Artificial Neural Networks

BACKPROPAGATION LEARNING ALGORITHMS

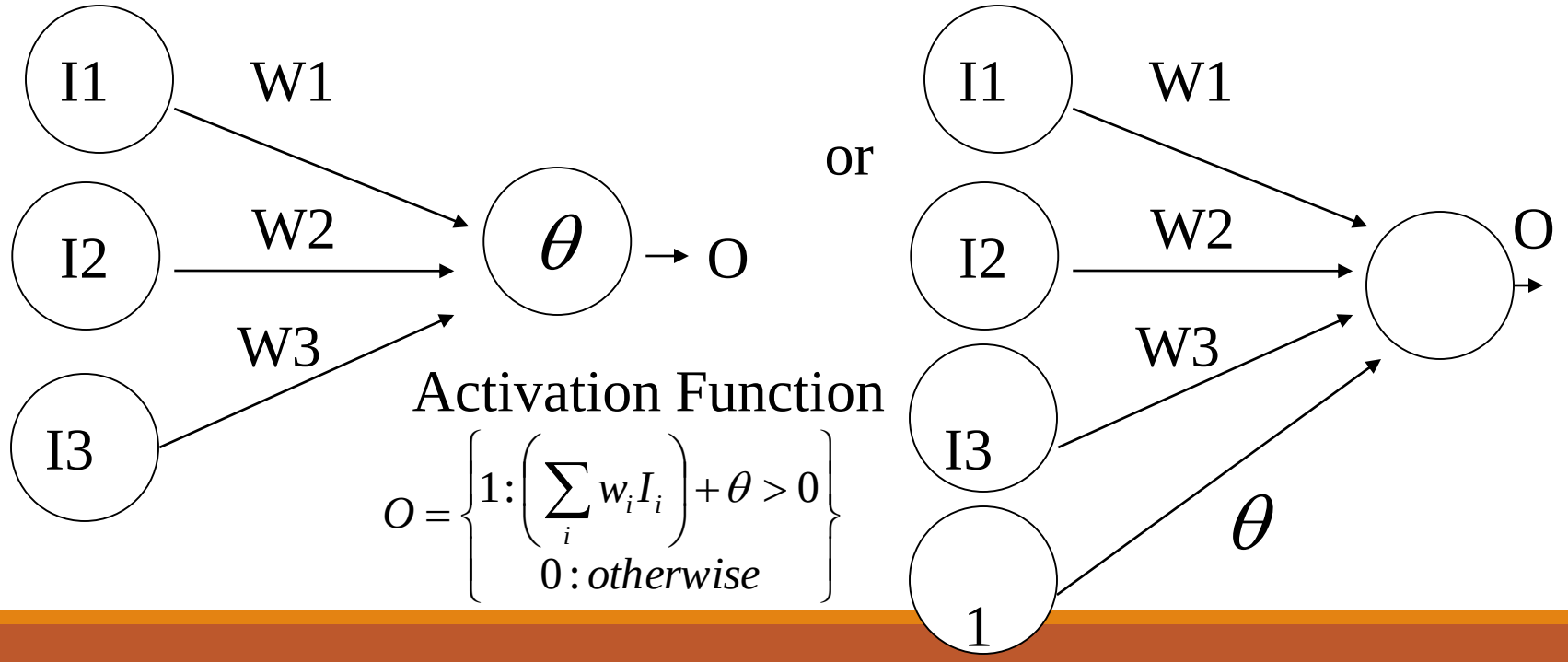
Topics

1. Perceptron
   a) Single layer vis-à-vis multi-layer networks
   b) Function estimation
   c) Perceptron Learning
   d) Backpropagation
   e) Batch and incremental training
   f) Examples

Perceptron

Initial proposal of connectionist networks Rosenblatt, 50's and 60's

Essentially a linear discriminant composed of nodes, weights



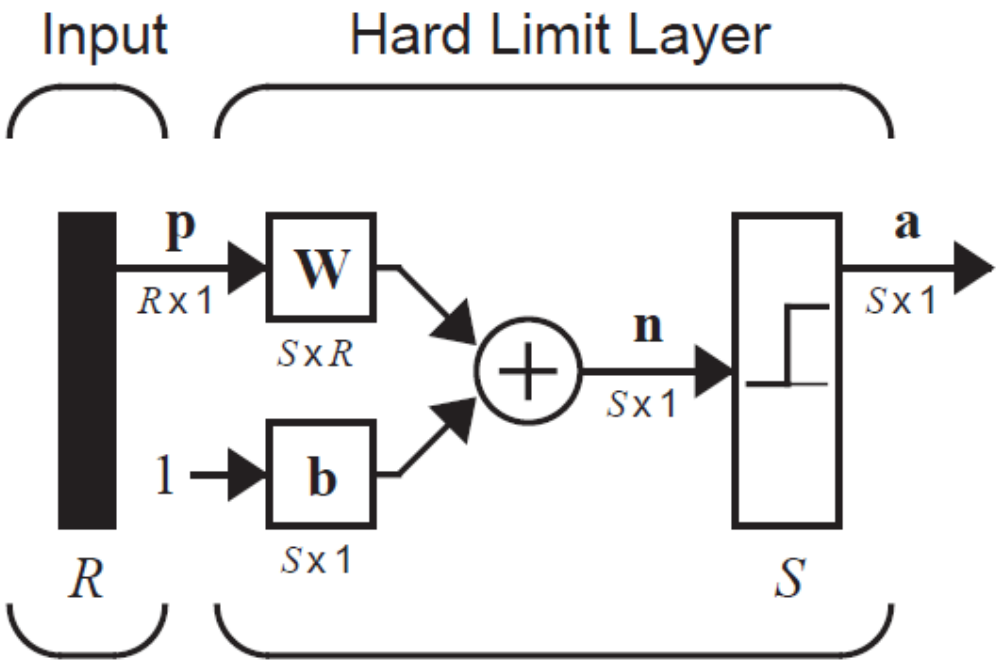
Perceptron

The activation function is a threshold (hard limit) function.

A perceptron a linear discriminant function dividing the input space into two areas.

The boundary is given by $W^T P + b = 0$

Perceptron



$$\mathbf{a} = \mathbf{hardlim}(\mathbf{Wp}+\mathbf{b})$$

Multiple Neuron Perceptron

Each neuron has its own decision boundary

$$W^T P + b = 0$$

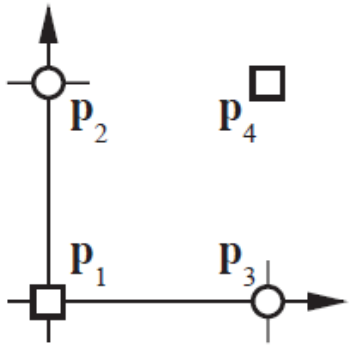
This results in 2^S different sub-space

Limitations of Single Perceptron Networks

A single perceptron divides the input (features) space into two parts. Hence, only linearly separated data sets can be classified using a single perceptron.

Multiple perceptron divide the space into multiple areas but still cannot perform non-linear classification

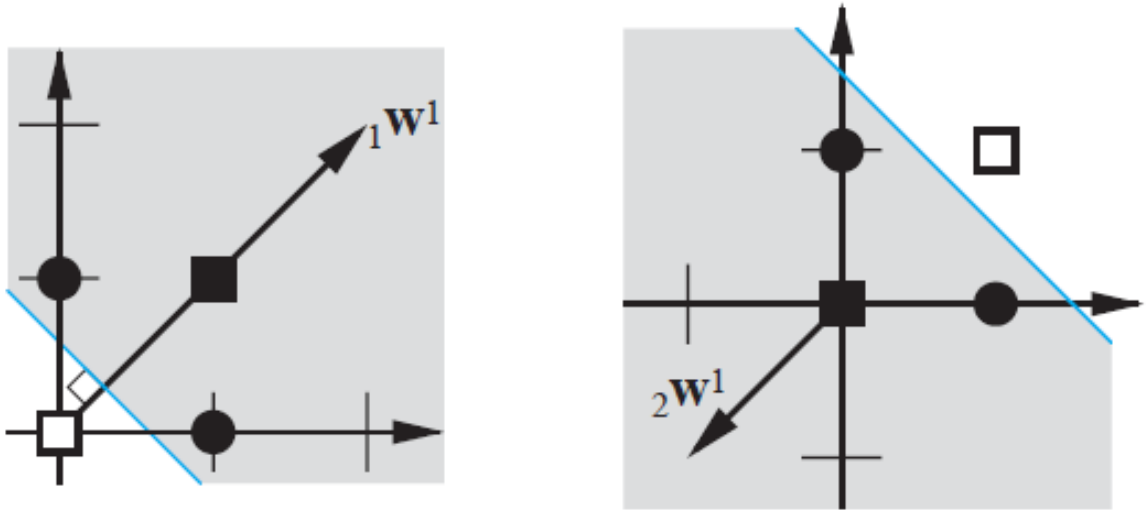
XOR Example



$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\0\end{bmatrix}, t_1 = 0\right\}\left\{\mathbf{p}_2 = \begin{bmatrix}0\\1\end{bmatrix}, t_2 = 1\right\}\left\{\mathbf{p}_3 = \begin{bmatrix}1\\0\end{bmatrix}, t_3 = 1\right\}\left\{\mathbf{p}_4 = \begin{bmatrix}1\\1\end{bmatrix}, t_4 = 0\right\}$$

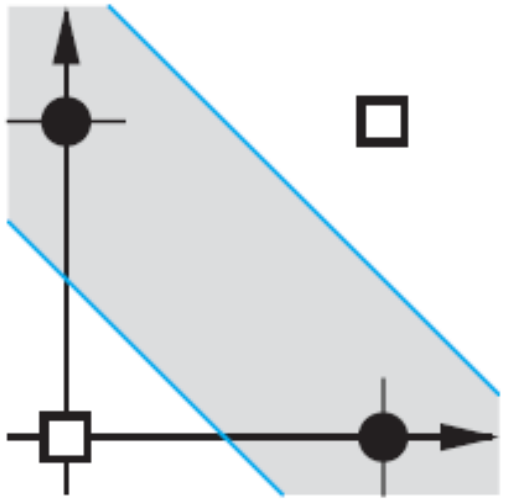
XOR Example

Two perceptron neurons can divide the space as shown

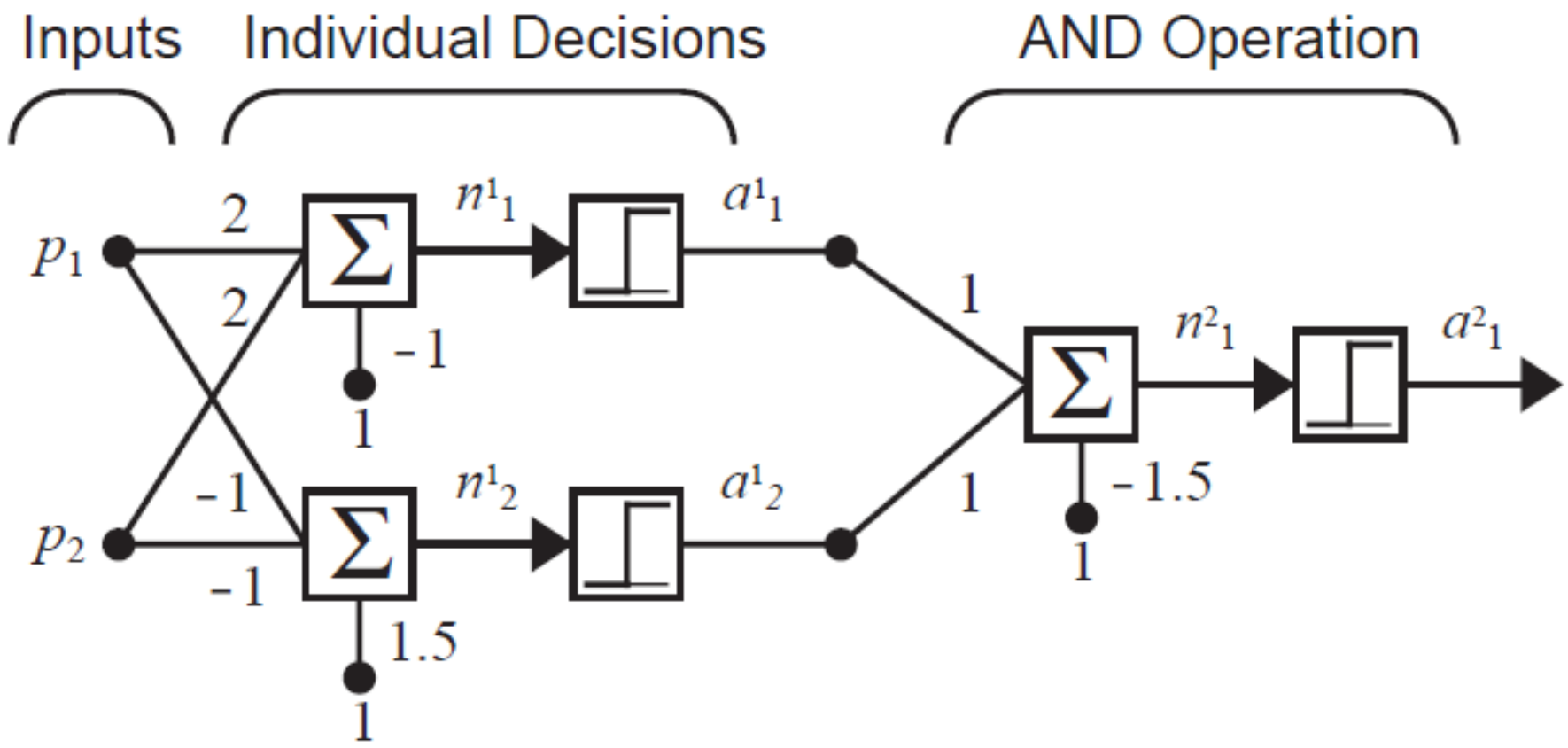


XOR Example

Joining the outputs of these neurons we can classify XOR pattern



XOR Example

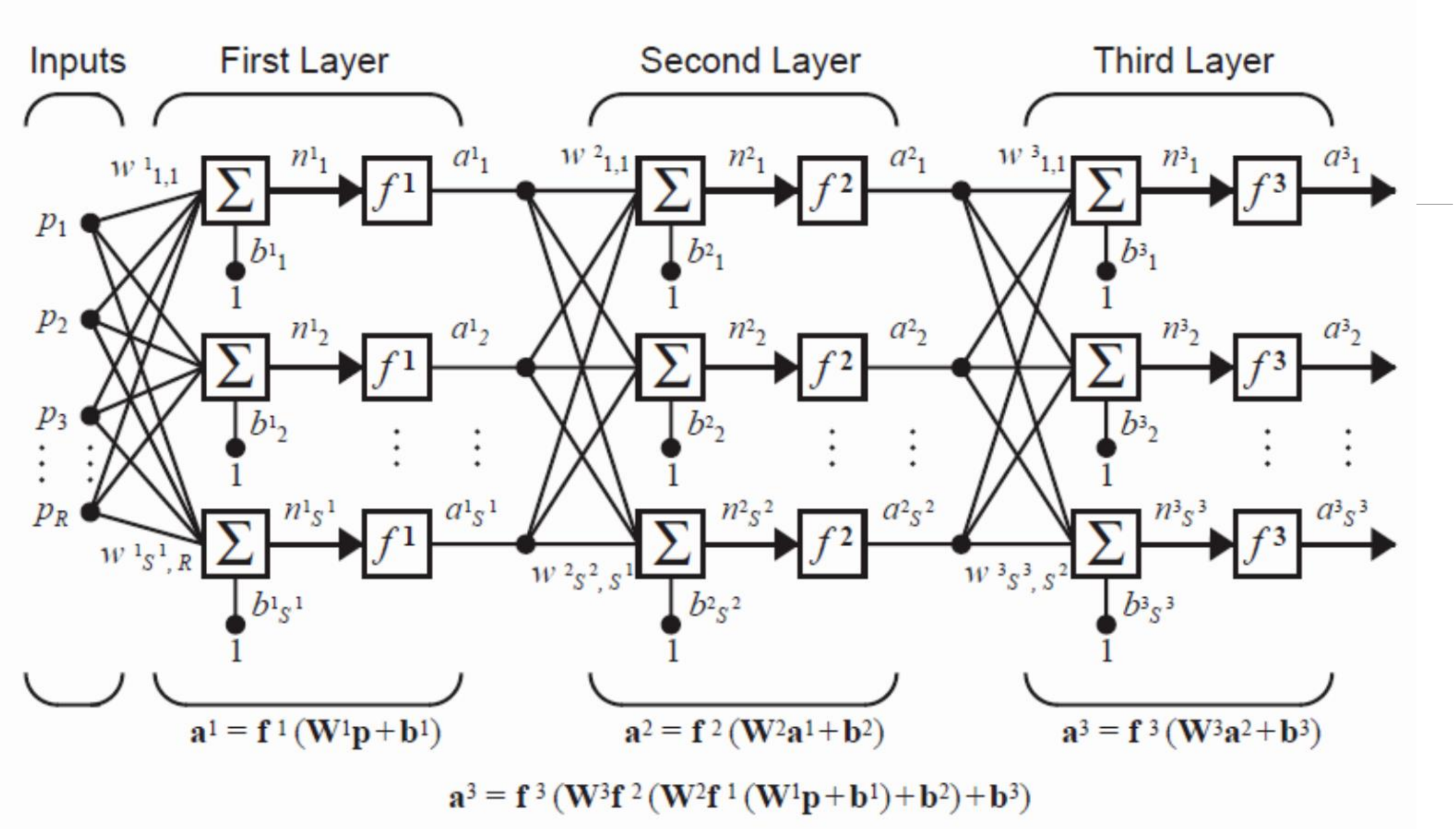

Inputs
Individual Decisions
AND Operation
p_1
p_2
2
2
-1
-1
n^1_1
n^1_2
a^1_1
a^1_2
-1
1.5
1
1
1
n^2_1
a^2_1
-1.5
1

Multilayer Perceptron

Multilayer perceptron is simply a cascade of multi-neuron single-layer perceptron networks.

The first layer is the input layer, the last layer is the output layer, all other layers are hidden layers.

The notation R-S^1-S^2-S^3 is also used to represent multilayer perceptron (4 layers here)



$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

Function Estimation

A multilayer perceptron network can be considered as a system with:

- Inputs ($x_1, \dots, x_n$)
- Weights (W_{ij})
- Thresholds (T_{ij})
- Outputs ($z_1, \dots, z_m$)

We may assume the network as a function $\mathbf{z}=f(\mathbf{x},\mathbf{W},\mathbf{T})$

Function Estimation

Therefore, by changing network parameters, we can approach the original function of

$$\mathbf{t} = g(\mathbf{x})$$

Function Estimation

How good can a neural network estimate a function?

P = dist(t, z)

dist = | t – z | (this function is mathematically inconvenient)

A better function can be dist = $| t – z |^2$

Function Estimation

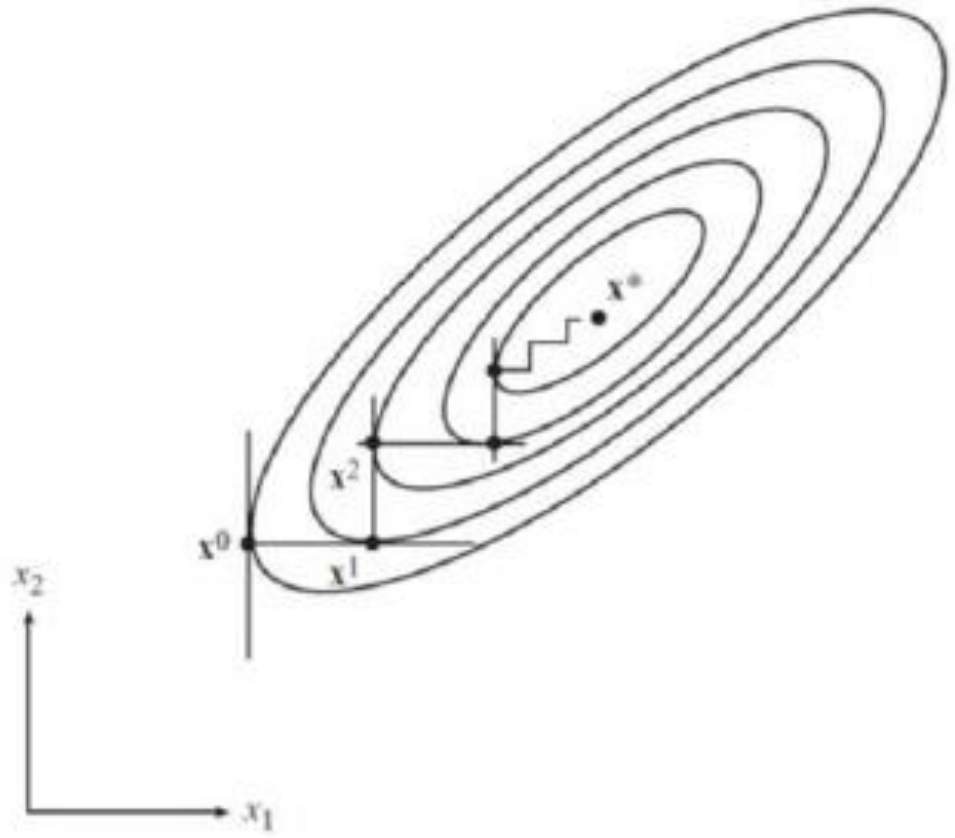
The goal of neural network learning can be defined as minimizing P value (distance between the true and the actual outputs)

Learning as Error Minimization

The learning in a neural network can be considered as a process for adjusting the network parameters to minimize the estimation error.

Starting from the initial weight values, we can follow the steepest descent towards the minimum error

Steepest Descent Minimization



Finding the Direction of Steepest Descent

Find the gradient of P using the following formula:

$P = |t - z|^2$

$$\nabla P = [\frac{\partial P}{\partial w_{ij}}]$$

Applying the Steepest Descent to Multilayer Perceptron

Forward propagation:

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \text{ for } m = 0, 1, \ldots, M-1$$

Backward propagation:

$$\mathbf{W}^{m}(k+1) = \mathbf{W}^{m}(k) - \alpha \mathbf{s}^{m}(\mathbf{a}^{m-1})^{T}$$

Applying the Steepest Descent to Multilayer Perceptron

The steepest descent direction is found from

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a}),$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}, \text{ for } m = M-1, \ldots, 2, 1,$$

Gradient Directions

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \dots & 0 \\ 0 & \dot{f}^m(n_2^m) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & \dot{f}^m(n_{S^m}^m) \end{bmatrix},$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}.$$

Learning Rate

The changes made to each weight can be adjusted with a scalar called learning rate

Hence the learning (updating the weights) will be in the form of

$$W_{ij}(k) = W_{ij}(k-1) - \alpha\Delta(W_{ij})$$

Where α is the learning rate

Batch versus Incremental Training

The backpropagation learning algorithm as described here involves incremental training, in which the network weights and biases are updated after each input is presented

Batch Training

It is also possible to perform batch training, in which the complete gradient is computed (after all inputs are applied to the network) before the weights and biases are updated.

Batch Training

In this case, if each input occurs with equal probability, the mean square error performance index can be written:

$$E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})] = \frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q)$$

Batch Training

The total gradient becomes:

$$\nabla\left\{\frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q)\right\}=\frac{1}{Q}\sum_{q=1}^{Q}\nabla\{(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q)\}$$

Drawbacks of Backpropagation

LMS algorithm is guaranteed to converge to a solution that minimizes the mean squared error.

This is true because the mean squared error for a single-layer linear network is a quadratic function.

The quadratic function has only a single stationary point.

Drawbacks of Backpropagation

When applied to multilayer networks, however, the performance surface for a multilayer network may have many local minimum points, and the curvature can vary widely in different regions of the parameter space.

Assignment-A

Using a multilayer perceptron estimate the following function:

$z = f(x,y) = y + \sin(x)$

Use MATLAB to create your network and train it. The network should have 3 layers (input, hidden, output). Train the network using backpropagation algorithm and evaluate the network using random input values.

Deadline: December 29th, 2017

Assignment-B

Discuss how different values of learning rate can affect the performance of the neural network. Explain if the network can avoid local minima by adjusting learning rate.

Besides, explain if it is possible to using an adaptive learning rate. How?

Deadline: December 6th, 2017