# Multiple-Layer Networks
and
# Backpropagation Algorithms

# Multiple-Layer Networks
and
# Backpropagation Algorithms

Backpropagation is the generalization of the Widrow-Hoff learning rule to **multiple-layer networks** and **nonlinear differentiable transfer functions**.

Input vectors and the corresponding target vectors are used to train a network until it can **approximate a function**, associate input vectors with specific output vectors, or **classify** input vectors in an appropriate way as defined by you.

# Architecture

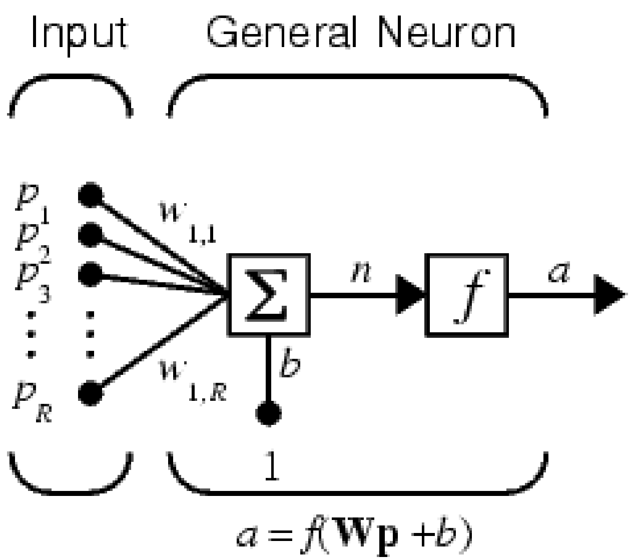
This section presents the architecture of the network that is most commonly used with the backpropagation algorithm **–**

**the multilayer feedforward network**

# Architecture

## Neuron Model

An elementary neuron with R inputs is shown below. Each input is weighted with an appropriate w. The sum of the weighted inputs and the bias forms the input to the transfer function f. Neurons can use any **differentiable transfer function** f to generate their output.




Where

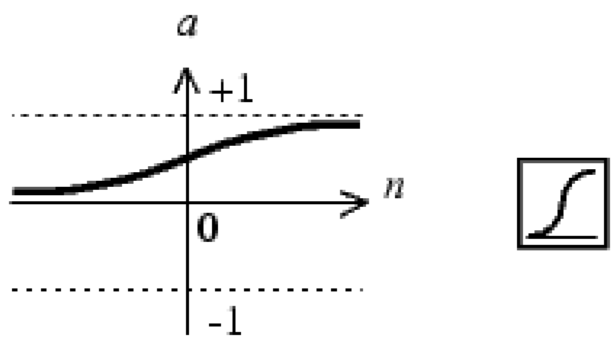
$R$ = number of elements in input vector

# Architecture

## Neuron Model

### Transfer Functions (Activition Function)

Multilayer networks often use **the log-sigmoid** transfer function **logsig**. The function logsig generates outputs between **0** and **1** as the neuron's net input goes from negative to positive infinity




$a = logsig(n)$

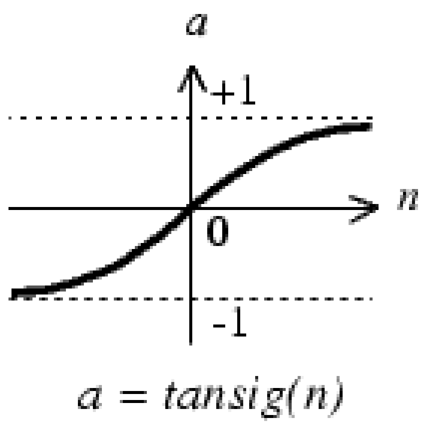
Log-Sigmoid Transfer Function

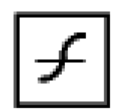
# Architecture

## Neuron Model

### Transfer Functions (Activition Function)

Alternatively, multilayer networks can use **the tan-sigmoid** transfer function-**tansig**.
The function logsig generates outputs between **-1** and **+1** as the neuron's net input goes from negative to positive infinity

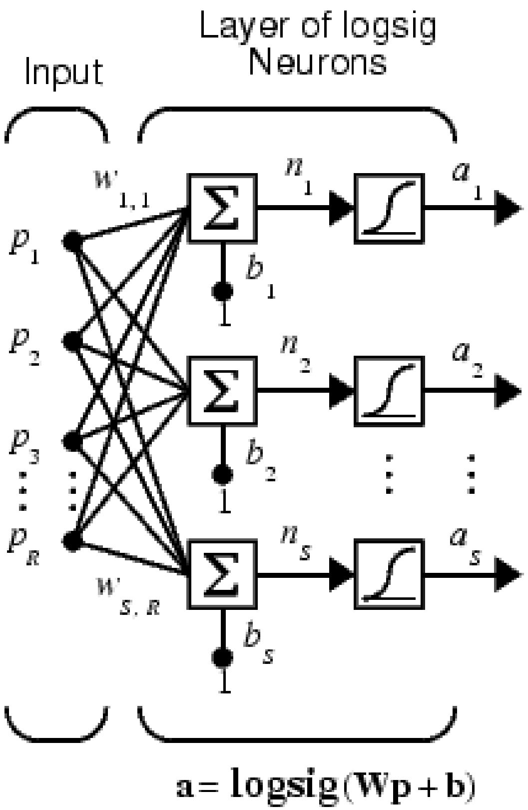


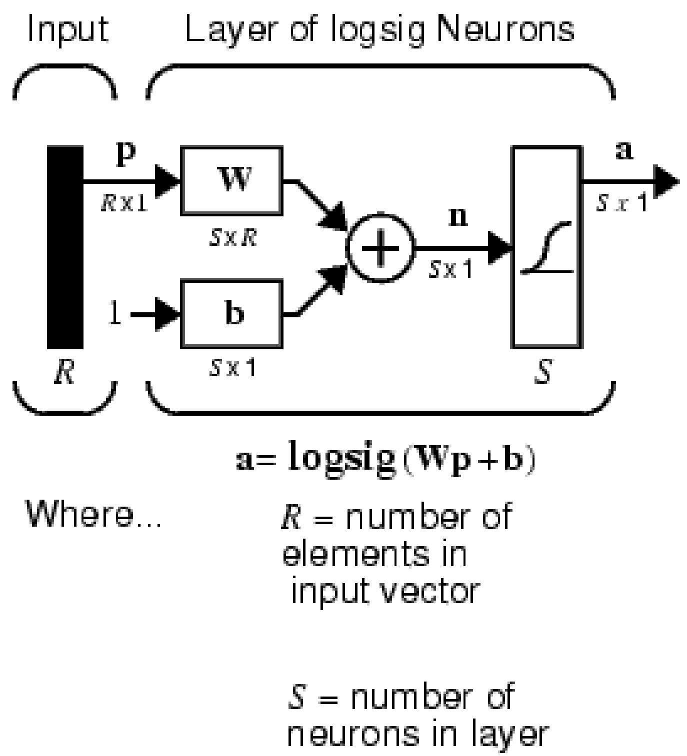



Tan-Sigmoid Transfer Function

# Architecture
## Feedforward Network

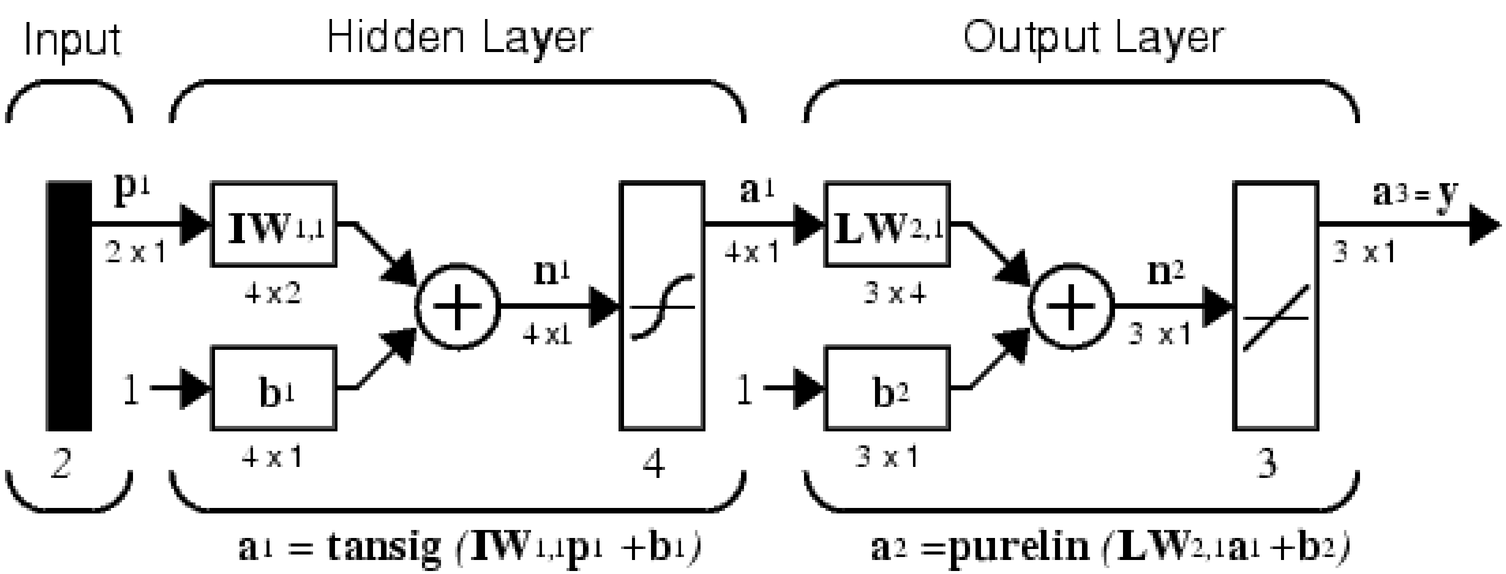
**A single-layer network of S logsig neurons** having R inputs is shown below in full detail on the left and with a layer diagram on the right.



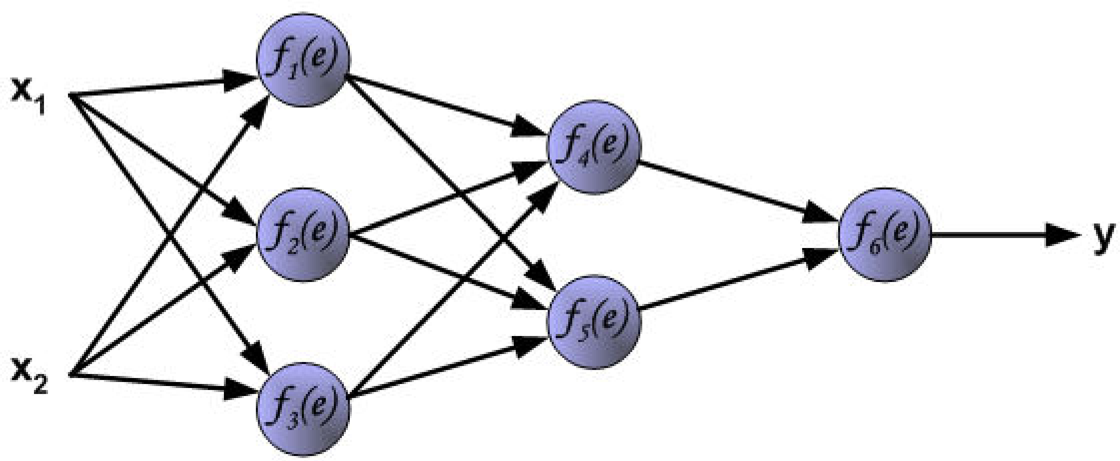
# Architecture
## Feedforward Network

Feedforward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons.
Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear and linear relationships between input and output vectors.
The linear output layer lets the network produce values outside the range -1 to +1. On the other hand, if you want to constrain the outputs of a network (such as between 0 and 1), then the output layer should use a sigmoid transfer function (such as logsig).



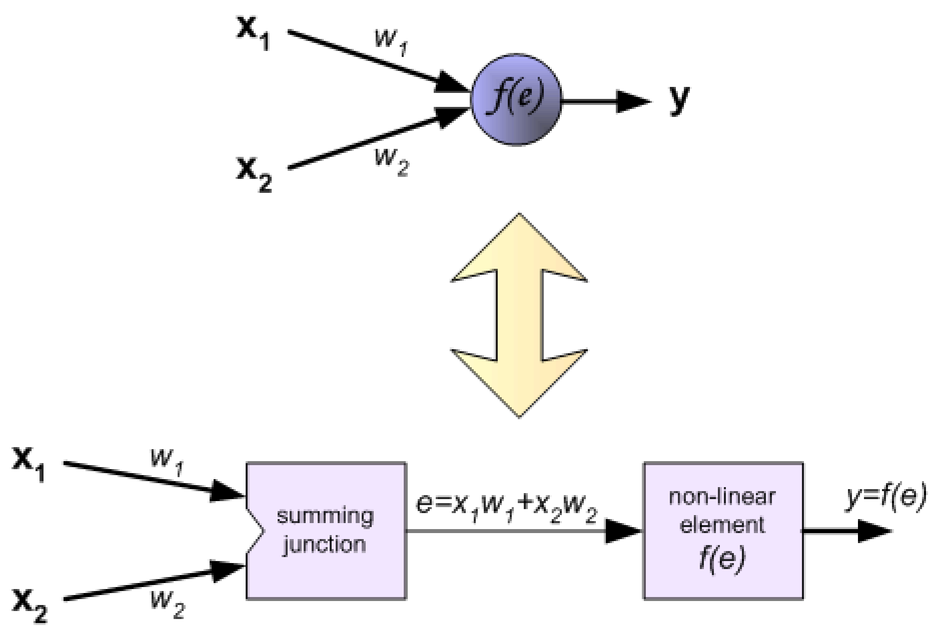
# Learning Algorithm: Backpropagation

The following slides describes **teaching process** of multi-layer neural network employing **backpropagation** algorithm. To illustrate this process the three layer neural network with two inputs and one output,which is shown in the picture below, is used:



# Learning Algorithm: Backpropagation

Each neuron is composed of two units. First unit adds products of weights coefficients and input signals. The second unit realise nonlinear function, called neuron transfer (activation) function. Signal *e* is adder output signal, and *y = f(e)* is output signal of nonlinear element. Signal *y* is also output signal of neuron.



# Learning Algorithm: Backpropagation

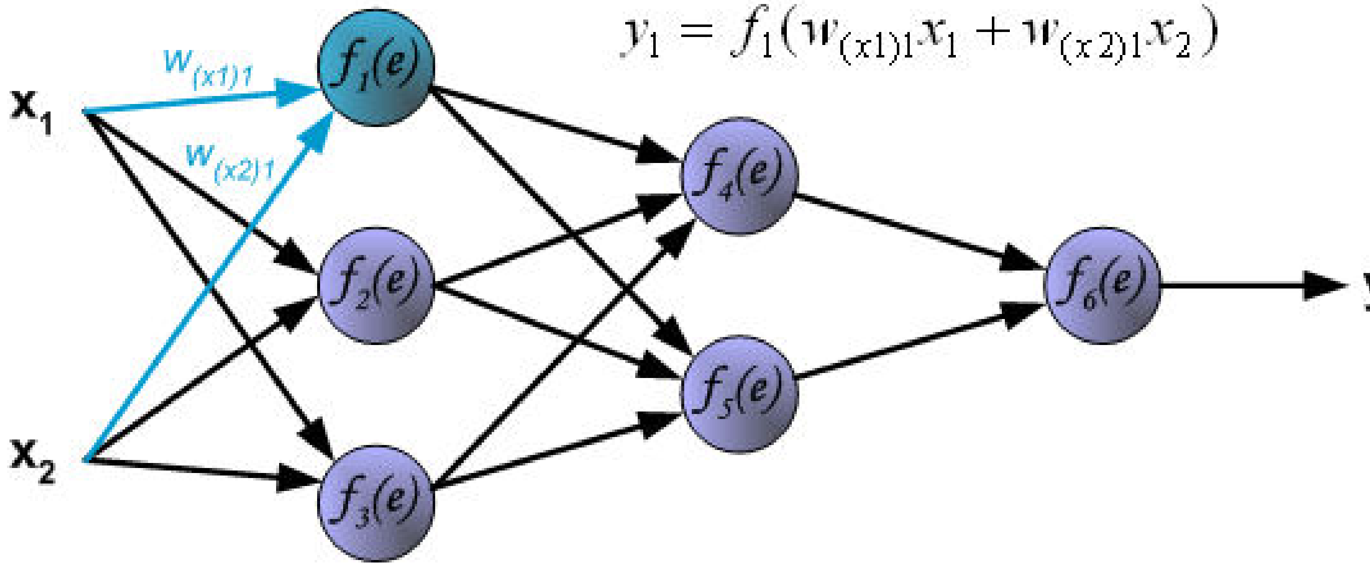
To teach the neural network we need training data set. The training data set consists of input signals ($x_1$ and $x_2$ ) assigned with corresponding target (desired output) $z$.

The network training is an iterative process. In each iteration weights coefficients of nodes are modified using new data from training data set. Modification is calculated using algorithm described below:

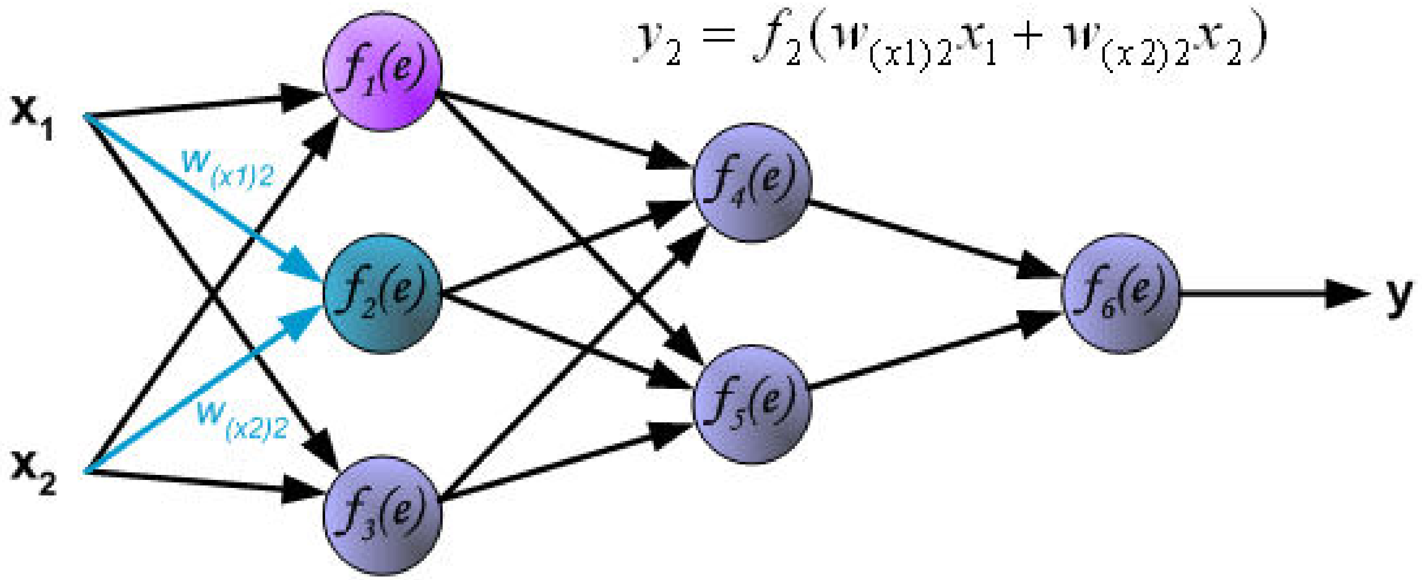
Each teaching step starts with forcing both input signals from training set. After this stage we can determine output signals values for each neuron in each network layer.

# Learning Algorithm: Backpropagation

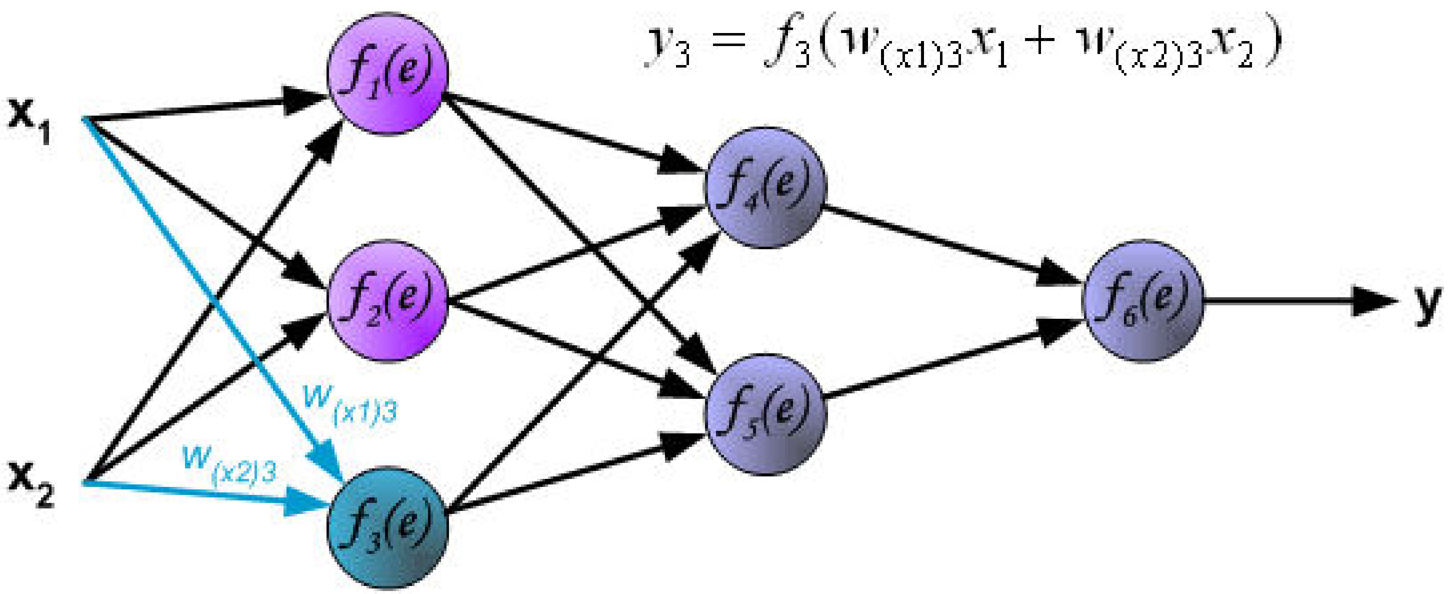
Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input $x_m$ and neuron $n$ in input layer. Symbols $y_n$ represents output signal of neuron $n$.



# Learning Algorithm: Backpropagation

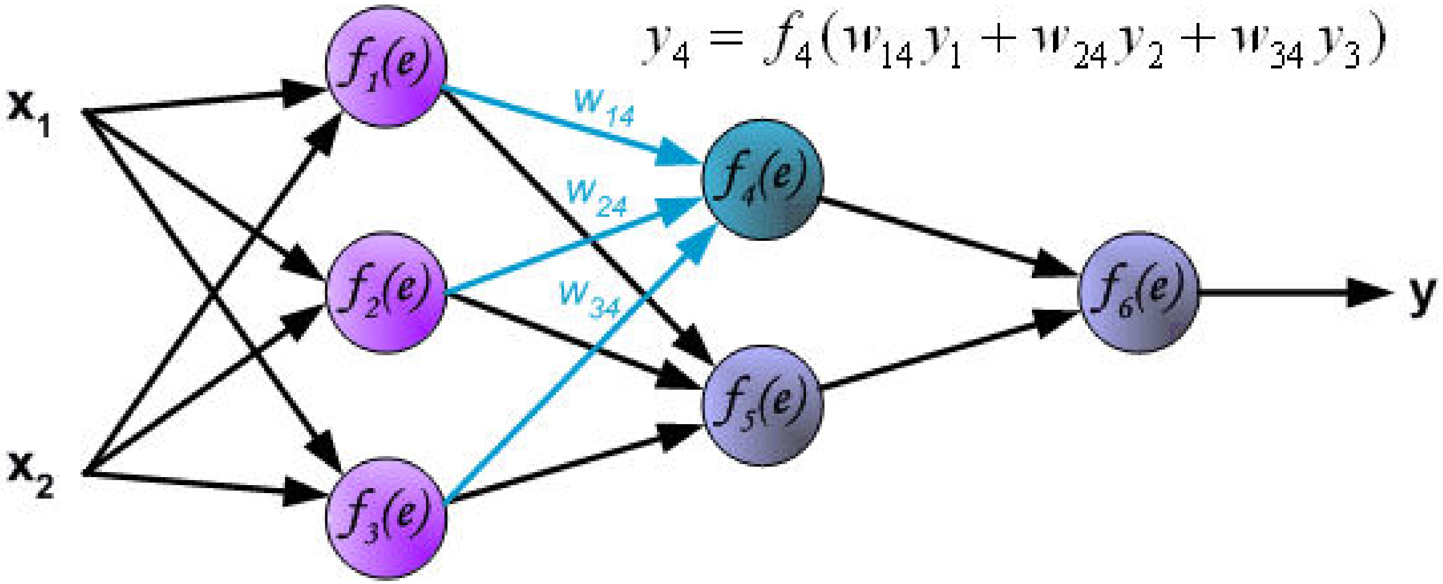


# Learning Algorithm: Backpropagation

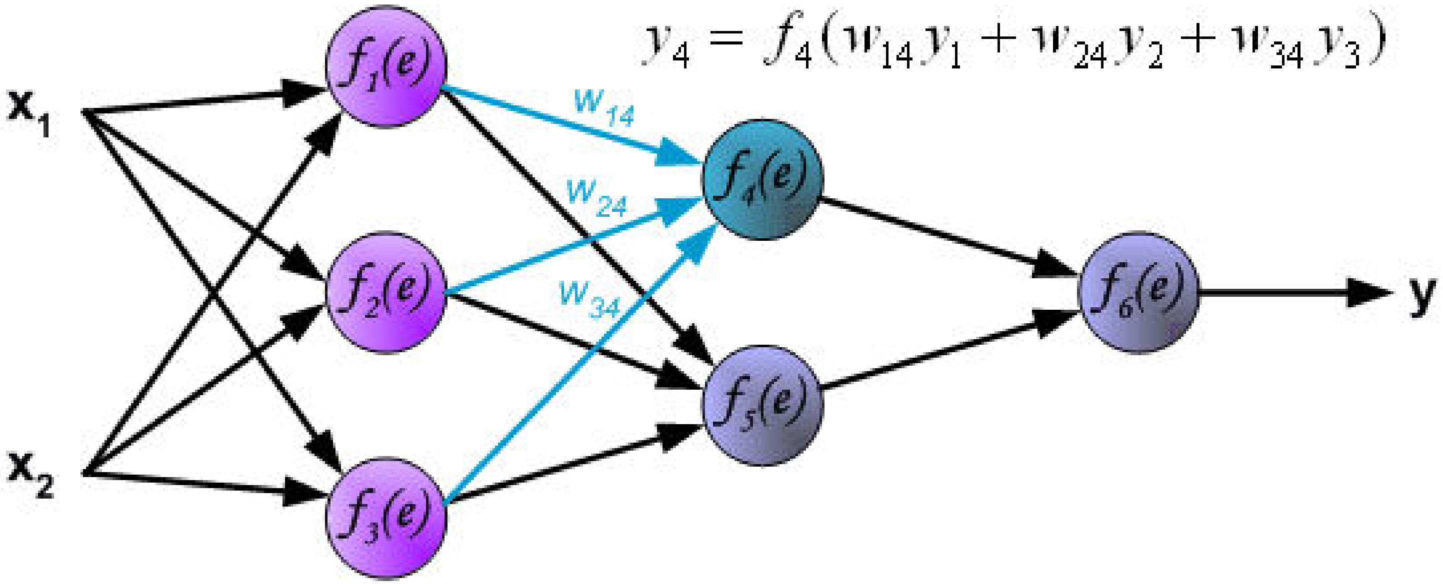


# Learning Algorithm: Backpropagation

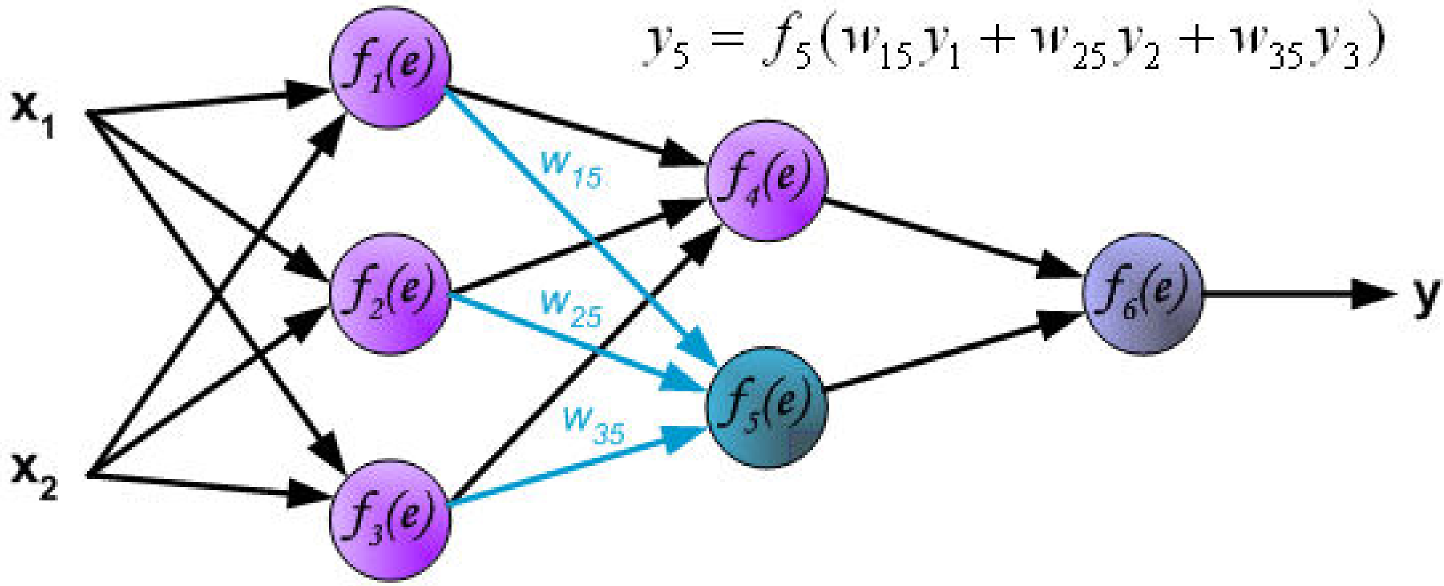
Propagation of signals through the hidden layer. Symbols $w_{mn}$ represent weights of connections between output of neuron *m* and input of neuron *n* in the next layer.



# Learning Algorithm: Backpropagation

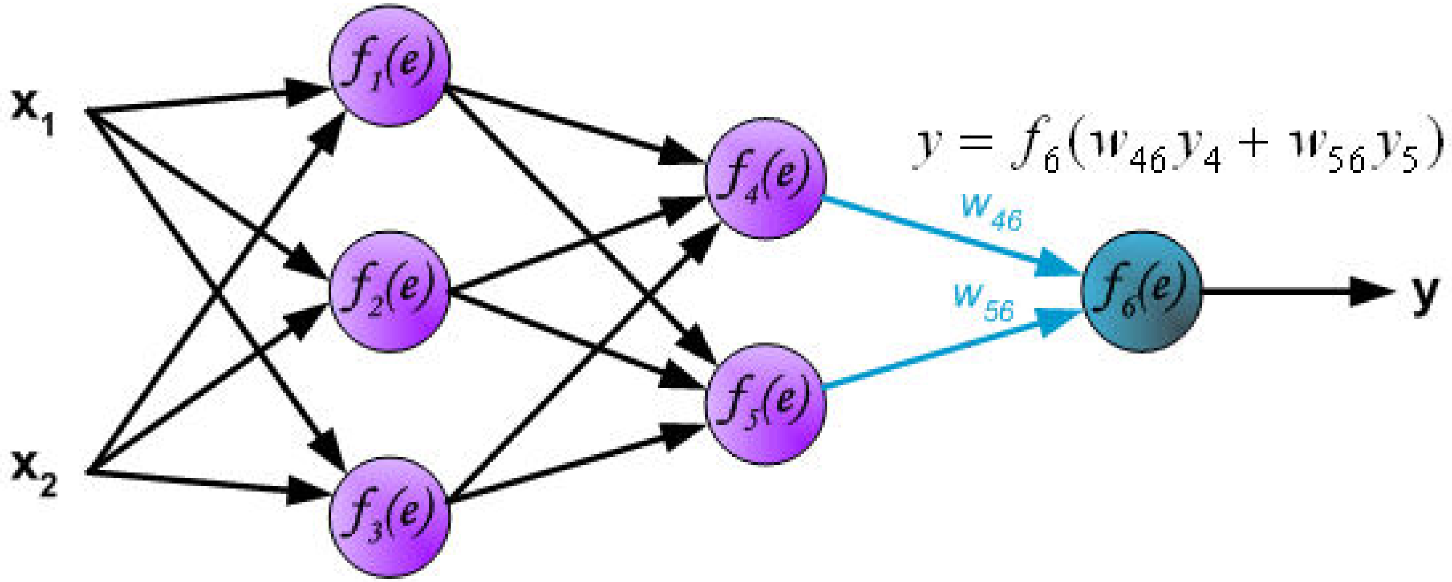


# Learning Algorithm: Backpropagation



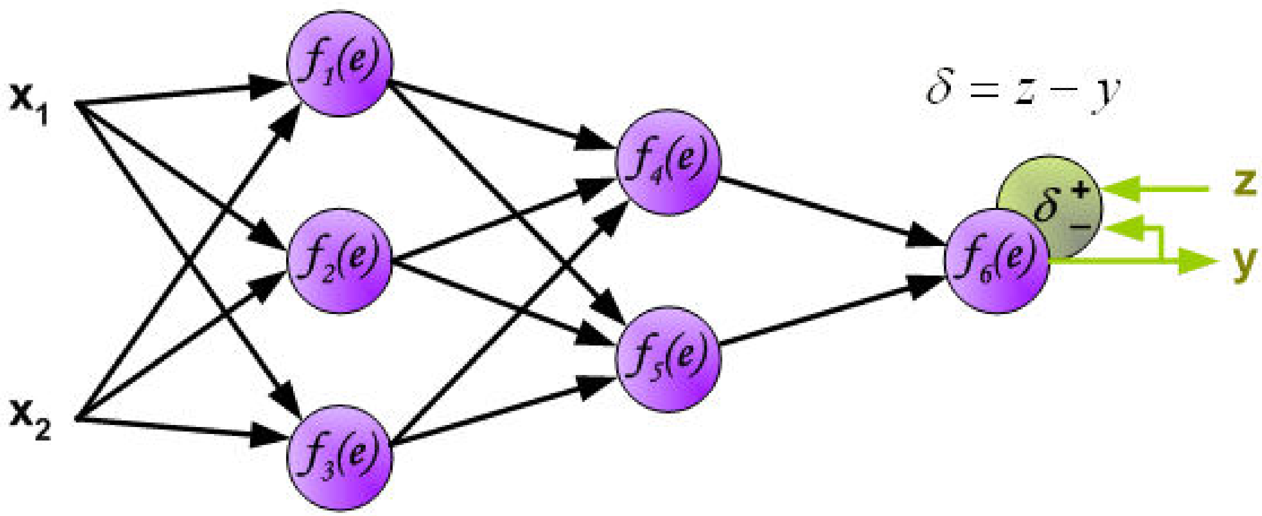
# Learning Algorithm: Backpropagation

Propagation of signals through the output layer.



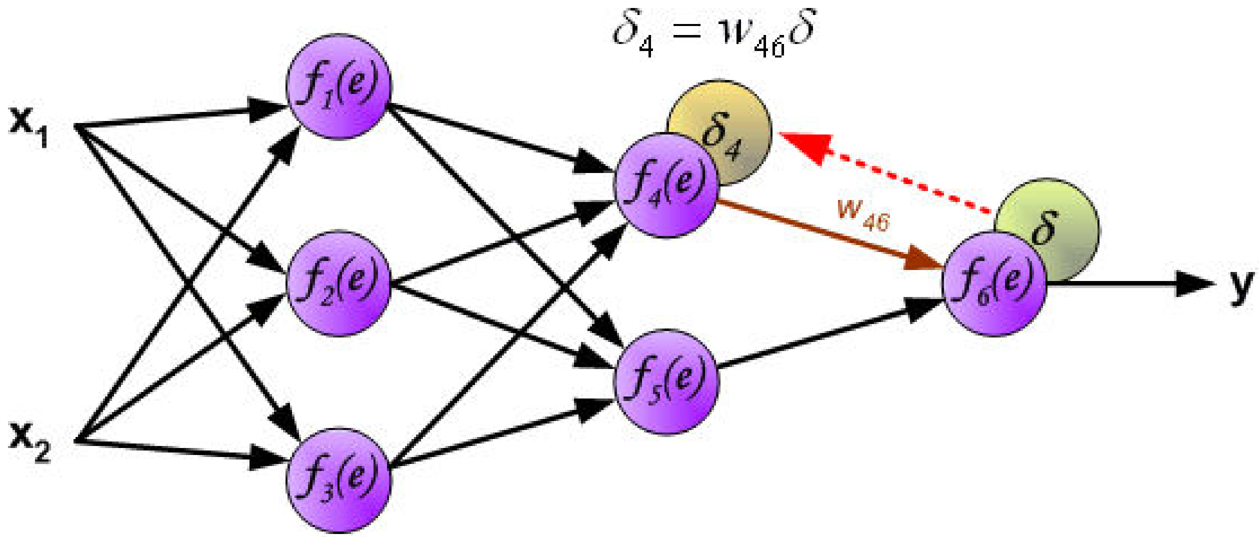
# Learning Algorithm: Backpropagation

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 signal *d* of output layer neuron



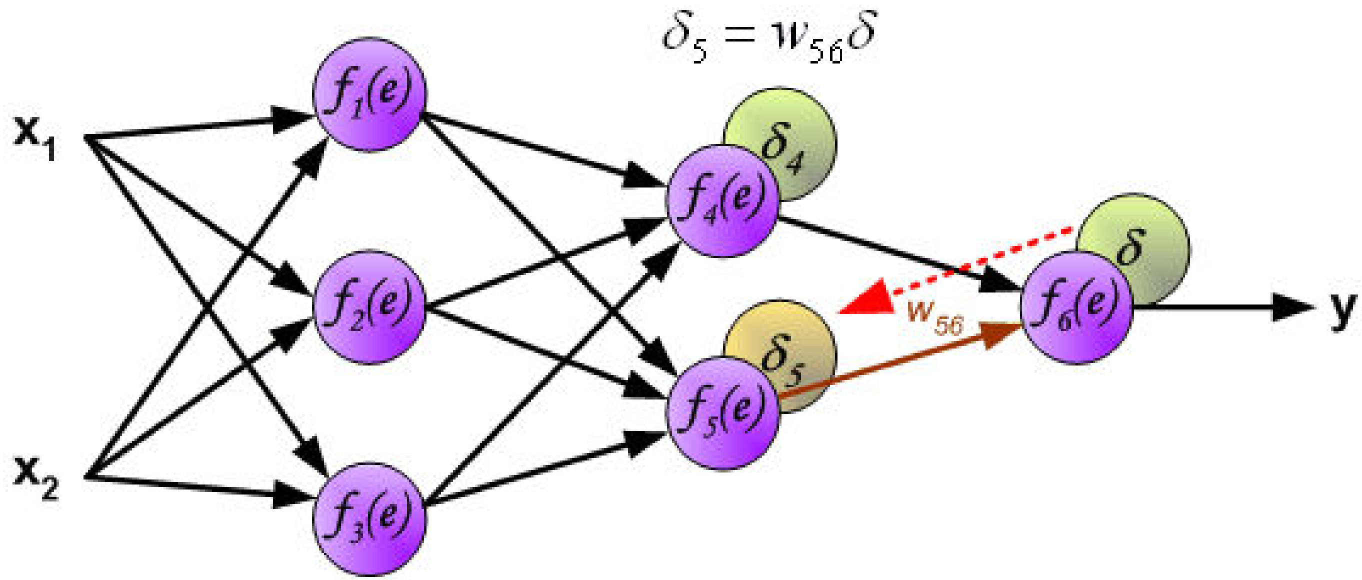
# Learning Algorithm: Backpropagation

The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.



# Learning Algorithm: Backpropagation

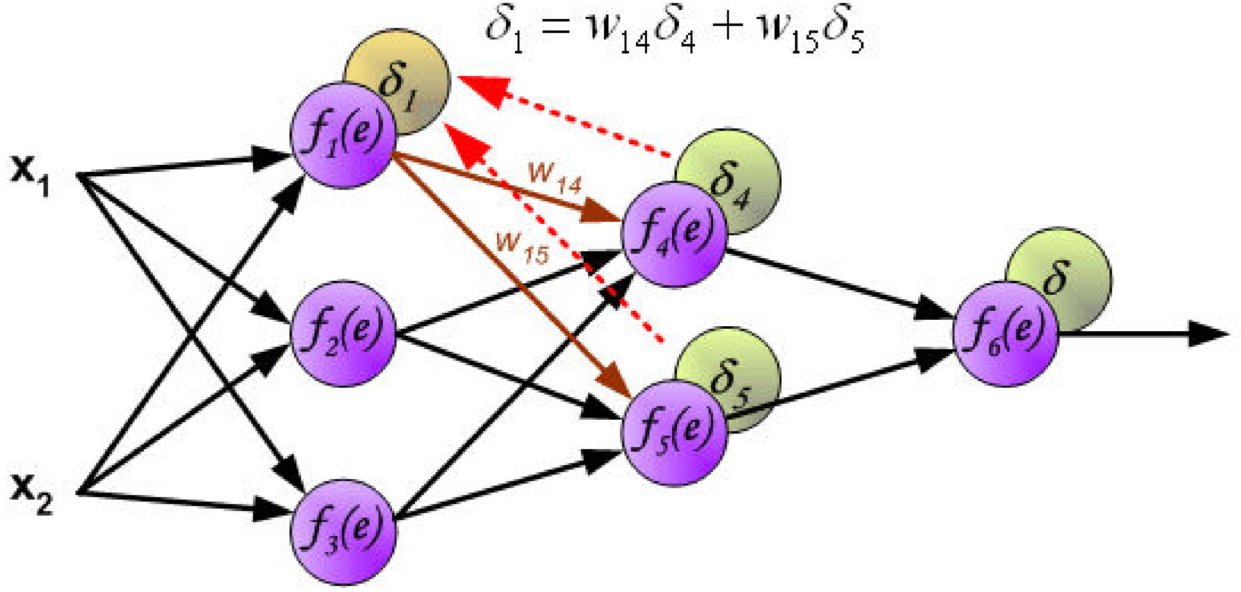
The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.



# Learning Algorithm: Backpropagation

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:

$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$$

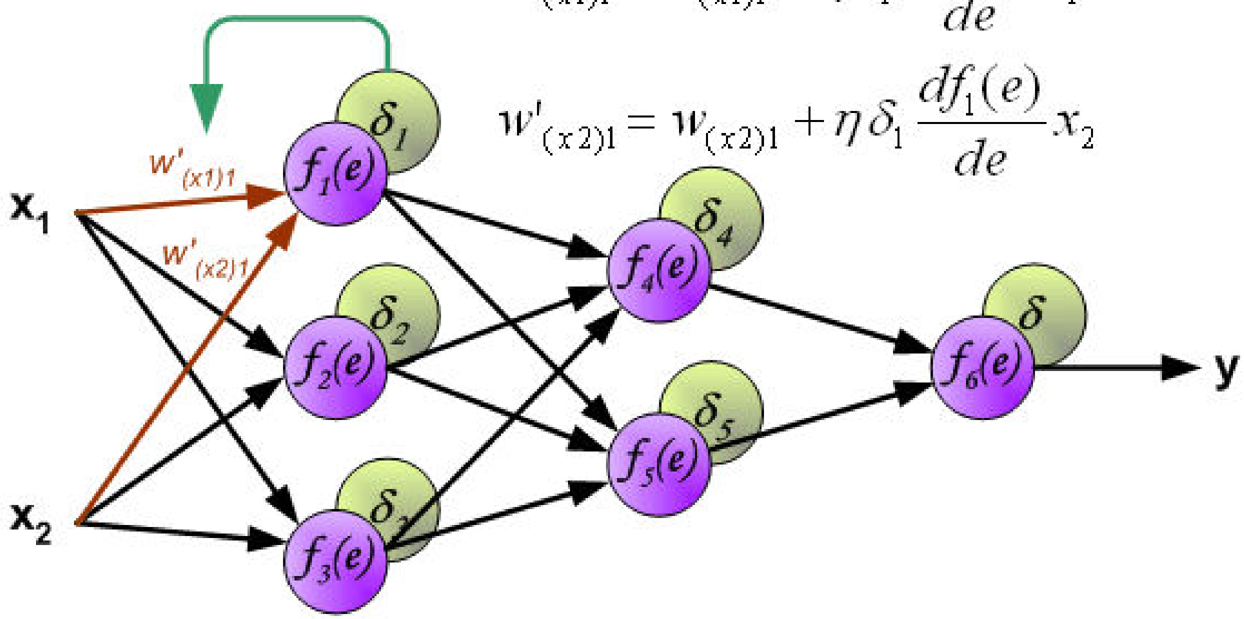


# Learning Algorithm: Backpropagation

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).

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$

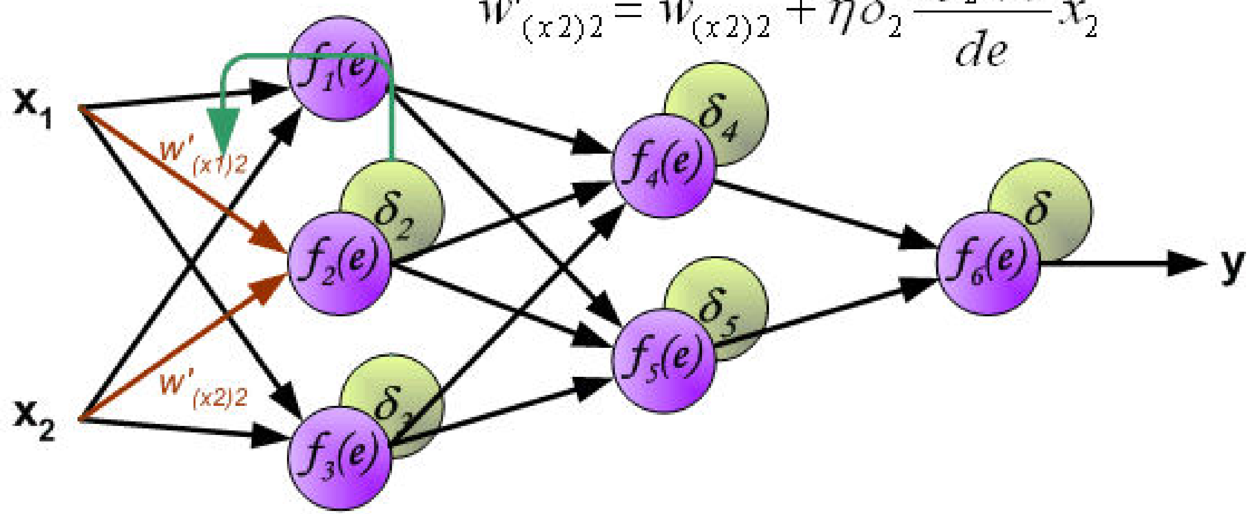


# Learning Algorithm: Backpropagation

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).

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



# Learning Algorithm: Backpropagation

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).

$$w'_{46} = w_{46} + \eta\,\delta\,\frac{df_6(e)}{de}\,y_4$$

$$w'_{56} = w_{56} + \eta\,\delta\,\frac{df_6(e)}{de}\,y_5$$