In The Name of Allah

Digital Media Laboratory
Sharif University of Technology

Statistical Pattern Recognition

Neural Networks & Backpropagation

Hamid R. Rabiee
Jafar Muhammadi

Spring 2012
http://ce.sharif.edu/courses/90-91/2/ce725-1/

Agenda

- Nature Inspired
- Network Structures
- Feed Forward Networks
  - Units (Neurons)
  - Activation Functions
  - Learning
- Perceptron Learning Algorithm
- Neural Networks and Separability
- Backpropagation Algorithm
- Conclusion and Problems

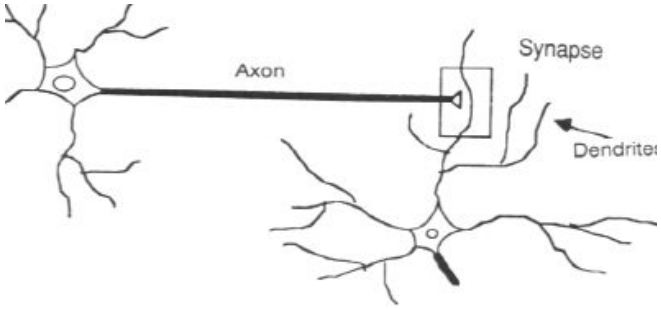
Nature Inspired

Brain

Interconnected network of **neurons** that collect, process and disseminate electrical signals via **synapses**

Neurons

Synapses

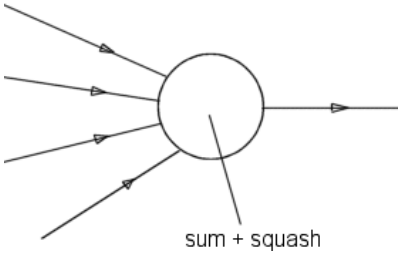


Neural Network

Interconnected network of **units** (or **nodes**) that collect, process and disseminate values via **links**

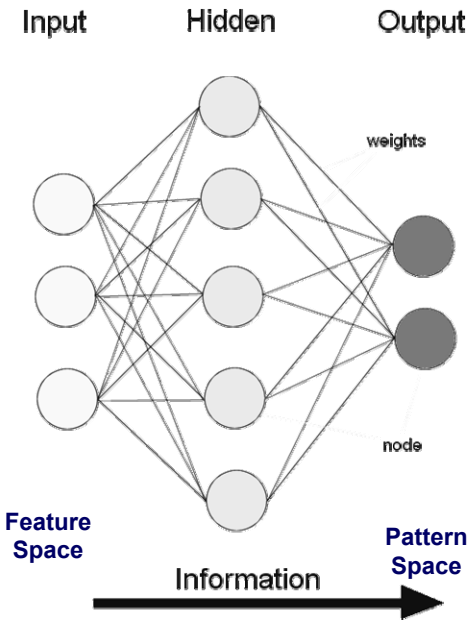
Nodes

Links



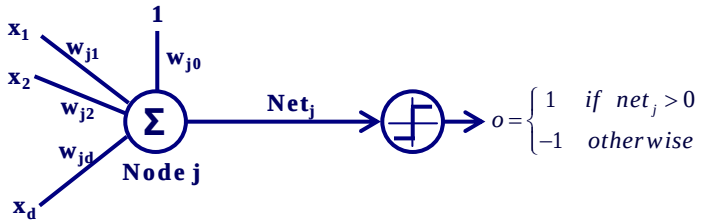
Feed Forward Networks

- Units (nodes) usually arranged in layers
  - Each unit receives input only from units in the preceding layer
  - Represent a function of its current inputs
- No internal state other than the weights on links
- Two types of feed-forward networks:
  - Single layer
  - Multilayer
- Most applications require at least three layers:
  - Input layer (e.g. read from files or electronic sensors)
  - Hidden layer(s)
  - Output layer (e.g. sends to another process or device)
- Large hidden layer – represent any continuous function of the inputs



Units (Neurons)

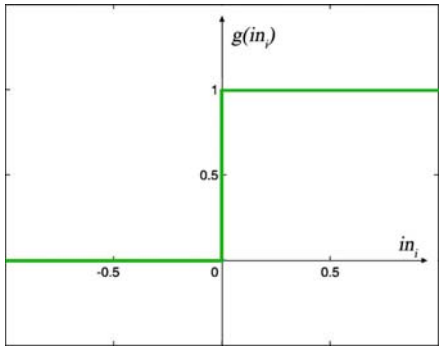
- **Each unit (Neuron) has some inputs and one output**
  - **A single "bias unit" is connected to each unit other than the input units**
  - **Net activation:** $net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv w_j^t . x$

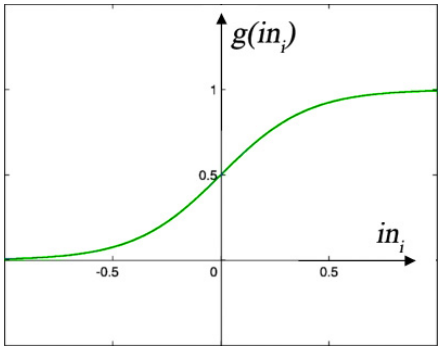


- **Each hidden unit emits an output that is a nonlinear function of its activation**
  - $y_j = f(net_j)$
    - *f named "Activation function"*
- **A single unit is equivalent to a linear classifier (why?)**

Activation Functions

- **A variety of activation functions used**
- **Two common ones:**
  - **Threshold function**
    - Θ(netinput) = {1 ,if netinput ≥ 0 ; 0 , otherwise}
    - Useful for classifying inputs into two groups.
    - Used to build networks that function as:
      - **Feature identifiers**
      - **Boolean input computers**
  - **Sigmoid function**
    - g(netinput) = $1/(1+ \exp^{-(\text{netinput})})$
    - Also known as logistic function
    - Main advantage is that it has a nice derivative (why?)
    - Helpful for learning



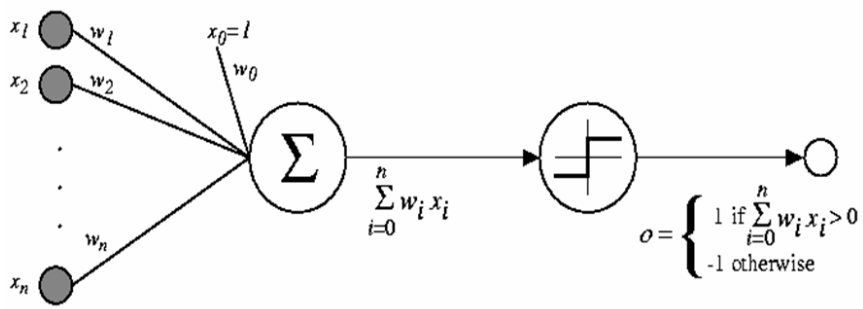


Learning

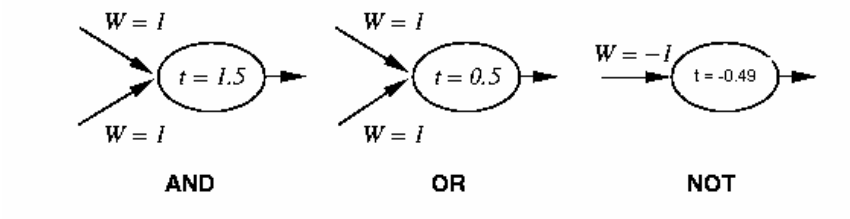
- **Weight settings determine the behavior of a network**
  - **Each connection in the NN diagram has a weight (Weights are adjustable)**
  - **The function of these weights is to reduce error between desired output and actual output**
  - **How can we find the right weights?**
- **Network learning**
  - **Requires training set (input / output pairs)**
  - **Perceptron**
    - A single neuron learning algorithm
  - **Backpropagation**
    - A multilayered feed forward network learning algorithm
    - It has been shown that a feed forward network trained using backpropagation with sufficient number of units can approximate any continuous function to any level of accuracy
    - Starts with small random weights
    - Error is used to adjust weights (supervised learning) - Gradient descent on error landscape

Perceptron

- **Single neuron with inputs, bias and output**



- **Example operations with single neuron**

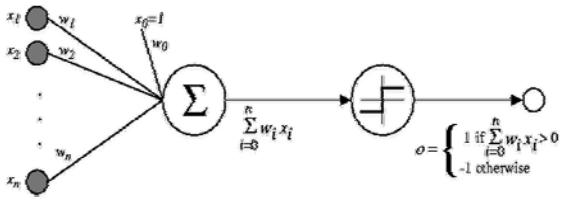


  - t can be considered as bias (Why?).

- **Does a single neuron can solve the XOR problem?**

Perceptron Learning Algorithm

- Perceptron Learning Rule: $w_i = w_i + \eta \Delta w_i$
  - $\Delta w_i = (t - o)\, x_i$
    - **t = c(x) is the target value**
    - **o is the perceptron output**
    - **η Is a small constant (e.g. 0.1) called *learning rate***
  - If the output is correct (t=o) the weights w_i are not changed
  - If the output is incorrect ($t \neq o$) the weights w_i are changed
    - **such that the output of the perceptron for the new weights is *closer* to t.**
  - The algorithm converges to the correct classification if,
    - **the training data is linearly separable**
    - **and η is sufficiently small**
- Gradient Descent Rule: $\mathbf{w = w - \eta \nabla E_D[w]}$
  - $E_D[w] = 1/2 \Sigma_d (t_d - o_d)^2$



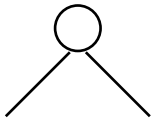
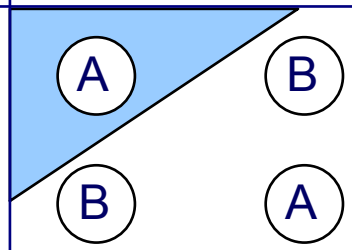
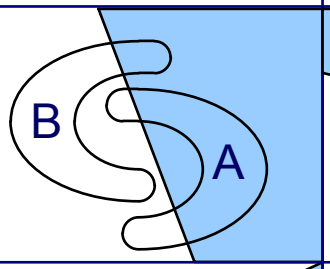
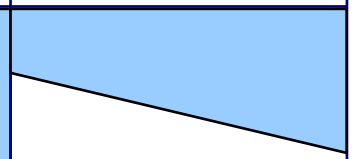
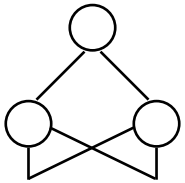
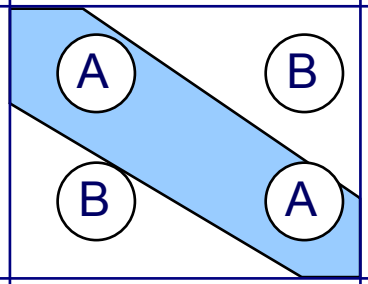
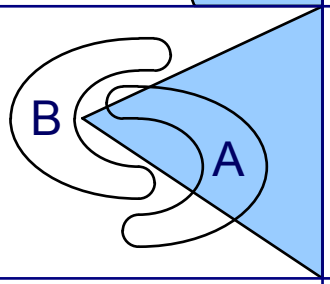
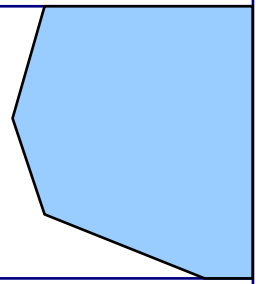
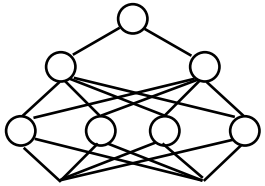
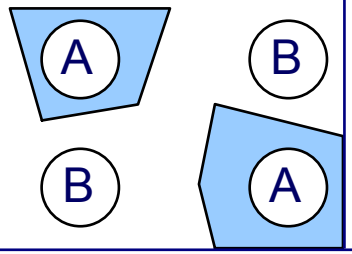
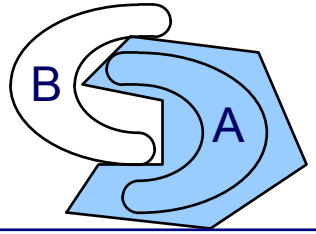
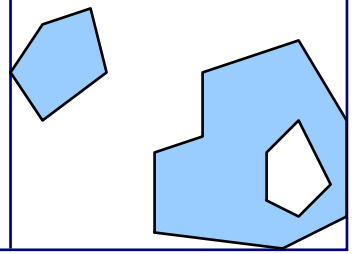
Perceptron vs. Gradient Descent Rule

- **Perceptron learning rule guaranteed to succeed if**
  - Training examples are linearly separable
  - Sufficiently small learning rate η
- **Linear unit training rules uses gradient descent**
  - Guaranteed to converge to hypothesis with minimum squared error (Why?)
  - Succeed if sufficiently small learning rate η
  - Even when training data contains noise
  - Even when training data not separable by Hyperplane (H)

Learning Modes

- **Batch mode**
  - Rule updating over the entire data D
- **Incremental mode**
  - Rule updating over individual training examples d
- **Incremental Gradient Descent can approximate Batch Gradient Descent arbitrarily closely if η is small enough**

Neural Network and Separability

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

Backpropagation Algorithm

- **Any function from input to output can be implemented as a three-layer neural network**
- **Our goal now is to set the interconnection weights based on the training patterns and the desired outputs.**
- **In a three-layer network, it is a straightforward matter to understand how the output, and thus the error, depend on the hidden-to-output layer weights**
- **The power of backpropagation is that it enables us to compute an effective error for each hidden unit, and thus derive a learning rule for the input-to-hidden weights, this is known as "The credit assignment problem".**

Backpropagation Algorithm

Network Learning

- Let t_k be the k-th target (or desired) output and z_k be the k-th computed output with $k = 1, \ldots, c$ and w represents all the weights of the network
- The training error is

$$J(w) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t - z\|^2$$

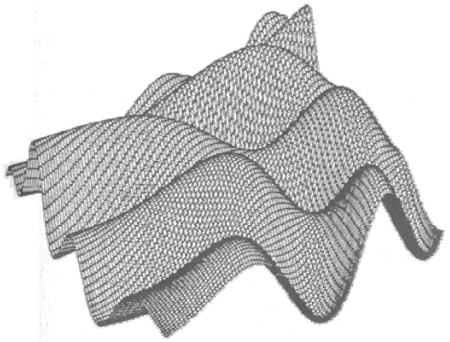
- The backpropagation learning rule is based on gradient descent
  - The weights are initialized with pseudo-random values and are changed in a direction that will reduce the error:

$$\Delta w = -\eta \frac{\partial J}{\partial w}$$

    - where η is the learning rate which indicates the relative size of the change in weights

$$w(m+1) = w(m) + \Delta w(m)$$

    where m is the m-th pattern presented



Backpropagation Algorithm

- **Error on the hidden–to-output weights**

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} = -\delta_k \frac{\partial net_k}{\partial w_{kj}}$$

where the sensitivity of unit k is defined as:

$$\delta_k = -\frac{\partial J}{\partial net_k}$$

and describes how the overall error changes with the activation of the unit's net

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

Since $net_k = w_k^t.y$ therefore:

$$\frac{\partial net_k}{\partial w_{kj}} = y_j$$

- **Conclusion: the weight update (or learning rule) for the hidden-to-output weights is:**

$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j$$

- **Error on the input-to-hidden units**

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

Backpropagation Algorithm

- **However,**

$$\frac{\partial J}{\partial y_j}=\frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k-z_k)^2\right]=-\sum_{k=1}^{c}(t_k-z_k)\frac{\partial z_k}{\partial y_j}$$

$$=-\sum_{k=1}^{c}(t_k-z_k)\frac{\partial z_k}{\partial net_k}.\frac{\partial net_k}{\partial y_j}=-\sum_{k=1}^{c}(t_k-z_k)\,f'(net_k)w_{kj}$$

- **Similarly as in the preceding case, we define the sensitivity for a hidden unit:**

$$\delta_j \equiv f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$$

- **which means that: "The sensitivity at a hidden unit is simply the sum of the individual sensitivities at the output units weighted by the hidden-to-output weights w_{kj}; all multiplied by $f'(net_j)$"**

- **Conclusion: The learning rule for the input-to-hidden weights is:**

$$\Delta w_{ji}=\eta x_i \delta_j=\eta\underbrace{\left[\Sigma w_{kj}\delta_k\right] f'(net_j)}_{\delta_j} x_i$$

- **Note that the activation function must be differentiable!**
  - Recall that sigmoid is differentiable and has a nice derivation!

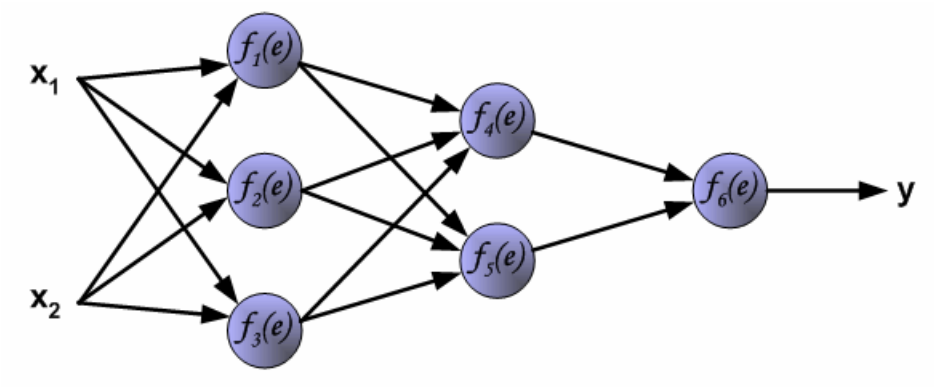
Backpropagation Algorithm

- **Starting with a pseudo-random weight configuration, the stochastic backpropagation algorithm can be written as:**

```
Begin  initialize    n_H; w, criterion θ, η, m ← 0
            do m ← m + 1
                 x^m ← randomly chosen pattern
                 w_ji ← w_ji + ηδ_j x_i; w_kj ← w_kj + ηδ_k y_j
            until ||∇J(w)|| < θ
       return w
End
```

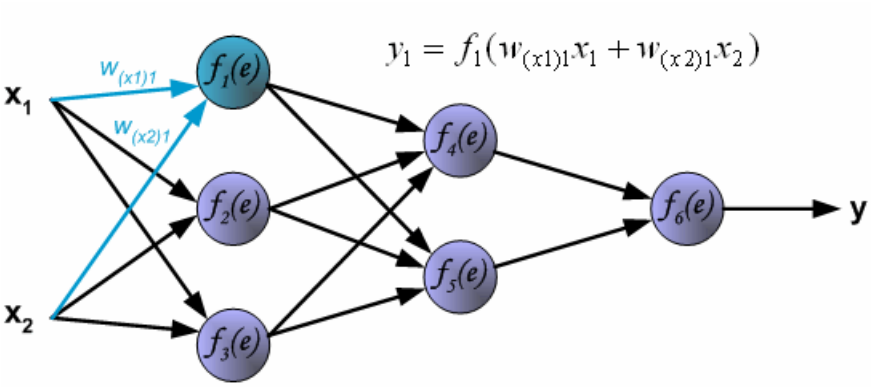

Backpropagation Illustration

- **3 layer / 2 inputs / 1 output:**



Backpropagation Illustration

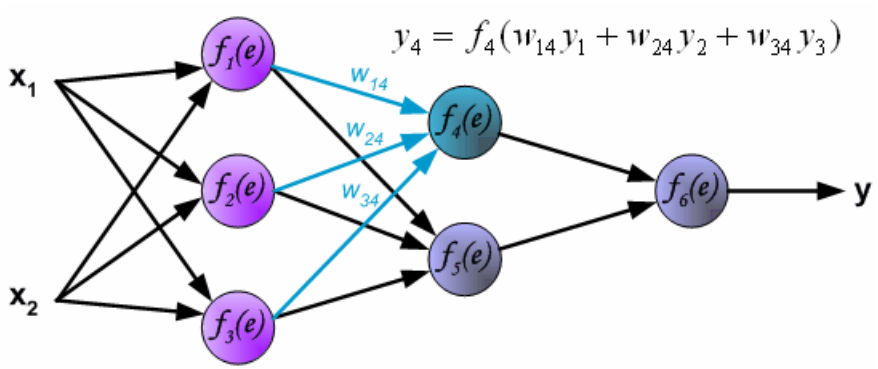
- **Training Starts through the input layer:**



- **The same happens for y2 and y3.**

Backpropagation Illustration

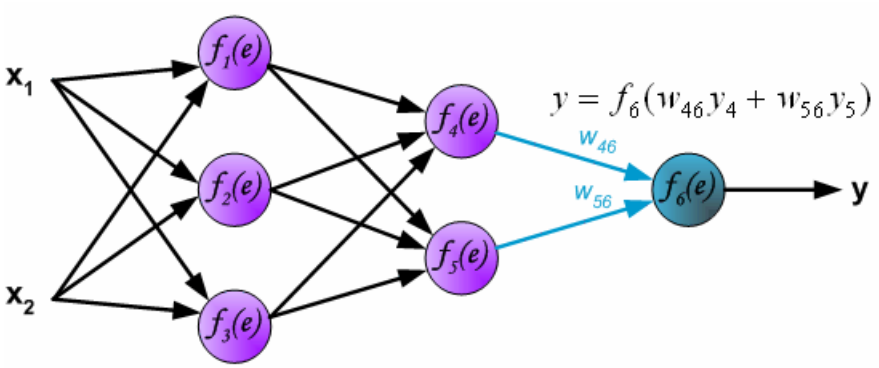
- **Propagation of signals through the hidden layer:**



- **The same happens for y5.**

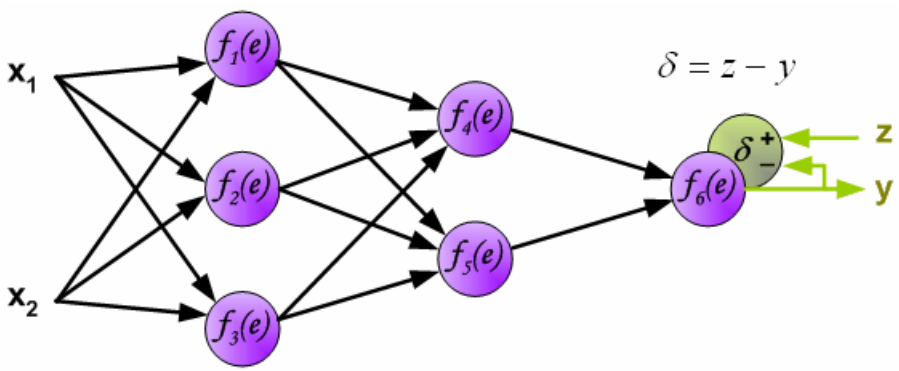
Backpropagation Illustration

- **Propagation of signals through the output layer:**



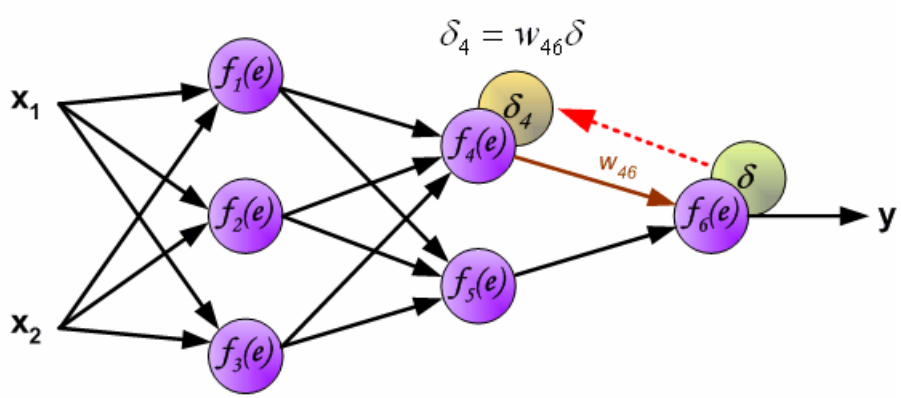
Backpropagation Illustration

- **Error signal of output layer neuron:**



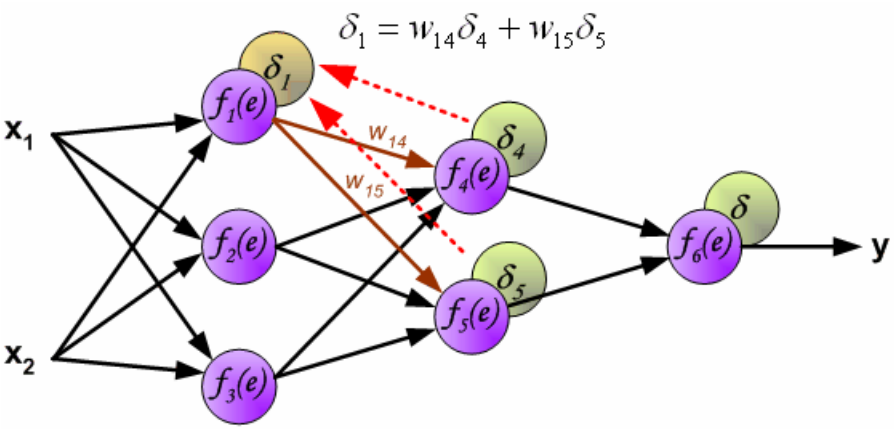
Backpropagation Illustration

- **propagate error signal back to all neurons.**



Backpropagation Illustration

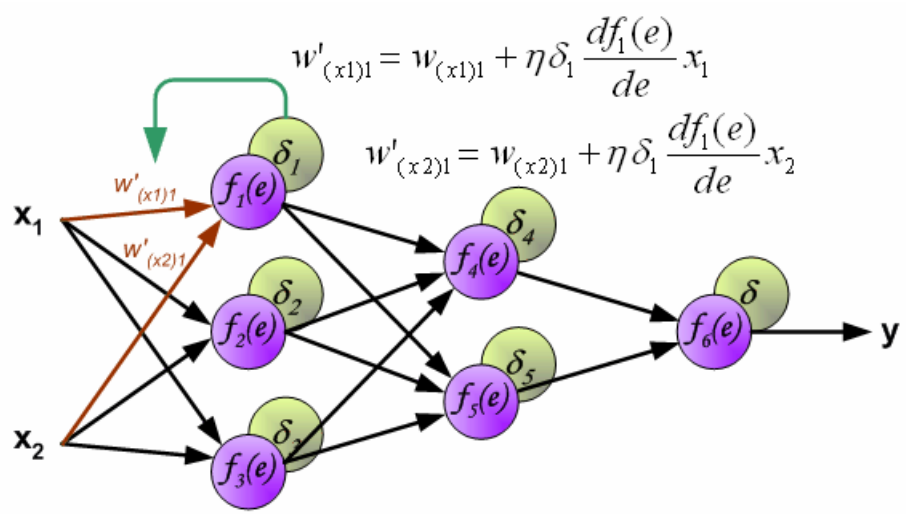
- **If propagated errors came from few neurons, they are added:**



- **The same happens for neuron-2 and neuron-3.**

Backpropagation Illustration

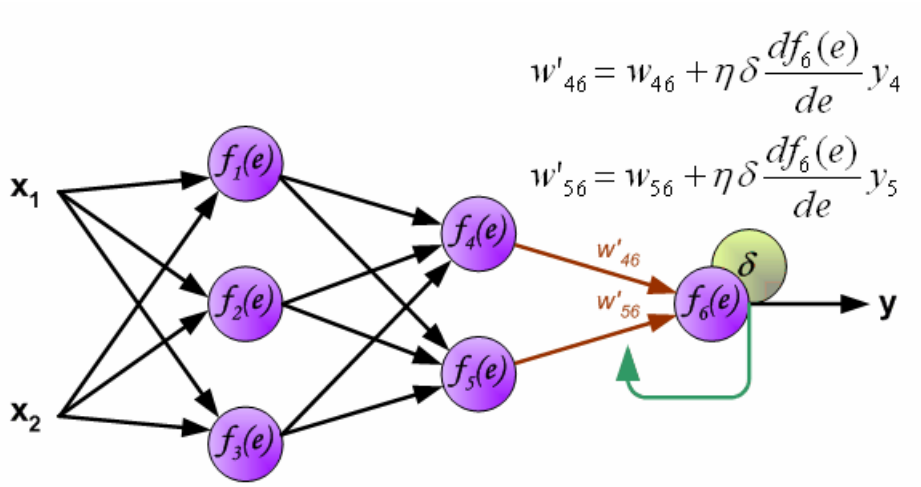
- **Weight updating starts:**



- **The same happens for all neurons.**

Backpropagation Illustration

- **Weight updating terminates in the last neuron:**



Backpropagation Algorithm

- **Stopping criterion**
  - The algorithm terminates when the change in the criterion function J(w) is smaller than some preset value θ
  - There are other stopping criteria that lead to better performance than this one
  - So far, we have considered the error on a single pattern, but we want to consider an error defined over the entirety of patterns in the training set
  - The total training error is the sum over the errors of n individual patterns

$$J = \sum_{p=1}^{n} J_p$$

  - A weight update may reduce the error on the single pattern being presented but can increase the error on the full training set
  - However, given a large number of such individual updates, the total error of above equation decreases

Backpropagation Algorithm

- **Another Stopping Criteria**
  - Stop if the error fails to improve (has reached a minimum)
  - Stop if the rate of improvement drops below a certain level
  - Stop if the error reaches an acceptable level
  - Stop when a certain number of epochs have passed

Backpropagation Algorithm

- **Learning Curves**
  - **Before training starts, the error on the training set is high; through the learning process, the error becomes smaller**
  - **The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network**
  - **A validation set is used in order to decide when to stop training ; we do not want to overfit the network and decrease the power of the classifier generalization**

    "we stop training at a minimum of the error on the validation set"

Neural Networks conclusion

- **When to use neural networks**
  - **Use for huge data sets (i.e. 50 outputs and 15,000 inputs) with unknown distributions**
  - **Smaller data sets with outliers**
    - neural networks are very resistant to outliers
  - **Neural networks should not be used when traditional methods are appropriate**
- **Drawbacks and Limitations**
  - **The results can be very hard to interpret**
  - **Will find a local, not necessarily global error minimum**
    - in practice often works well (can be invoked multiple times with different initial weights)
  - **Many Parameters to be set**
    - Number of layers, Number of neurons, Learning rate, …
    - Overfitting problem
  - **Training can be slow typical 1000-10000 iterations**
    - Using network after training is fast
  - **Since they are data dependent performance will improve as sample size increases**

Other Types of Neural Networks

- **Feed forward networks**
  - Simplest approach. Mostly used for pattern classification
- **Radial Basis Functions**
  - A replacement for multi-layer perceptrns. Mostly used for function approximation and interpolation.
- **Kohonen self-organizing networks**
  - Perform unsupervised learning. Also used for dimensional reduction.
- **Recurrent Neural Networks**
  - Bi-directional data flow. Can perform pattern association among other things
- **Time Delay Neural Networks**
  - For time series analysis
- **Restricted Boltzmann Machines**
  - A generative instead of discriminative approach

Any Question

End of Lecture 9

Thank you!

Spring 2012

http://ce.sharif.edu/courses/90-91/2/ce725-1/