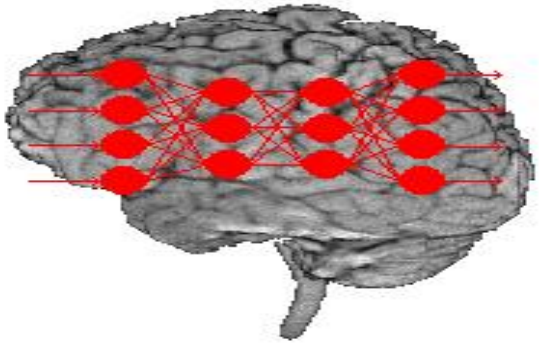
Deep Learning
Lecture-7&8

Artificial Neural Network



Asst. Lect. Ali Al-khawaja

2025-2026

Class Room

Backpropagation Network

- In 1969 a method for learning in multi-layer network, Backpropagation, was invented by Bryson and Ho.

- The Backpropagation algorithm is a sensible approach for dividing the contribution of each weight.

- Works basically the same as perceptron

Backpropagation Learning Principles: **Hidden Layers** and Gradients

There are two **differences for the updating rule :**

1) The activation of the hidden unit is used instead of activation of the input value.

2) The rule contains a term for the gradient of the activation function.

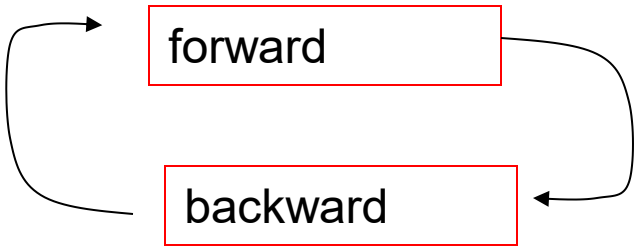
Backpropagation Network training

- 1. Initialize network with random weights
- 2. For all training cases (called examples):
  - **a.** Present training inputs to network and calculate output
  - **b.** For all layers (starting with output layer, back to input layer):
    - i. Compare network output with correct output (error function)
    - ii. Adapt weights in current layer

This is what you want

Backpropagation Learning Details

- Method for learning weights in feed-forward (FF) nets
- Can't use Perceptron Learning Rule
  - no teacher values are possible for hidden units
- Use gradient descent to minimize the error
  - propagate deltas to adjust for errors
    backward from outputs
    to hidden layers
    to inputs



Backpropagation Algorithm – Main Idea – error in hidden layers

The ideas of the algorithm can be summarized as follows :

1. Computes the **error term for the output units** using the observed error.

2. From output layer, repeat
   - propagating the error term back to the previous layer and
   - updating the weights between the two layers

   until the earliest hidden layer is reached.

Backpropagation Algorithm

- Initialize weights (typically random!)
- Keep doing epochs
  - For each example **e** in training set do
    - **forward pass** to compute
      - O = neural-net-output(network,e)
      - miss = (D-O) at each output unit
    - **backward pass** to calculate deltas to weights
    - update all weights
  - end
- until tuning set error stops improving

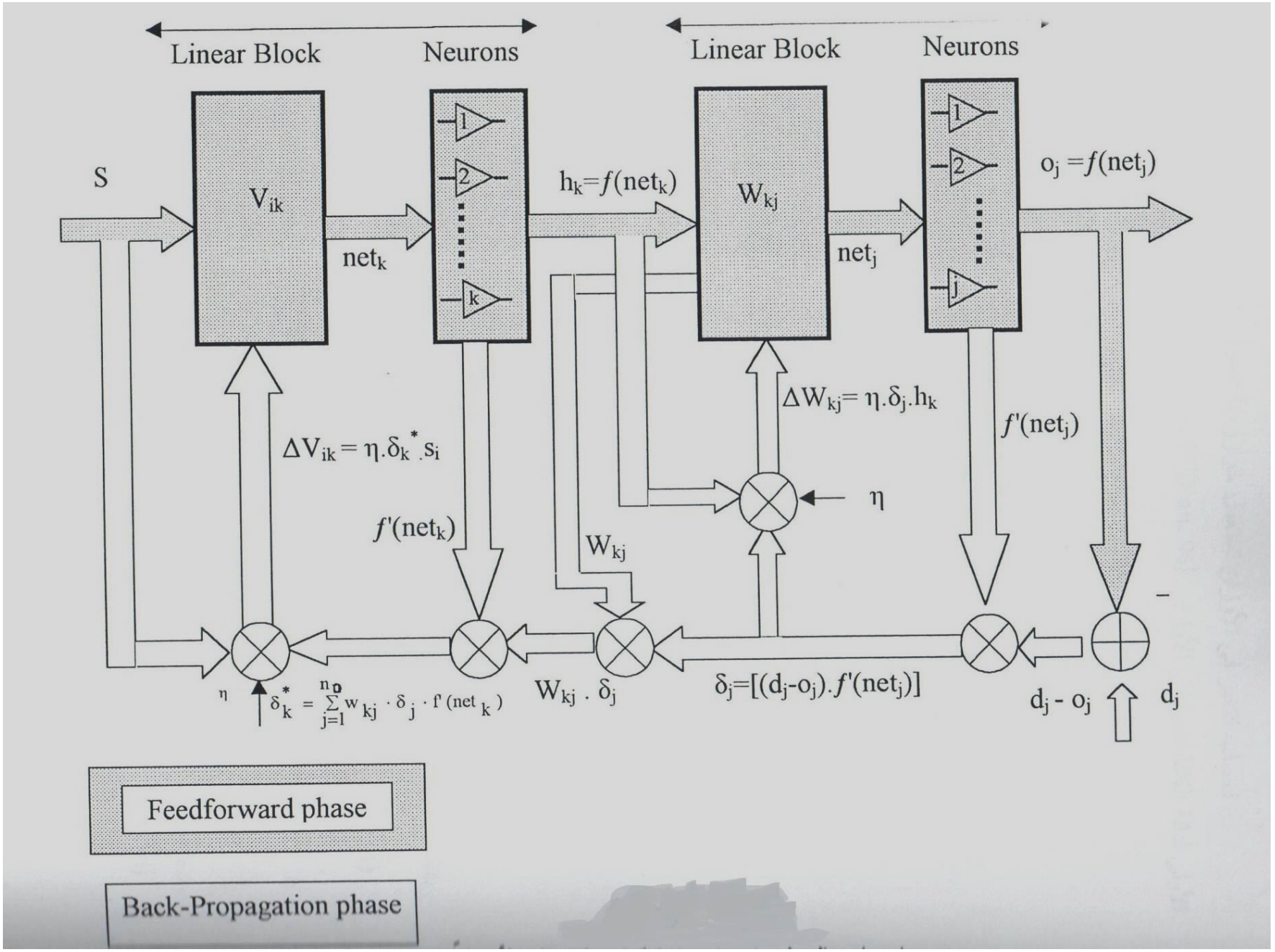
Forward pass explained earlier

Backward pass explained in next slide

Backward Pass

- Compute deltas to weights
  - from hidden layer
  - to output layer

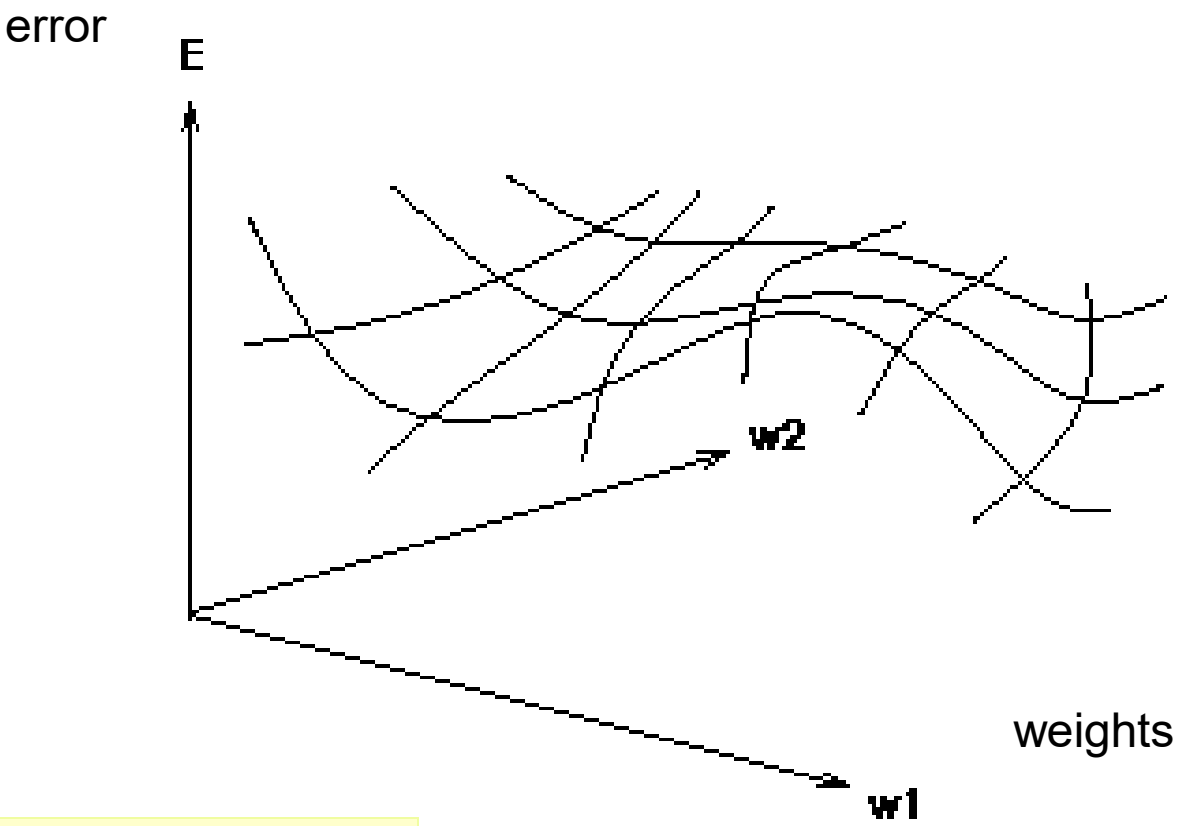
- Without changing any weights (yet), compute the actual contributions
  - within the hidden layer(s)
  - and compute deltas


Linear Block
Neurons
Linear Block
Neurons
S
V_{ik}
net_k
$h_k = f(net_k)$
W_{kj}
net_j
$o_j = f(net_j)$
1
2
k
1
2
j
$\Delta V_{ik} = \eta . \delta_k^* . s_i$
$\Delta W_{kj} = \eta . \delta_j . h_k$
$f'(net_j)$
$f'(net_k)$
η
W_{kj}
η
$\delta_k^* = \sum_{j=1}^{n_0} w_{kj} \cdot \delta_j \cdot f'(net_k)$
$W_{kj} . \delta_j$
$\delta_j = [(d_j - o_j) . f'(net_j)]$
$d_j - o_j$
d_j
Feedforward phase
Back-Propagation phase

Gradient Descent

- Think of the N weights as a point in an N-dimensional space
- Add a dimension for the observed error
- Try to minimize your position on the “error surface”

Error Surface



Error as function of weights in multidimensional space

Compute deltas

Gradient

- Trying to make error decrease the fastest
- Compute:
  - $Grad_E$ = [dE/dw1, dE/dw2, . . ., dE/dwn]
- Change i-th weight by
  - $delta_{wi}$ = -*alpha* * dE/dwi

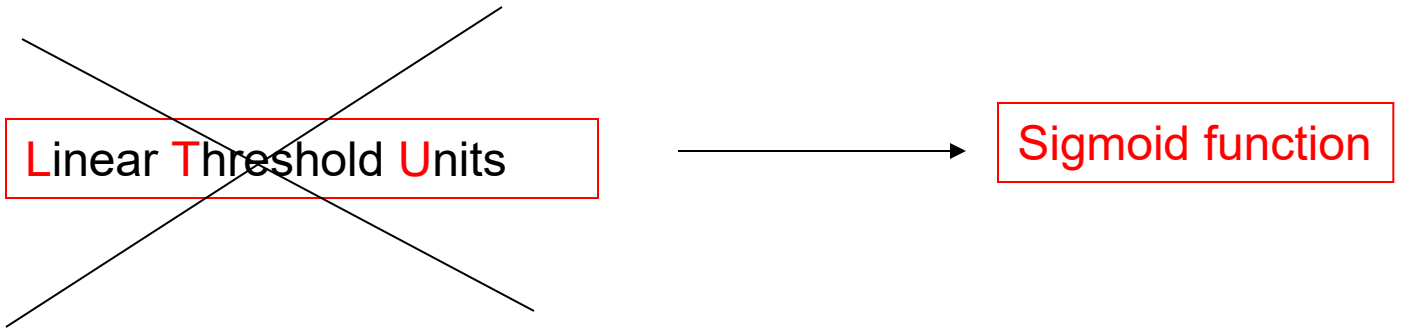
Derivatives of error for weights

- We need a derivative!
- Activation function must be **continuous**, differentiable, non-decreasing, and easy to compute

Can't use LTU

- To effectively assign credit / blame to units in hidden layers, we want to look at the first derivative of the activation function

- Sigmoid function is easy to differentiate and easy to compute forward



Updating hidden-to-output

- We have teacher supplied desired values

- $delta_{wji} = \alpha * a_j * (D_i - Oi) * g'(in_i)$
  $= \alpha * a_j * (D_i - O_i) * O_i * (1 - O_i)$

  – for sigmoid the derivative is, $g'(x) = g(x) * (1 - g(x))$

alpha

Here we have general formula with derivative, next we use for sigmoid

miss

derivative

Updating interior weights

- Layer k units provide values to all layer k+1 units
  - "miss" is ***sum of misses*** from all units on k+1
  - $miss_j = \Sigma [a_i(1- a_i) (D_i - a_i) w_{ji}]$
  - weights coming into this unit are adjusted based on their contribution

$delta_{kj} = \alpha * I_k * a_j * (1 - a_j) * miss_j$

For layer k+1

Compute deltas

How do we pick α?

1. Tuning set, or
2. Cross validation, or
3. Small for slow, conservative learning

How many hidden layers?

- Usually just one (i.e., a 2-layer net)

- How many hidden units in the layer?
  - Too few ==> can't learn
  - Too many ==> poor generalization

How big a training set?

- Determine your target error rate, *e*
- Success rate is 1- *e*
- Typical training set approx. n/*e*, where n is the number of weights in the net
- Example:
  - *e* = 0.1, n = 80 weights
  - training set size 800

    trained until 95% correct training set classification

    should produce 90% correct classification

    on testing set (typical)

Learning Algorithm: Backpropagation

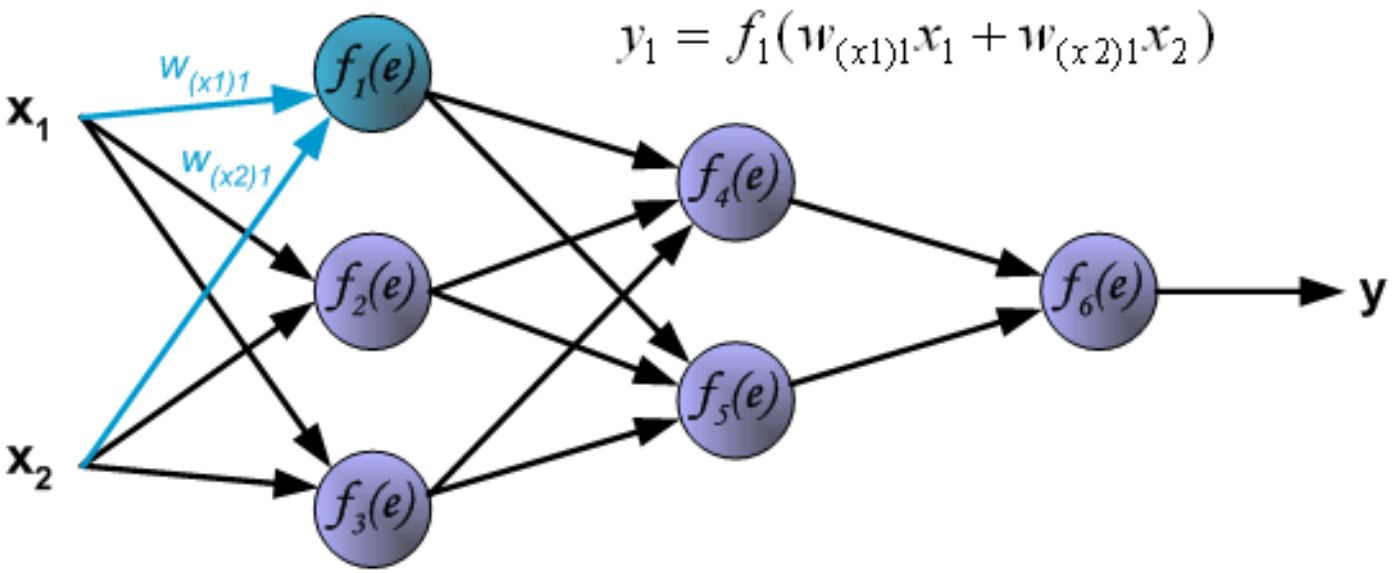
To teach the neural network we need training data set. The training data set consists of input signals (x_1 and x_2) assigned with corresponding target (desired output) z.

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:

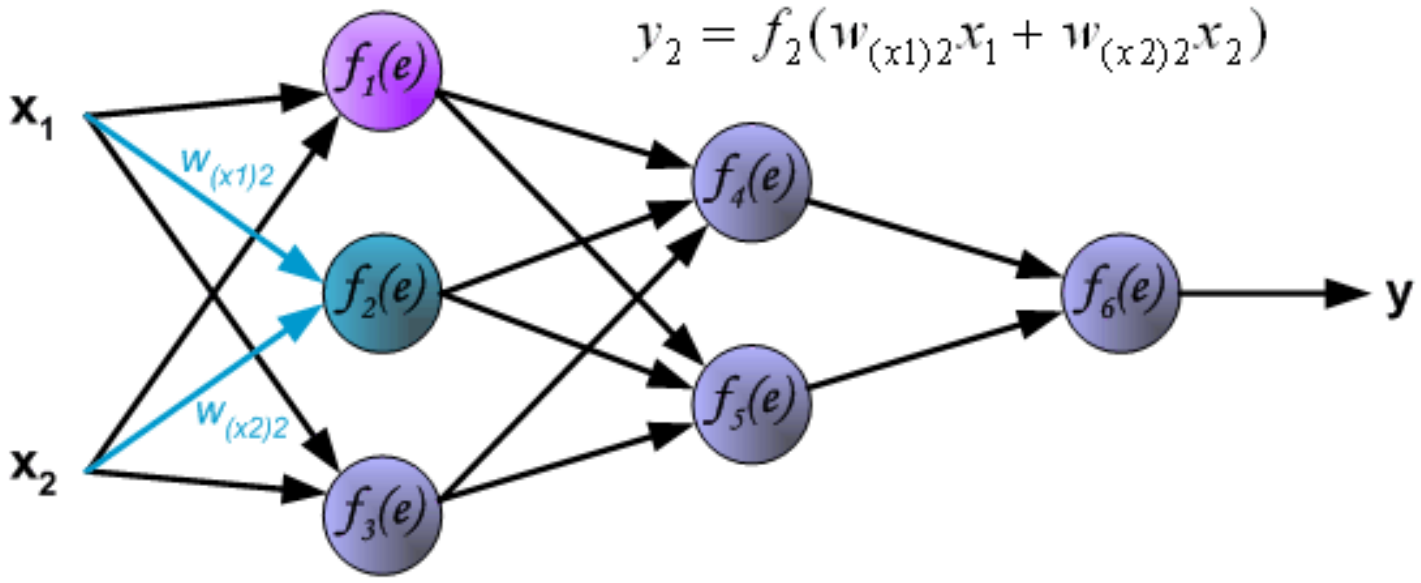
Each teaching step starts with forcing both input signals from training set. After this stage we can determine output signals values for each neuron in each network layer.

Learning Algorithm: Backpropagation

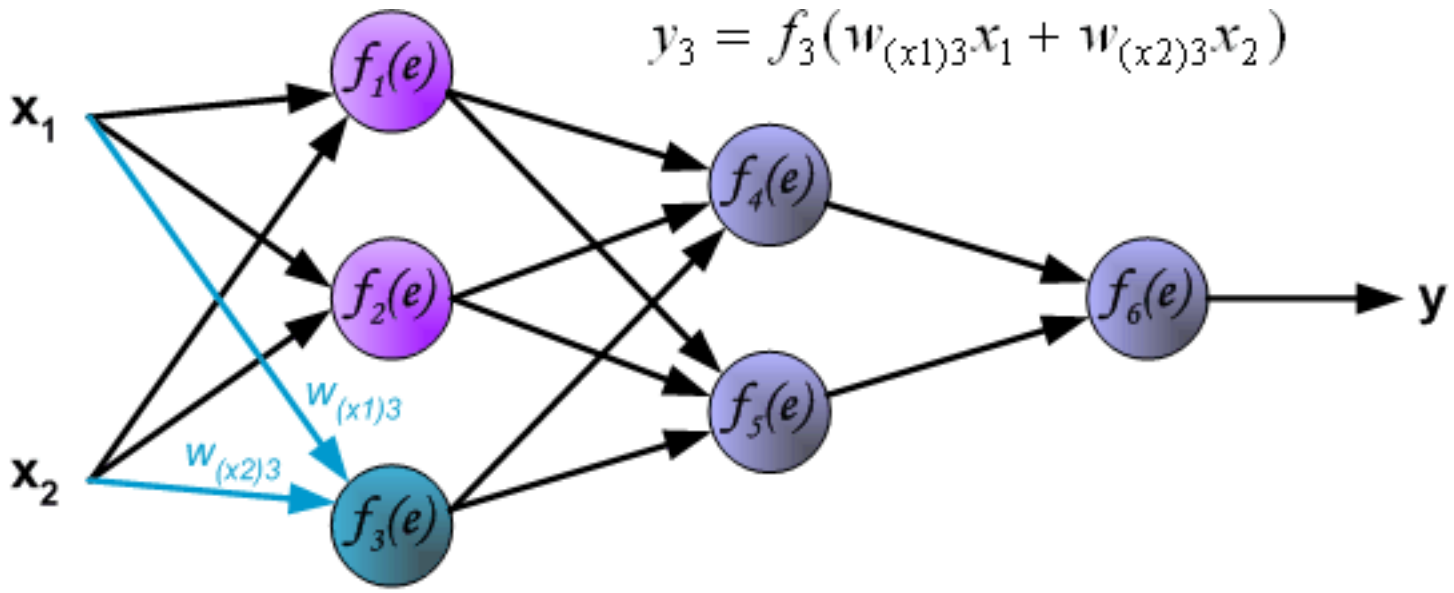
Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n.



Learning Algorithm: Backpropagation

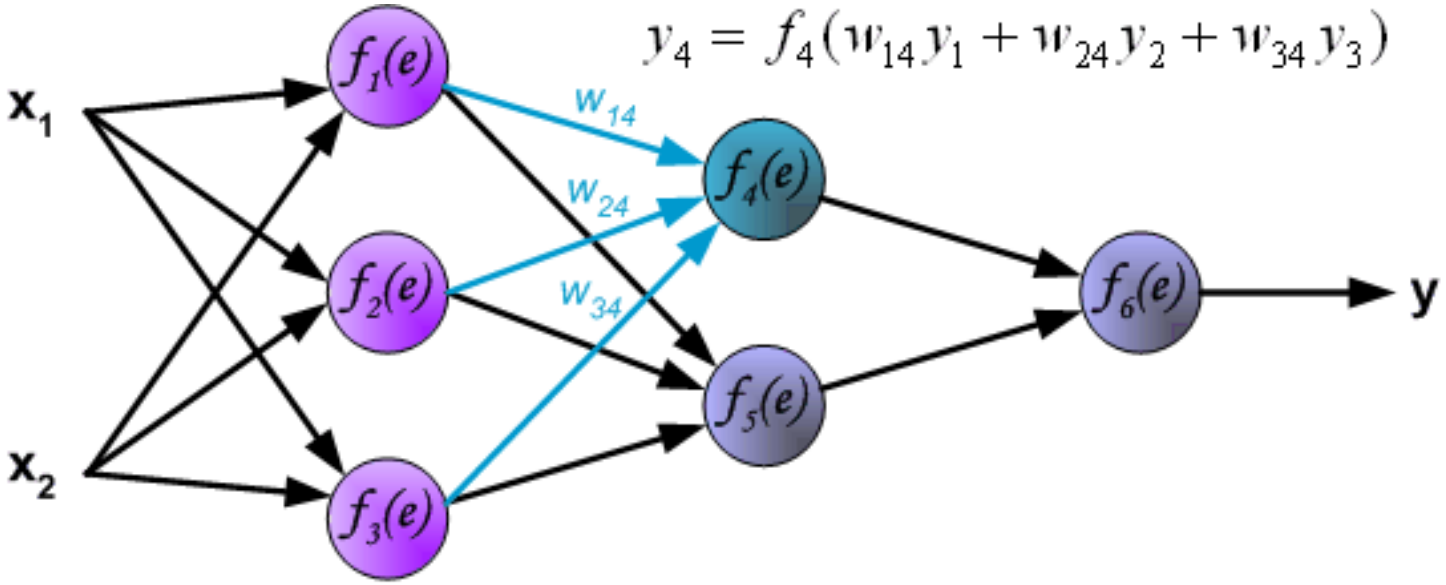


Learning Algorithm: Backpropagation

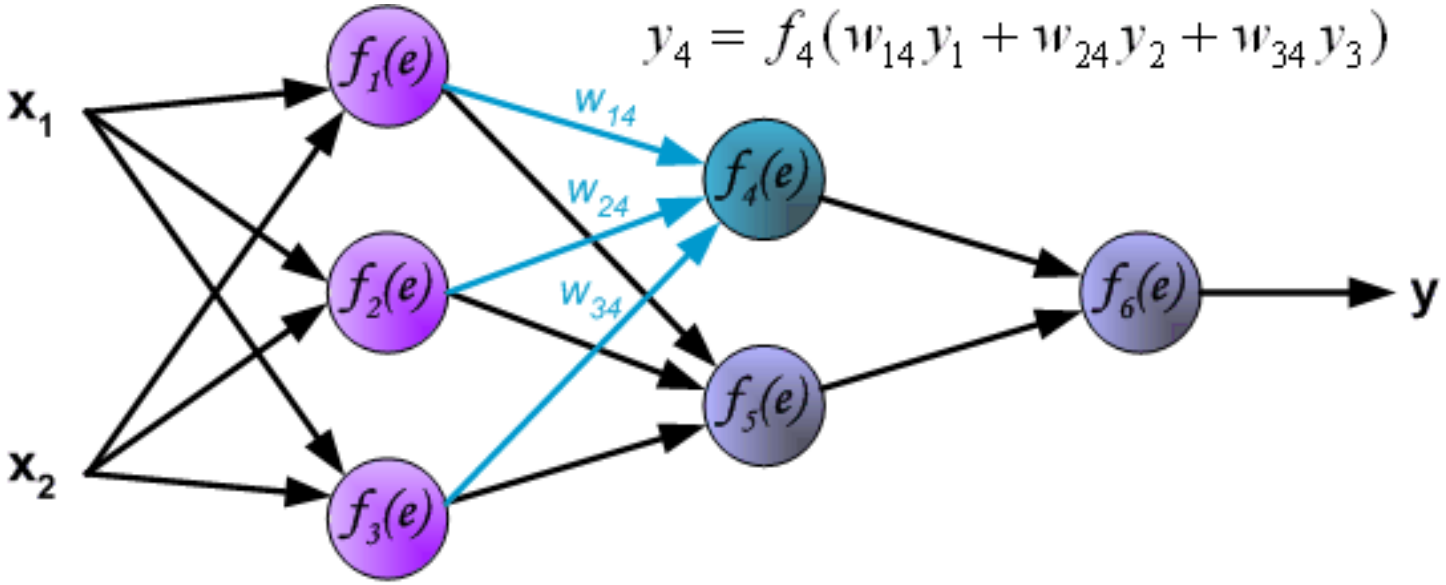


Learning Algorithm: Backpropagation

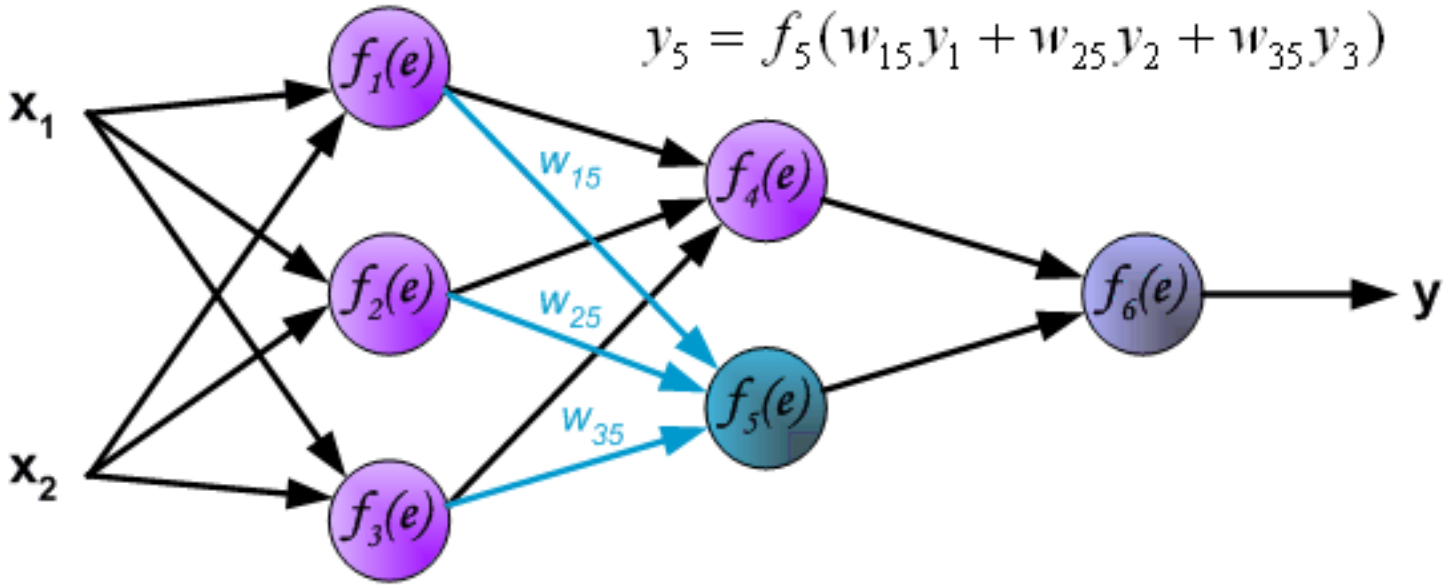
Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron m and input of neuron n in the next layer.



Learning Algorithm: Backpropagation

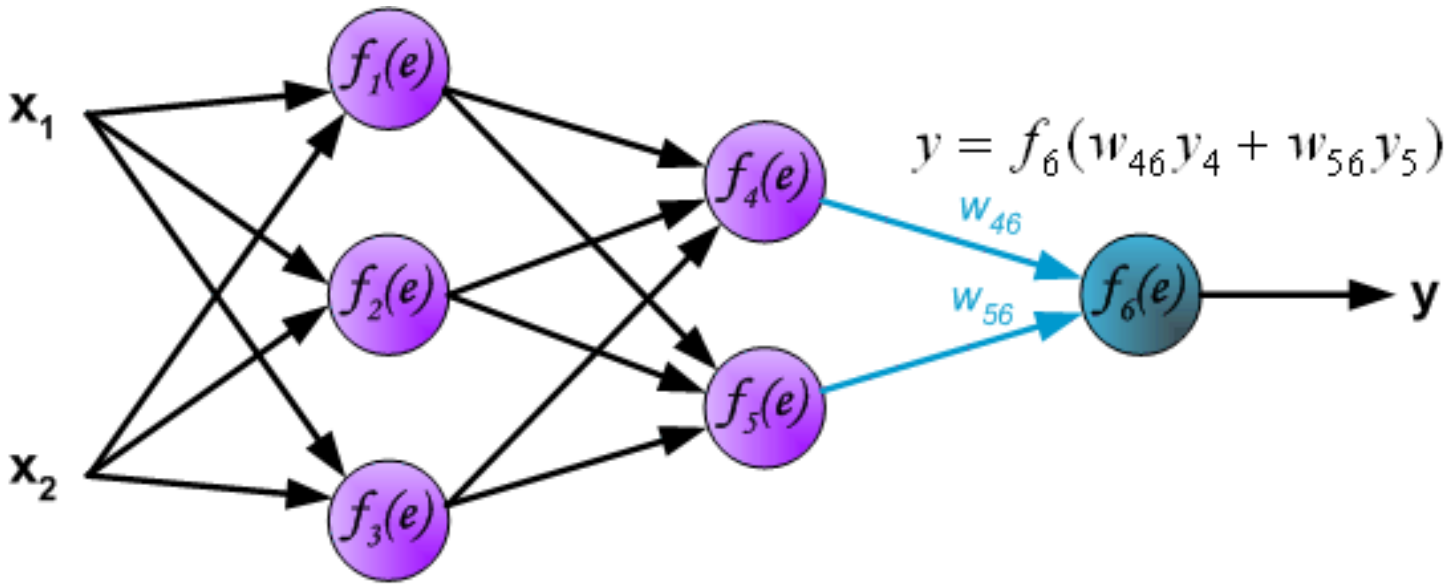


Learning Algorithm: Backpropagation



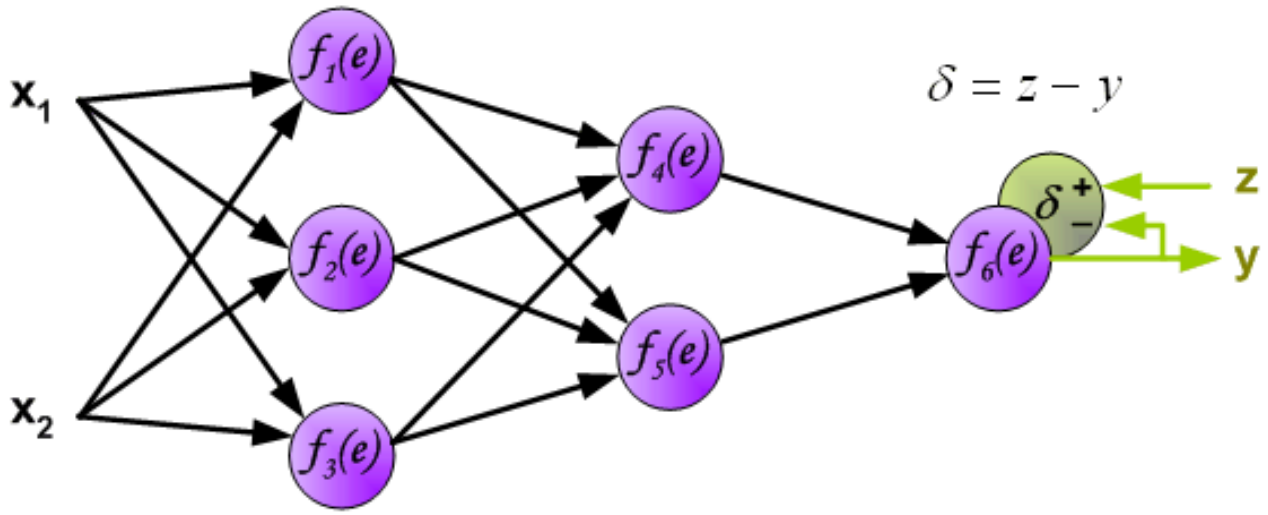
Learning Algorithm: Backpropagation

Propagation of signals through the output layer.



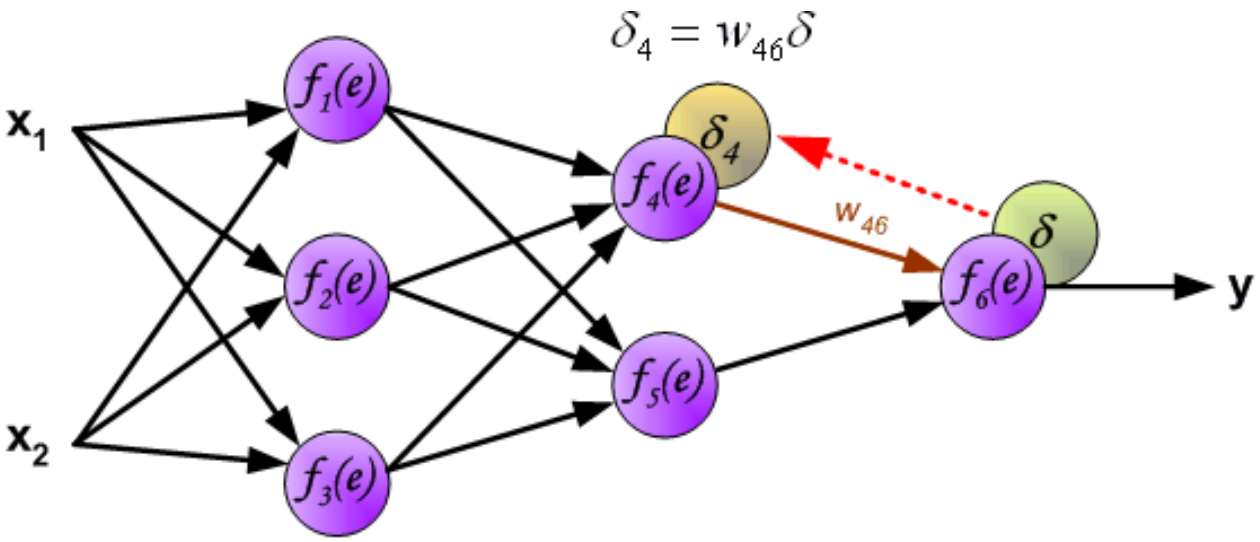
Learning Algorithm: Backpropagation

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



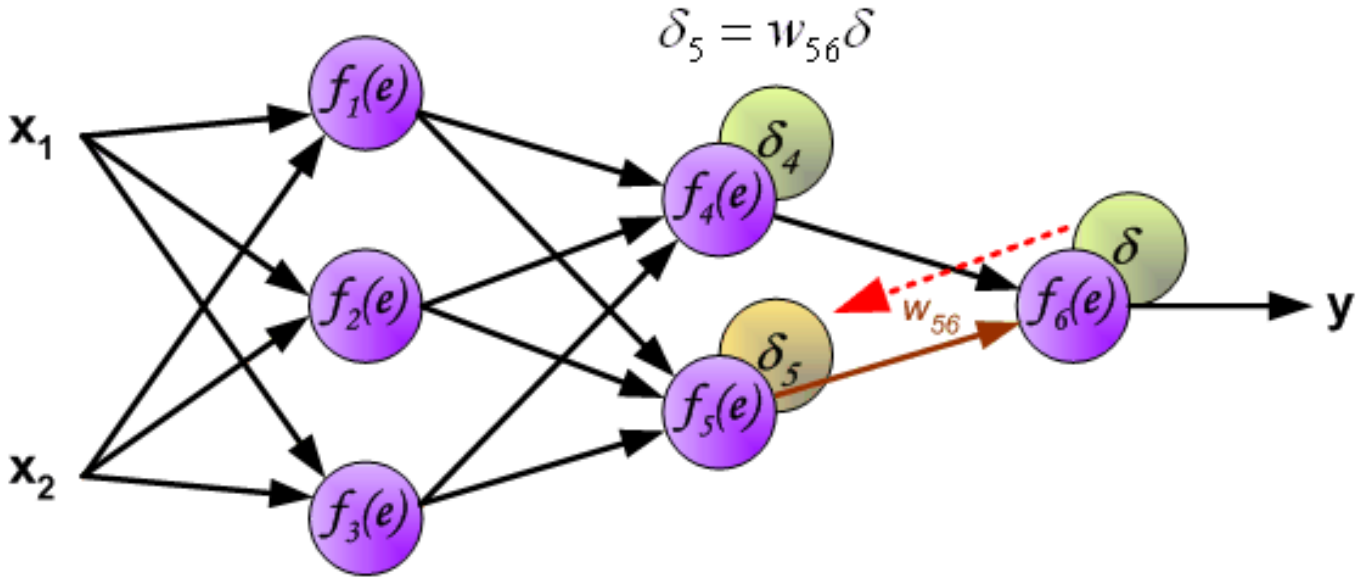
Learning Algorithm: Backpropagation

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.



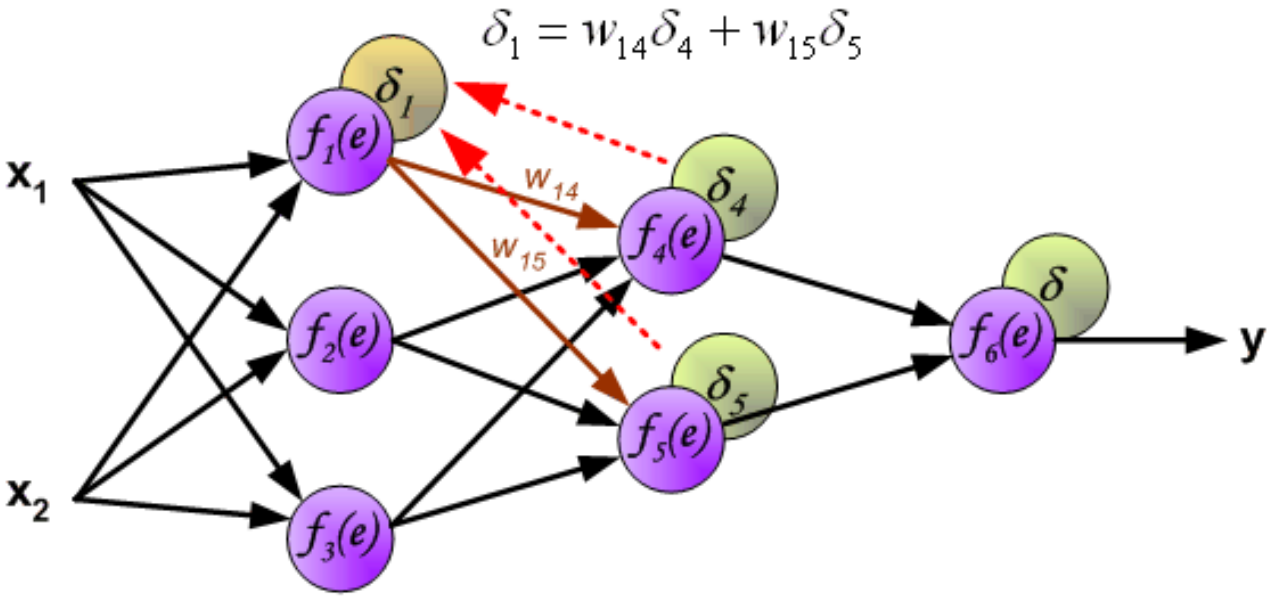
Learning Algorithm: Backpropagation

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:

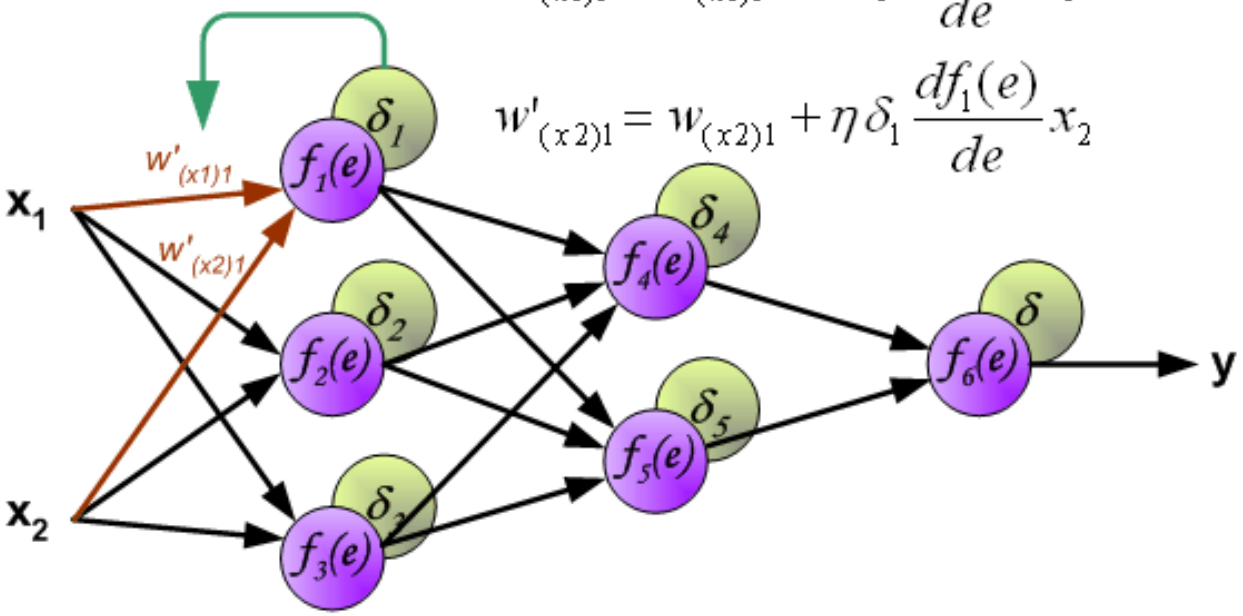


Learning Algorithm: Backpropagation

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).

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

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

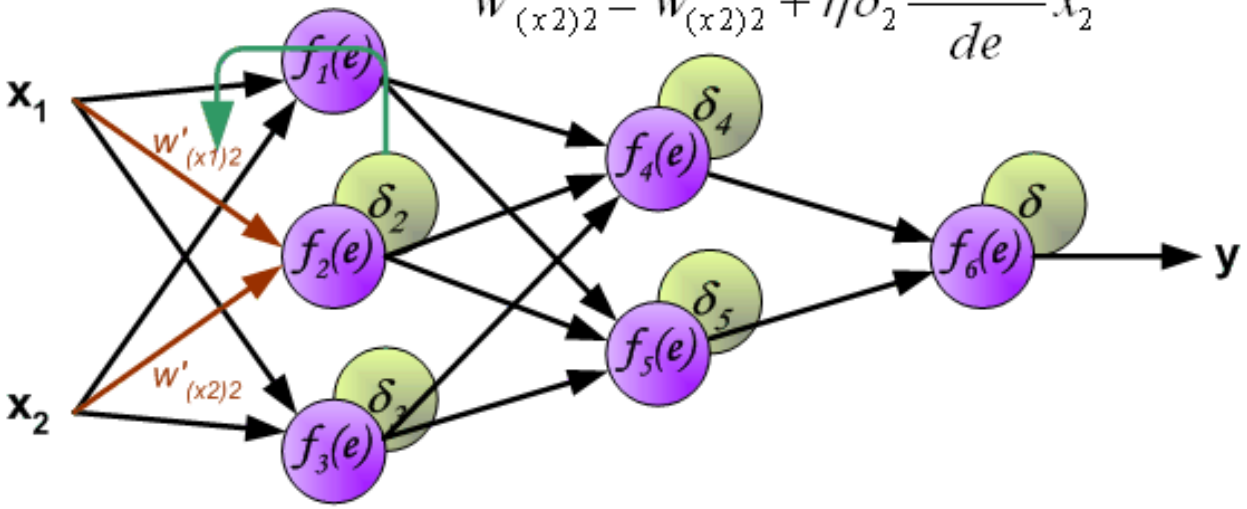


Learning Algorithm: Backpropagation

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).

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

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

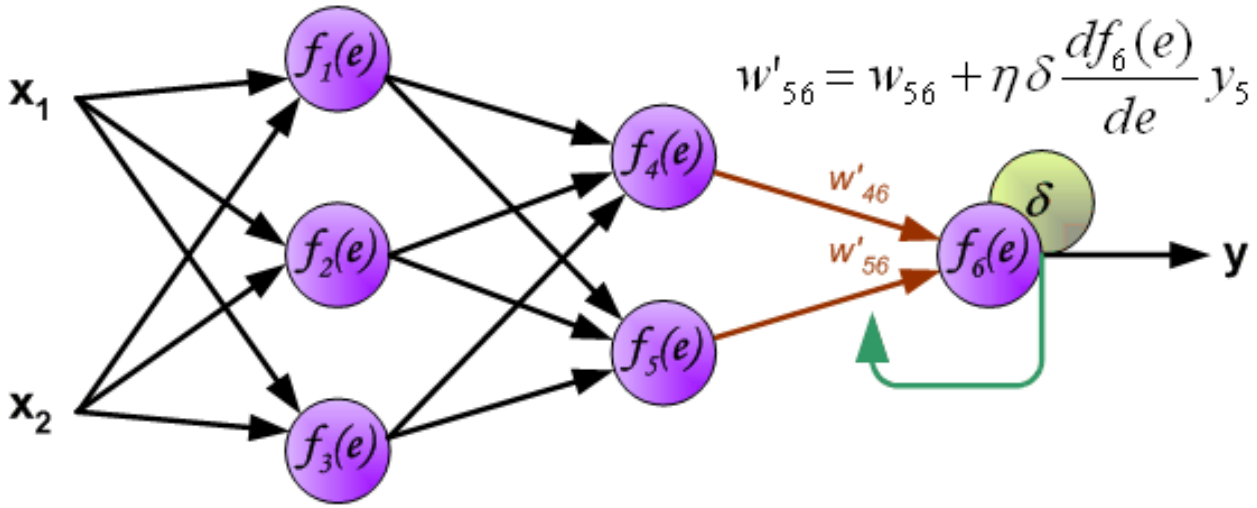


Learning Algorithm: Backpropagation

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).

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

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



Back-propagation problems

1. Design problem

- There are no formal method for choosing the proper design for the network.
- We mean by the design problem is specifying the number of hidden layers and the number of neurons in each layer.
- There are many method for doing this , one of them are is trial and error .

2. Convergence

- The main problem of the back-propagation is reaching the convergence.
- The correct value of the learning rate have a very high influence in convergence.

3. Generalization

- We mean by generalization is the network capability for recognizing new pattern that are not used in the training process.
- The “Overfitting” problem is the problem of increasing number of the network weight that compared with the number of the training pattern and that make the network memorize the training pattern. That will increase the learning performance and decrease the generalization performance.
- Many algorithms that are suggested in order to reduce the number of weight in the network.

4. Premature Saturation

- If the value of the initial weight is very high then the weight is growing very fast and the gradient is near zero and therefore there are no update in the weight and the error value is still high then the neuron is saturated.
- To solve that problem we can use some algorithms ,like genetic algorithm, to suggest the initial weights.

Thank you…

Any questions??

يرجى مسح رمز الاستجابة السريعة QR Code لتعبئة نموذج التغذية الراجعة حول المحاضرة. ملاحظاتكم مهمة لتحسين المحاضرات القادمة.

My google site