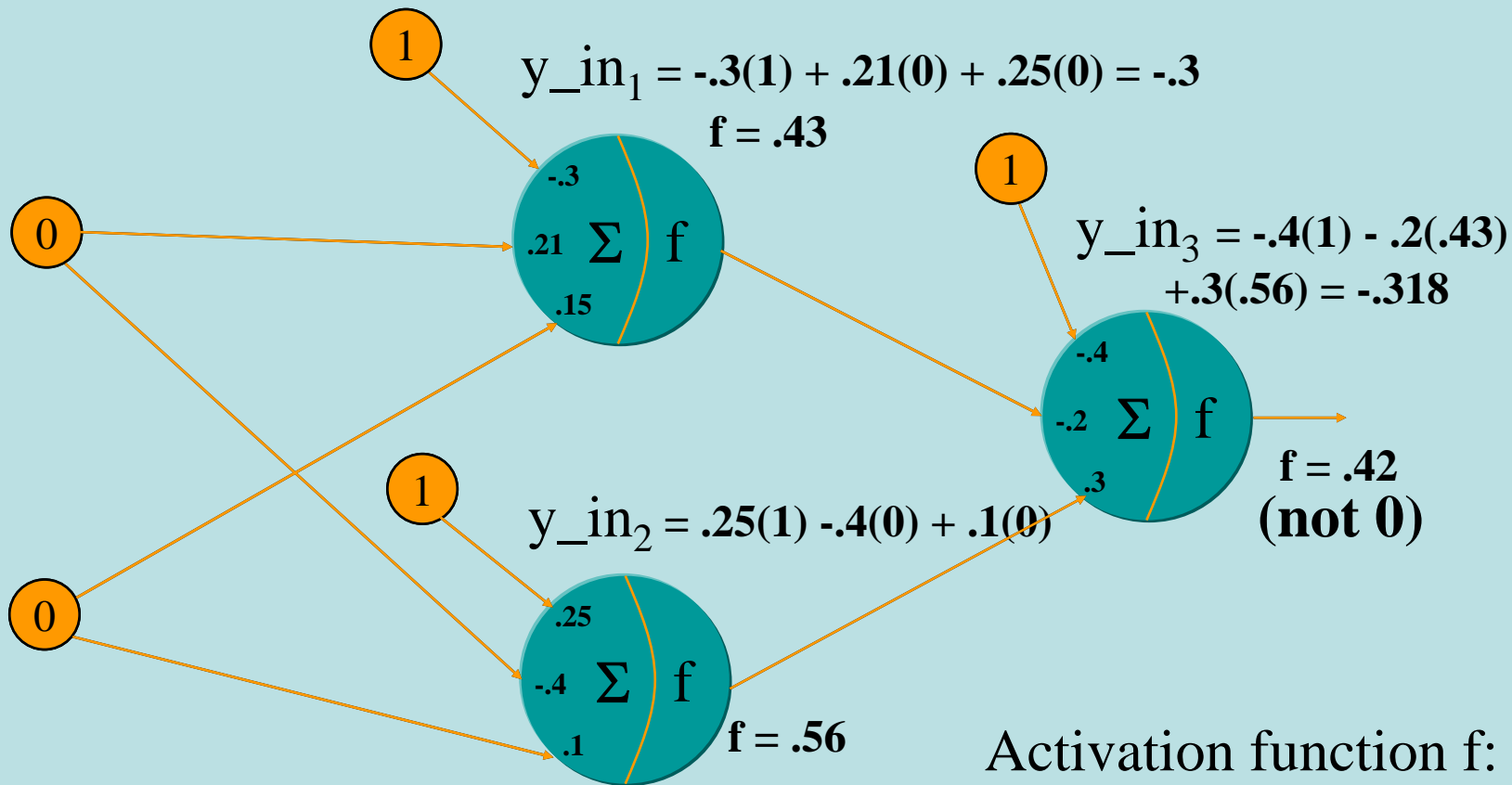


Initial Weights

- Randomly assign small weight values:



Feedforward – 1st Pass

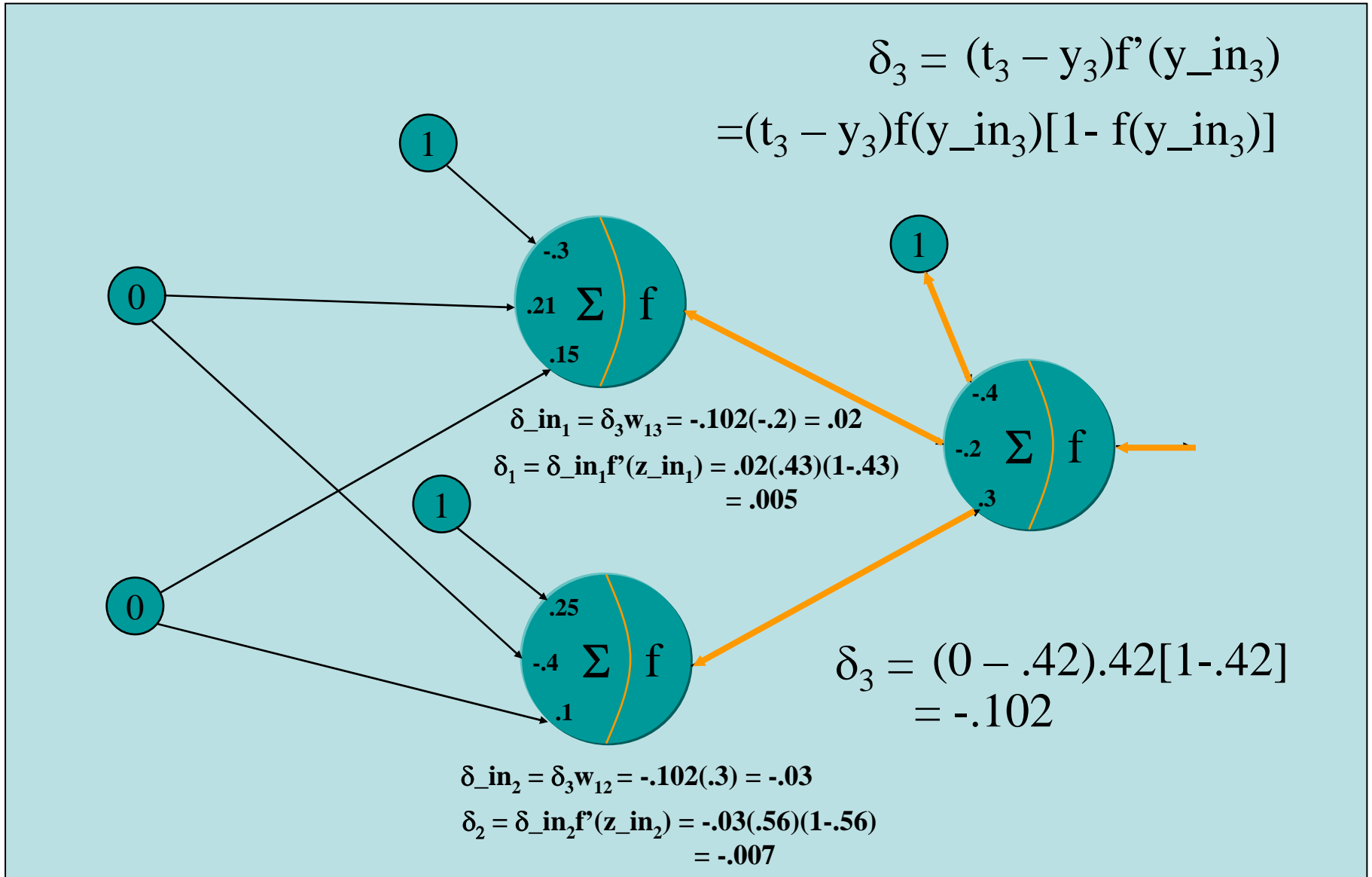


Activation function f:

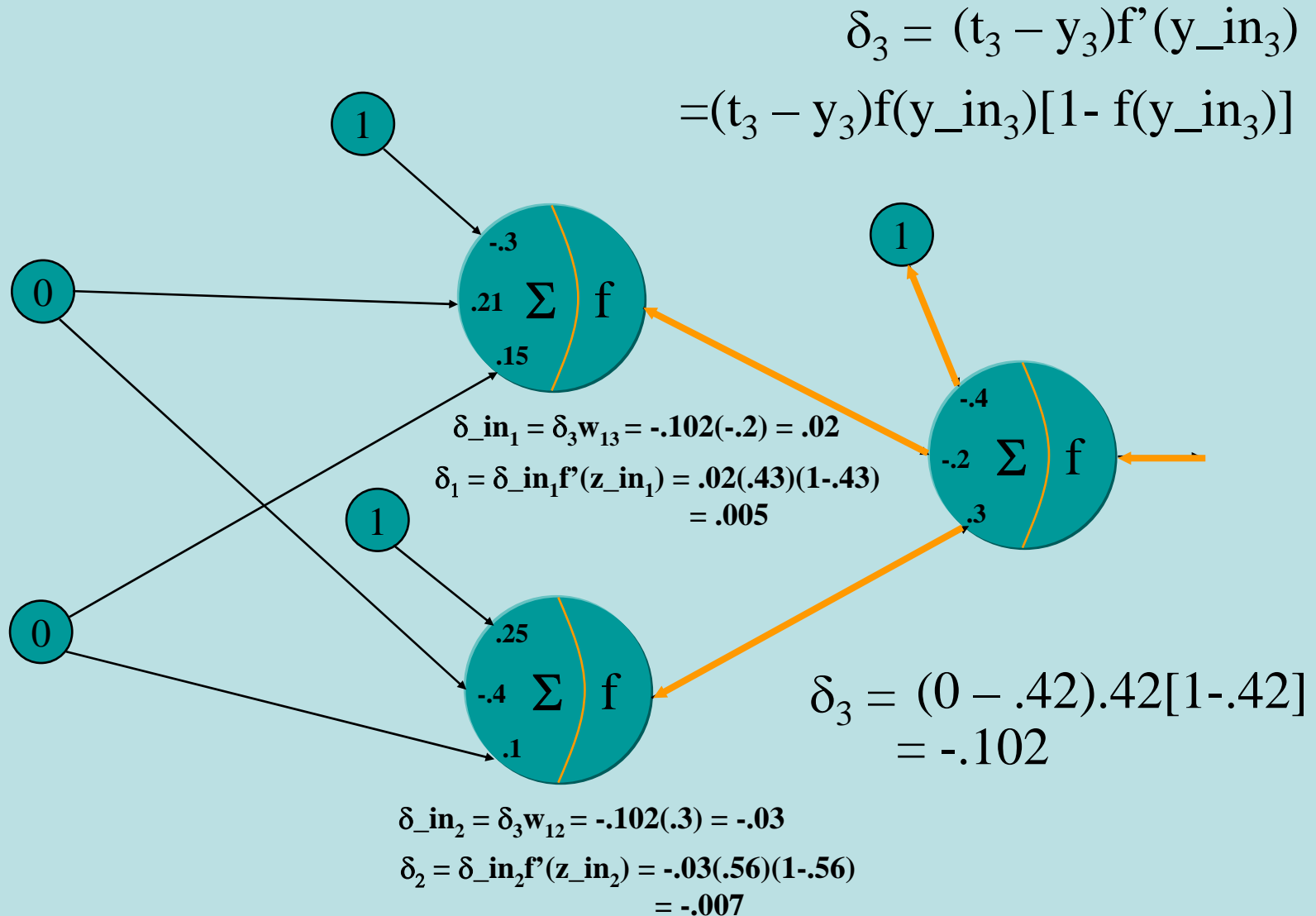
$$f(y_j) = \frac{1}{1 + e^{y_{in_i}}}$$

Training Case: (0 0 0)

Backpropagate



Update the Weights – First Pass



Final Result

- After about 500 iterations:

