# SysEng 5212 /EE 5370
# Introduction to Neural Networks and Applications

## *Week 5 :Multilayer Perceptrons and Backpropagation Learning*

**Cihan H Dagli, PhD**

*Professor of Engineering Management and Systems Engineering*
*Professor of Electrical and Computer Engineering*
*Founder and Director of Systems Engineering Graduate Program*

dagli@mst..edu

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY
Rolla, Missouri, U.S.A.

SYSTEMS ENGINEERING
Research Center
Department of Defense UARC

Smart Engineering Systems Lab
Engineering Management and Systems Engineering Department

# Volunteers Tally

- Murat Aslan
- Tatiana Cardona Sepulveda
- Prince Codjoe
- Xiongming Dai
- Jeffrey Dierker

# Volunteers Tally

- Venkata Sai Abhishek Dwivadula
- Brian Guenther
- Anthony Guertin
- Timothy Guertin
- Seth Kitchen

# Volunteers Tally

- Gregory Leach
- Yu Li
- John Nganga
- Igor Povarich
- Jack Savage

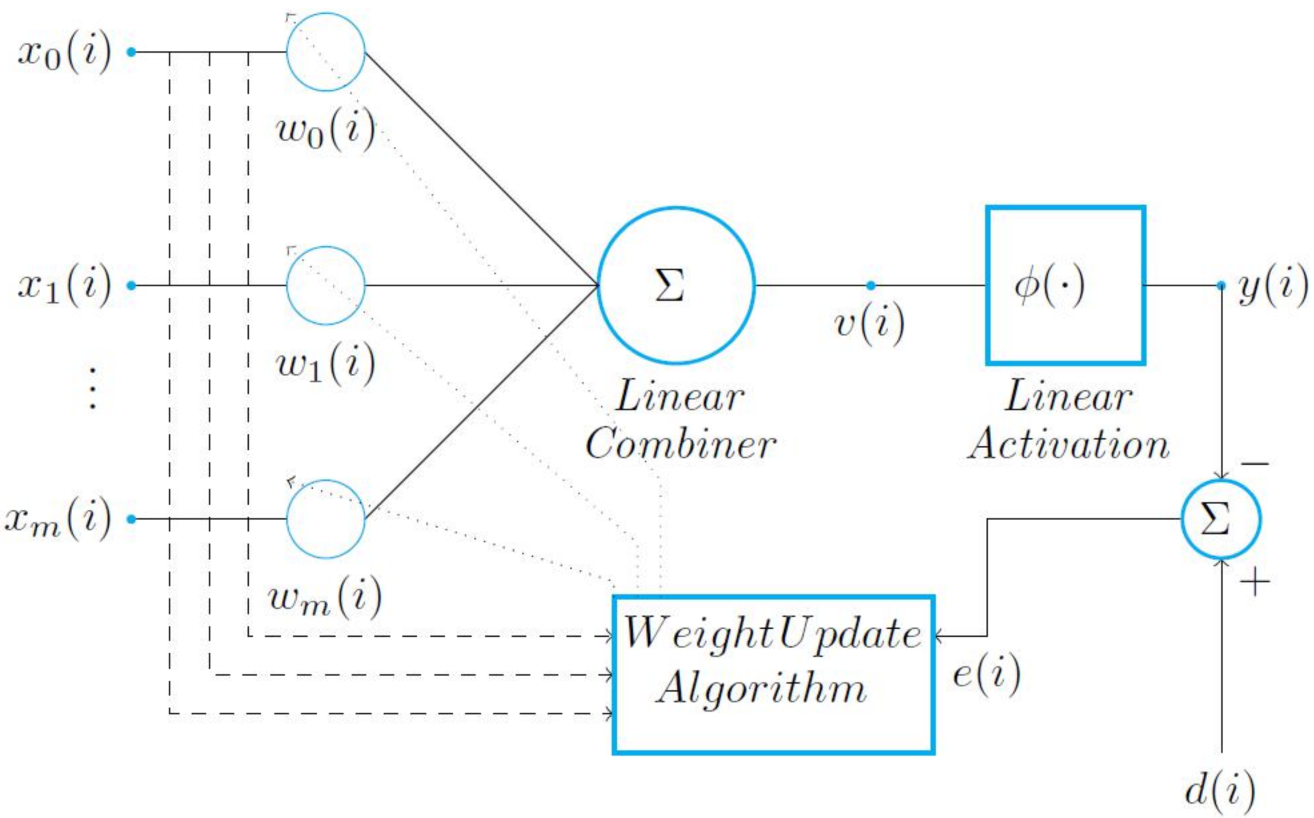
# Volunteers Tally

- William Symolon
- Wayne Viers III
- Tao Wang
- Kari Ward
- Julia White
- Jun Xu

# Lecture outline

1. Nonlinear Perceptron
   – Character Recognition Example
2. Multilayer Perceptrons
   – XOR Problem
3. Backpropagation
   – Derivation of Backpropagation Weight Update Rule
   – Backpropagation Examples
4. Improving Performance

# Perceptron with a Nonlinear Activation Function




这个就是y=v，斜率等于1的直线

MATLAB: y = purelin(v)

# Network Output

Given that the output of the linear combiner is, $v(n) = \boldsymbol{w}^{\boldsymbol{T}} x$, the network output is given by,

$$y(n) = \boldsymbol{\Phi}(v(n))$$

Most commonly used nonlinear activation function is the sigmoidal nonlinearity; Its two variations are,

- Logistic function, where a > 0 is the slope parameter.

$$\boldsymbol{\Phi}\big(v(n)\big) = \frac{1}{1+e^{-av(n)}}$$

双曲线的 正切的

- Hyperbolic tangent function, where a and b are positive constants

$$\boldsymbol{\Phi}\big(v(n)\big) = a\tanh(bv(n))$$

# Weight Update Rule for the Linear Neuron

Weight update for the linear neuron,

$$\hat{w}(\mathrm{n}+1) = \hat{w}(n) + \eta x(n)\hat{e}(n)$$

Where $-x(n)\hat{e}(n)$ is the instantaneous estimate of the gradient.

瞬间的

The error e(n) is,

$$e(n) = d(n) - y(n) = d(n) - \boldsymbol{\Phi}(v(n))$$

The derivative of the error e'(n) is,

$$e'(n) = -\Phi'\big(v(n)\big) = \Phi(\boldsymbol{w}^{\boldsymbol{T}}\,\boldsymbol{x}) = -x(n)$$

这个就是对w的求导

# Weight Update Rule for the Nonlinear Neuron

For the nonlinear neuron, the derivative of the error is,

$$e'(n) = -\Phi'\big(v(n)\big)$$

and the instantaneous estimate of the gradient is given by,

$$\widehat{w}(\mathrm{n}+1) = \widehat{w}(n) + \eta\hat{e}(n)\,\Phi'\big(v(n)\big)$$

# Derivative of the Logistic Function

Logistic function:

$$\phi(v(n)) = \frac{1}{1+e^{-av(n)}}$$

求导的时候，直接把a作为1计算。

The derivative:

$$\begin{aligned}
\phi'(v(n)) &= \frac{ae^{-av(n)}}{[1+e^{-av(n)}]^2} \\
&= a\left(\frac{e^{-av(n)}}{[1+e^{-av(n)}]}\right)\left(\frac{1}{[1+e^{-av(n)}]}\right) \\
&= ay(n)[1-y(n)]
\end{aligned}$$

LMS weight update with the logistic function,

$$\hat{w}(n+1) = \hat{w}(n) + a\eta\hat{e}(n)y(n)[1-y(n)]$$

# Derivative of the Hyperbolic Tangent Function

Hyperbolic tangent function:

$$\phi(v(n)) = a\tanh(bv(n))$$

The derivative:

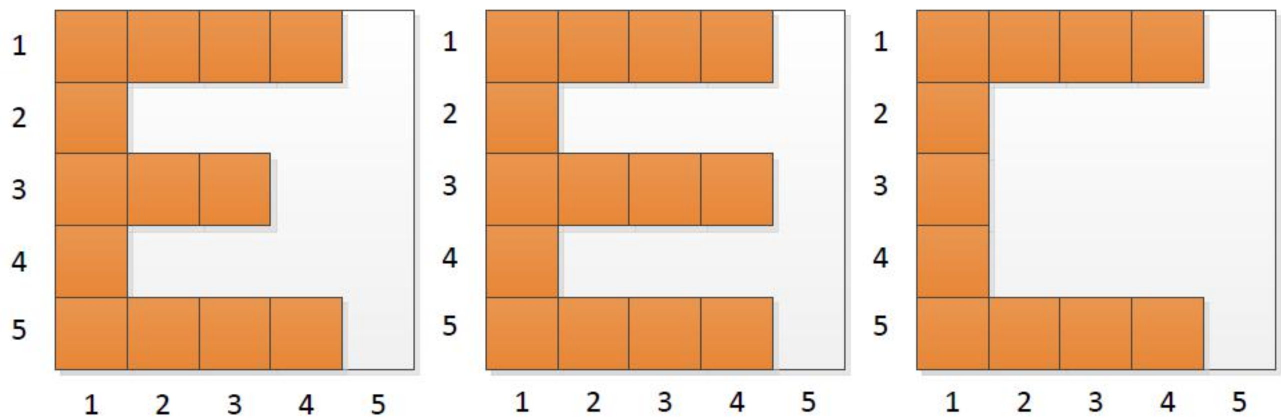
$$\begin{aligned}\phi'(v(n)) &= ab\,\mathrm{sech}^2(bv(n)) \\ &= ab[1-\tanh^2(bv(n))] \\ &= ab\left[1-\frac{y^2(n)}{a^2}\right] \\ &= \frac{b}{a}[a-y(n)][a+y(n)]\end{aligned}$$

LMS weight update with the hyperbolic tangent function,

$$\hat{w}(n+1) = \hat{w}(n) + \frac{b}{a}\eta\hat{e}(n)[a-y(n)][a+y(n)]$$

# Character Recognition Example

Use a perceptron with a sigmoid activation function to learn the letter **E**.




Each image consists of a 5  5 array of pixels.

- The ON pixels have a value of 1
- The OFF pixels have a value of 0

# Preparing the Input

Each input character is a 5  5 array,

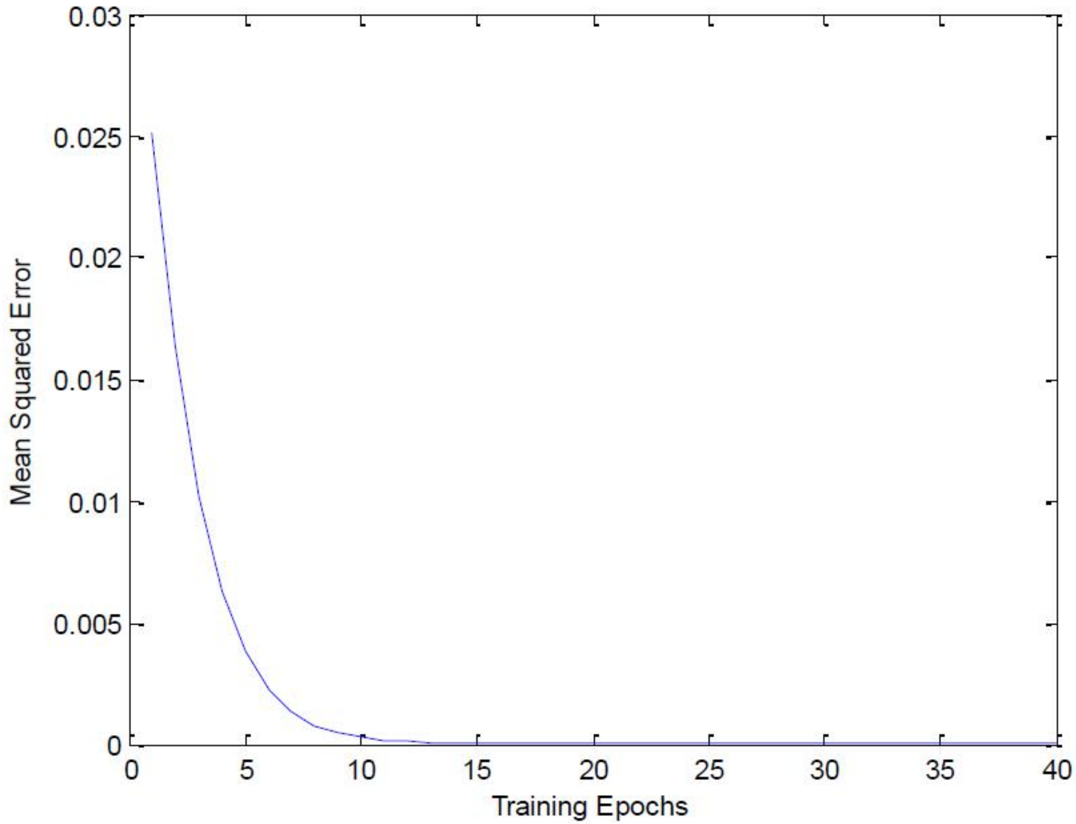
$$x=\begin{bmatrix}1 & 1 & 1 & 1 & 0\\ 1 & 0 & 0 & 0 & 0\\ 1 & 1 & 1 & 0 & 0\\ 1 & 0 & 0 & 0 & 0\\ 1 & 1 & 1 & 1 & 0\end{bmatrix}$$

To present the input to the neuron, we vectorize it,

$$x = [1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0]^T$$

We set the desired response to 0.5

# Performance Curve



# Compensating for Noisy Input

补偿，修正

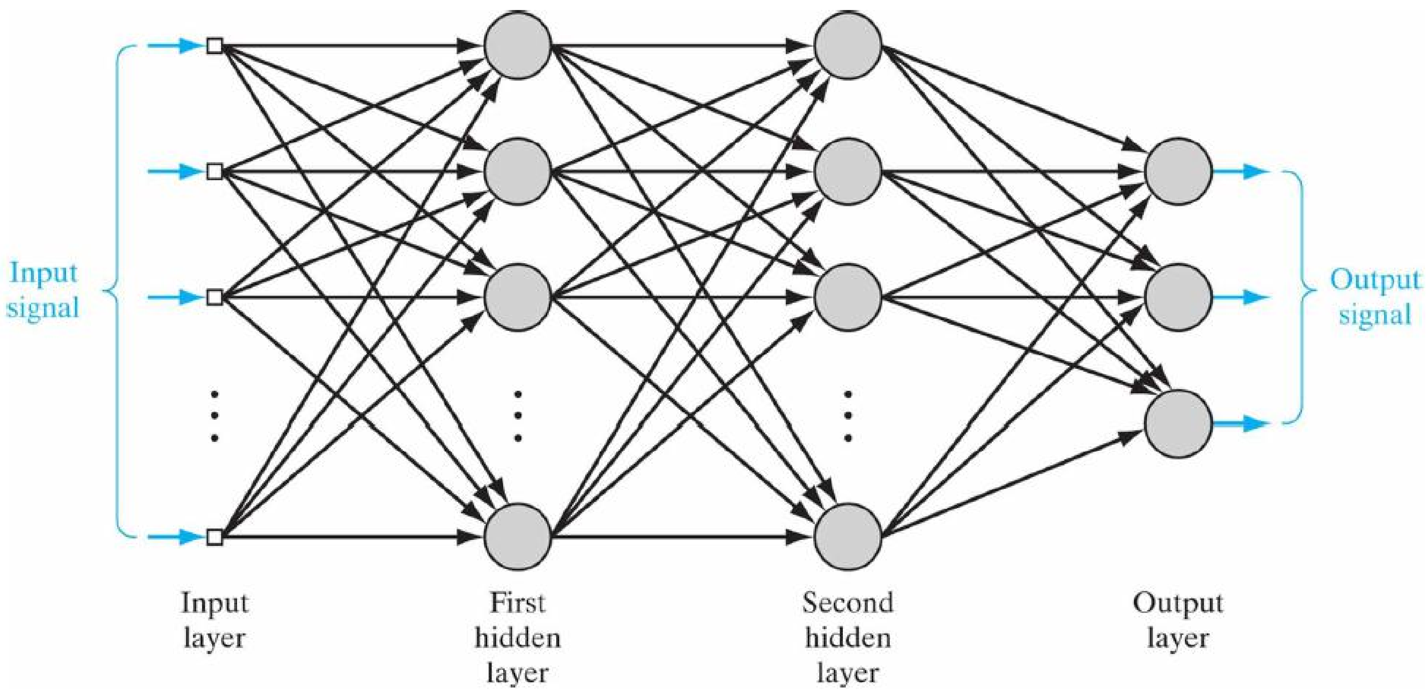
Vectors representing noisy input,(red numbers are noise)

$x = [1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0]^T$

$x = [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0]^T$

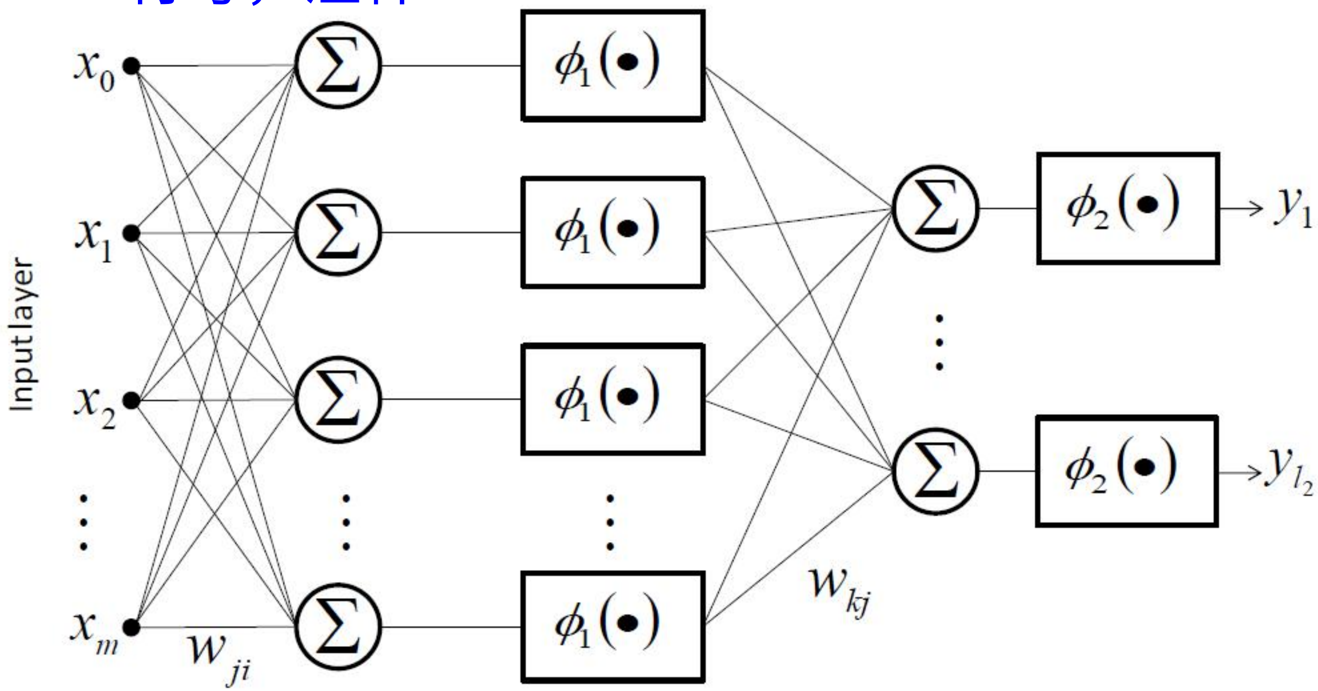
- A single neuron cannot compensate very well for noisy input. As noise increases performance degrades.

# Layered Architecture



# Notation and Network Output

符号，注释




$i = 0 : m_0$ $\quad j = 1 : m_1$ $\quad k = 1 \rightarrow : m_2$

# Basic Features of an MLP

- Each neuron has a nonlinear activation function that is continuously differentiable

- Each network contains one or more hidden layers

- Network exhibits a high degree of connectivity determined by the synaptic weights of the network

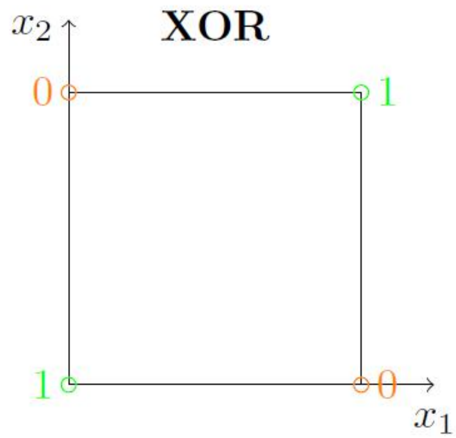
# Purpose of the Hidden Neurons

- Transform the input space into a new space called the feature space
- Act as feature detectors that identify the most useful components of the input
- Separation of classes becomes easier in this new space than in the original input space
- Provide the MLP with the capability to perform nonlinear separation.

# Solving the XOR Problem

| $x_1$ | $x_1$ | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

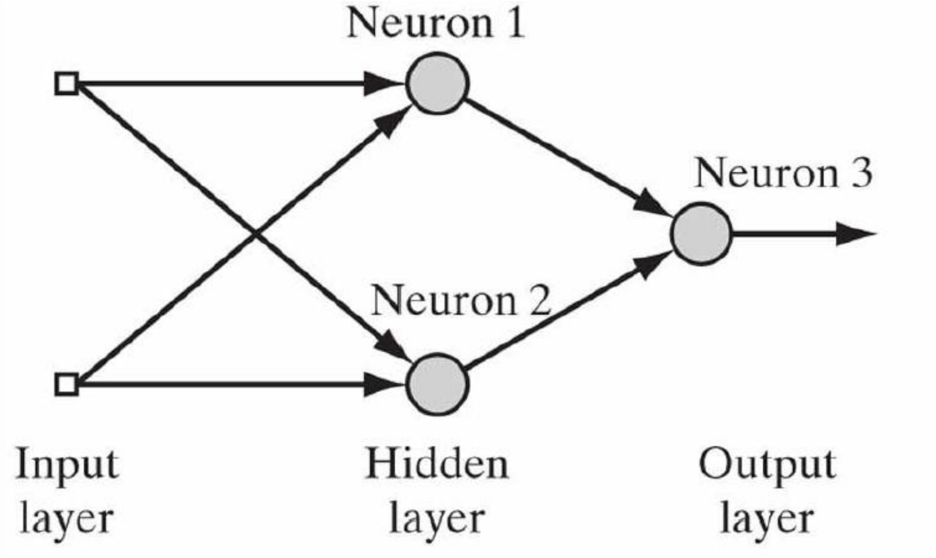
Separation of the input space for the XOR function,



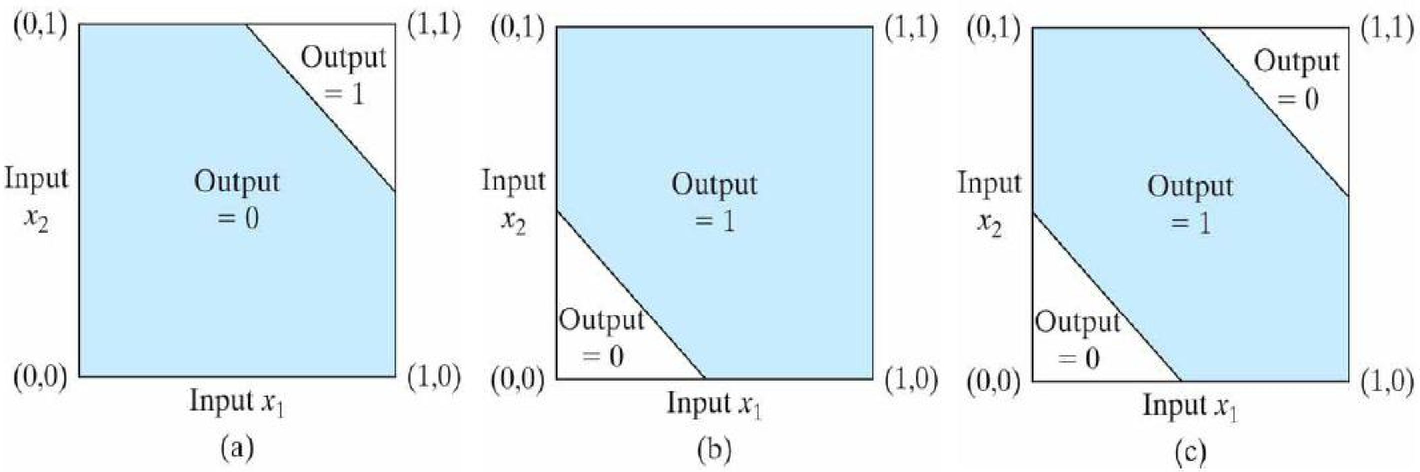
# Multilayer Network for Solving the XOR Problem

Rosenblatt's perceptron could not classify the XOR input patterns, as they are not linearly separable.

We can solve this problem by using a network with a single hidden layer with two neurons.



# Decision Boundaries of the Network

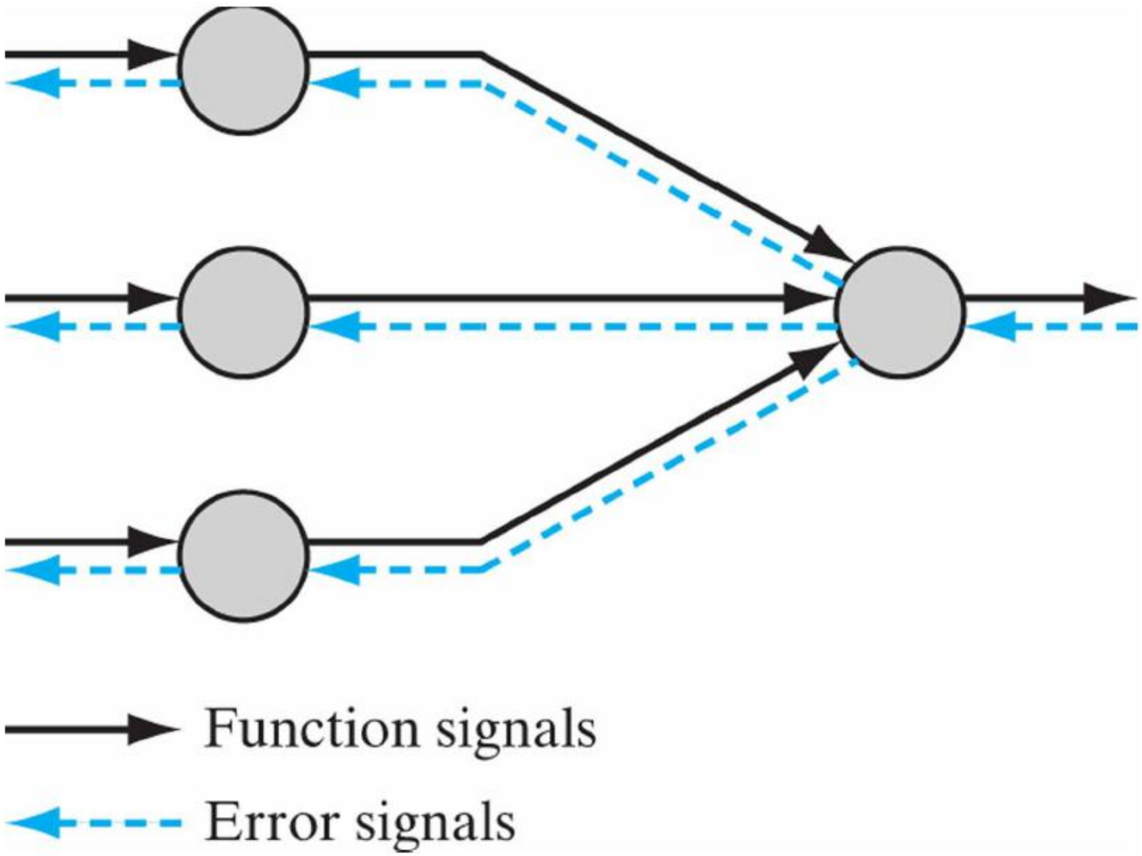



$$\{w_1 1 \; = w_1 2 = \; +1, \quad b_1 = -3/2\}$$

$$\{w_2 1 \; = w_2 2 = \; +1, \quad b_1 = -1/2\}$$

$$\{w_3 1 \; = -2, \; w_3 2 = \; +1, \quad b_1 = -1/2\}$$

# Signal Flows in a Multilayer Perceptron



# Forward Pass: Computation of Network Error

**Output at layer 1:**

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) x_i(n)$$

$$y_j(n) \ = \Phi(v_j(n))$$

**Output at layer 2 (output layer):**

$$v_k(n) = \sum_{i=0}^{m_1} w_{kj}(n) y_j(n)$$

$$y_k(n) \ = \Phi(v_k(n))$$

**Error at the output layer:**

$$e_k(n) = d_k(n) - y_k(n)$$

# Generalizing the LMS Cost Function

From the LMS derivation, instantaneous error for a single neuron k is $\frac{1}{2}\ e^2{}_k(\mathrm{n})$. Combined error of all neurons in the output layer for iteration n is,

又是这个二分之一惹的祸！！！！

$$\varepsilon = \frac{1}{2}\sum_{j\epsilon C} e^2{}_k(n)$$

Average squared error over all the input samples (N) is,

$$\mathcal{E}_{av} = \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n)$$

The objective of the learning process is to minimize the average squared error. To do so, we will consider an incremental training approach.

# Chain Rule of Calculus

$$\frac{df(n(t))}{dt} = \frac{df(n)}{dn} \times \frac{dn(t)}{dt}$$

Consider, $f(n) = ln(n)$, $n = t^2$, and $f(n(t)) = ln(t^2)$.

$$\frac{df(n(t))}{dt} = \frac{df(n)}{dn} \times \frac{dn(t)}{dt} = \frac{1}{n} \times 2t = \frac{2t}{n}$$

# Backward Pass: Weight Correction

Similar to the LMS algorithm, the backpropagation algorithm applies the correction $\Delta w_{kj}(n)$.

$$\frac{\partial \mathcal{E}(n)}{\partial w_{kj}(n)} \sim \triangle w_{kj}(n)$$

Using the chain rule of calculus,

$$\frac{\partial \mathcal{E}(n)}{\partial w_{kj}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{kj}(n)}$$

# Computing the Gradient

Differentiating w.r.t $e_k(n)$, with regard to;with reference to;

$$\frac{\partial \varepsilon(n)}{\partial e_k(n)} = e_k(n)$$

Differentiating w.r.t $y_k(n)$,

$$\frac{\partial e_k(n)}{\partial y_k(n)} = -1$$

Differentiating w.r.t $v_k(n)$,

$$\frac{\partial y_k(n)}{\partial v_k(n)} = \phi_k'(v_k(n))$$

Differentiating w.r.t $w_{jk}(n)$,

$$\frac{\partial v_k(n)}{\partial w_{jk}(n)} = y_j(n)$$

where $y_j(n) = x_i(n)$ for the first layer.

Thus the gradient is given by,

$$\frac{\partial \mathcal{E}(n)}{\partial w_{kj}(n)} = -e_k(n)\phi_k'(v_k(n))y_j(n)$$

From the LMS algorithm, the error correction $\triangle w_{kj}(n)$ applied to weight $w_{kj}(n)$ is defined by the *delta rule*,

$$\triangle w_{kj}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{kj}(n)}$$

which can be rewritten as,

$$\triangle w_{kj}(n) = \eta \delta_k(n) y_j(n)$$

where $\delta_k(n)$ is known as the *local gradient*.

$$\begin{aligned} \delta_k(n) &= \frac{\partial \mathcal{E}(n)}{\partial v_k(n)} \\ &= e_k(n)\phi_k'(v_k(n)) \end{aligned}$$

# Case 1: Output Neuron

The error and the local gradient for output neurons can be computed in a straightforward manner using the expressions we just derived.

$$e_k(n) = d_k(n) - y_k(n)$$

$$\delta_k(\mathrm{n}) = e_k(n)\, \Phi_k{}'\big(v(n)\big)$$

Thus the error correction for output neurons is,

$$\Delta w_{kj}(n) = \ \eta\delta_k(n) y_j(n)$$

# Case 2: Hidden Neuron

- No desired responses are available for the hidden neurons.
- The error signal must be worked out backwards.
- The error at each hidden neuron is a combination of its share of the error at each output neuron.

# Case 2: Hidden Neuron

Local gradient for the hidden neuron,

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$
$$= -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \phi_j'(v_j(n))$$

To calculate $\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}$, consider the error at the output layer,

$$\mathcal{E}(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$$

Differentiate w.r.t $y_j(n)$,

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k}{\partial y_j(n)}$$

Using the chain rule, we expand the above result,

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k}{\partial v_k(n)} \frac{\partial v_k}{\partial y_j(n)}$$

Error at the output neuron $k$ is,

$$e_k(n) = d_k(n) - \phi_k(v_k(n))$$

Hence,

$$\frac{\partial e_k}{\partial v_k(n)} = -\phi'_k(v_k(n))$$

Recall that $v_k(n) = \sum_{j=0}^{m_1} w_{kj}(n) y_j(n)$; Differentiating w.r.t $y_j(n,$

$$\frac{\partial v_k}{\partial y_j(n)} = w_{kj}(n)$$

Thus, $\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}$ is given by,

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = -\sum_k e_k \phi'_k(v_k(n)) w_{kj}(n) = -\sum_k \delta_k(n) w_{kj}(n)$$

Finally the backpropagation formula for local gradient of the hidden neuron is,

$$\delta_j(n) = \phi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

Thus the error correction for the hidden neuron is,

$$\triangle w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

# Summary of Weight Update Rules for Backpropagation

The weight correction $\Delta w_{ji}(n)$ applied to the weight connecting any neuron $i$ to any neuron $j$ is given by the delta rule,

$$\Delta w_{kj}(n) = \eta\delta_j(n)y_i(n)$$

Where $y_i(n)$ is the input signal of neuron j, and the local gradient $\delta_j(n)$ varies as,

1. If j is an output neuron,

$$\delta_j(\mathrm{n}) = e_j(n)\,\Phi_j{}'\big(v_j(n)\big)$$

2. If j is a hidden neuron,

$$\delta_j(\mathrm{n}) = \Phi_j{}'\big(v_j(n)\big)\sum_k \delta_k(\mathrm{n})w_{kj}(n)$$

# Computation of the Local Gradient

- To compute $\delta$, we need to compute the derivative of the activation function.
- The activation function of an MLP must be nonlinear and continuously differentiable.
- Two commonly used activation functions are the logistic function and the hyperbolic tangent function.

# Local Gradient for the Logistic Function

We calculated the derivative of the logistic function as,

$$\phi'(v(n)) = ay(n)[1 - y(n)]$$

- The local gradient of the output neuron $j$ with a logistic activation

$$\begin{aligned}\delta_j(n) &= e_j(n)\phi_j'(v_j(n)) \\ &= a[d_j(n) - y_j(n)]y_j(n)[1 - y_j(n)]\end{aligned}$$

- The local gradient of the hidden neuron $j$ with a logistic activation

$$\begin{aligned}\delta_j(n) &= \phi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \\ &= ay_j(n)[1 - y_j(n)] \sum_k \delta_k(n) w_{kj}(n)\end{aligned}$$

$k$ is a neuron in the next layer.

# Local Gradient for the Hyperbolic Tangent Function

Hyperbolic tangent function:

$$\phi(v(n)) = a \tanh(bv(n))$$

The derivative:

$$\begin{aligned}
\phi'(v(n)) &= ab \operatorname{sech}^2(bv(n)) \\
&= ab[1 - \tanh^2(bv(n))] \\
&= ab\left[1 - \frac{y^2(n)}{a^2}\right] \\
&= \frac{b}{a}[a - y(n)][a + y(n)]
\end{aligned}$$

LMS weight update with the hyperbolic tangent function,

$$\hat{w}(n+1) = \hat{w}(n) + \frac{b}{a}\eta\hat{e}(n)[a - y(n)][a + y(n)]$$

# Generalized Delta Rule

The rule for updating the synaptic weights can be written as,

$$w_{ji}(n+1) = w_{ji}(n) + \eta\delta_j(n)y_i(n)$$

$$y_i(n) = x(n) \text{ for layer 1.}$$

The value of $\delta_j(n)$ depends on whether neuron j is an output neuron or a hidden neuron.

# Comments on Learning Rate

- Backpropagation is a generalization of the LMS algorithm.
- The learning rate behaves similar to the that of the LMS algorithm
  - smaller learning rate → smoother trajectory → slower rate of convergence
  - Larger learning rate → zigzagging trajectory → faster rate of learning; possible unstable behavior

# Comments on Learning Rate

To avoid instability while increasing the learning rate, a modified delta rule can be used,

$$\Delta w_{ji}(n) = \alpha\, w_{ji}(n-1) + \eta\delta_j(n)y_i(n)$$

or,

$$w_{ji}(n+1) = w_{ji}(n) + \alpha[\, w_{ji}(n+1)] + \eta\delta_j(n)y_i(n)$$

可能是n

Here, $\alpha\, w_{ji}(n-1)$ is the momentum term.

- $\alpha$ is the momentum constant; $0 \leq \alpha < 1$
- $\alpha$ is generally positive
- Setting $\alpha = 0$ gives the original delta rule

Momentum accelerates descent, however it has a stabilizing effect on oscillatory behavior of the direction of descent.

# Training Methods

How often should we update the weights?

- after each input sample
- after a complete presentation of all input samples
- after a small batch of input samples

How do we set the learning rate?

- Fixed learning rate
- Use annealing methods
- Use momentum with learning rate

# Stopping Criteria

Some `reasonable' criteria for stopping training,

- When the norm of the gradient vector goes below a threshold value
- When the average squared error per iteration is below a threshold value
- When the generalization performance has peaked

# Example

- https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

# Function Approximation Example

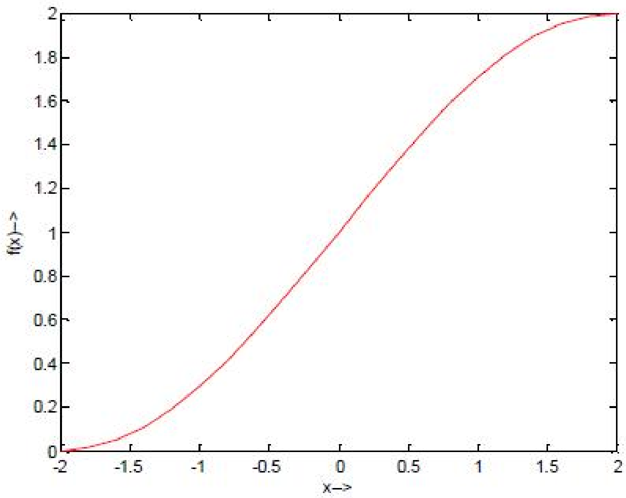
We want to use a MLP trained using backpropagation to approximate the function,

$$f(x) = 1 + \sin\frac{pi}{4}x, \qquad \text{for } -2 \le x \le 2$$

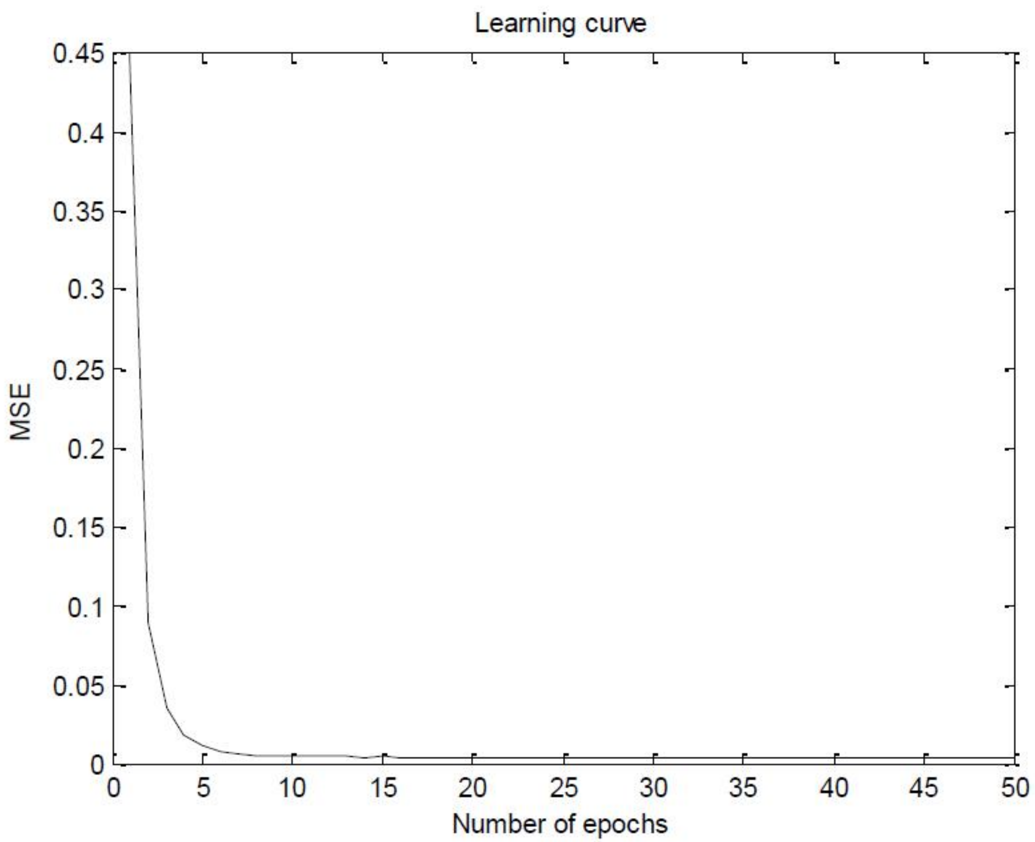
Create a dataset by evaluating the function for several values of x within the specified interval.

Input: x

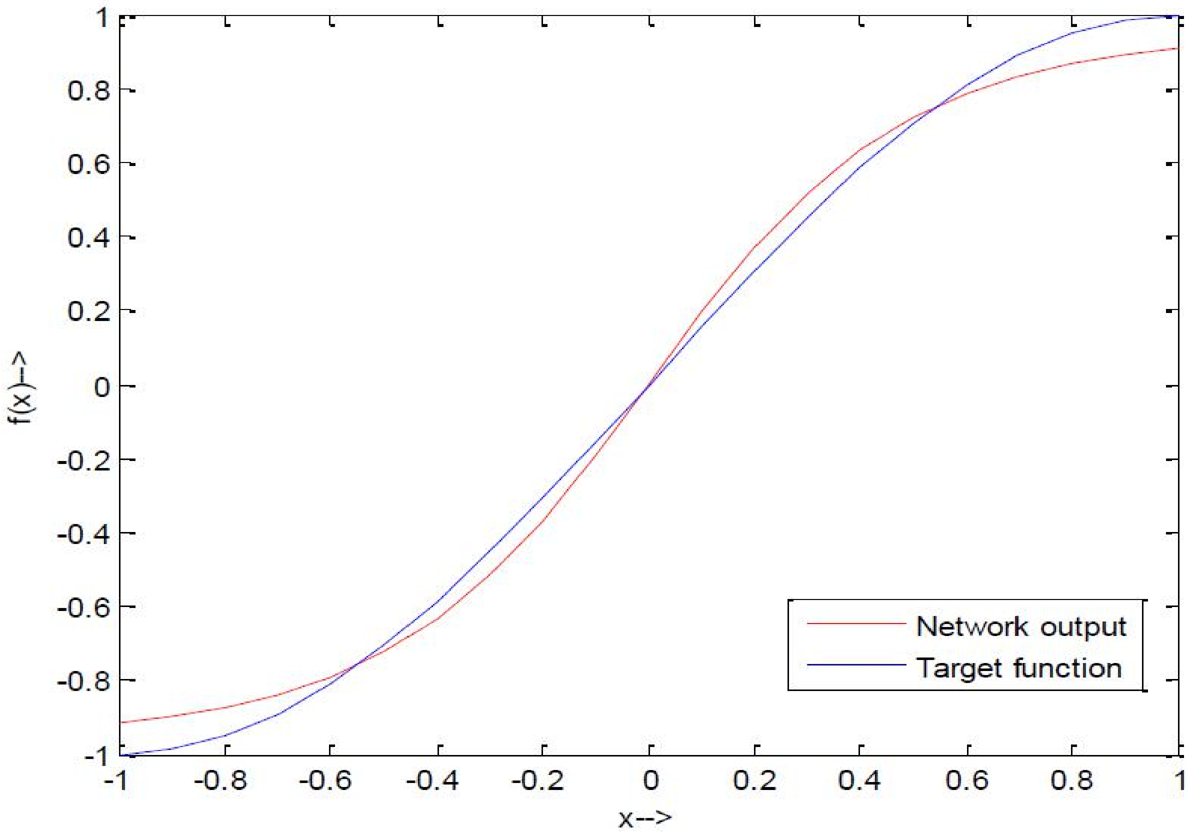
Target: F(x)



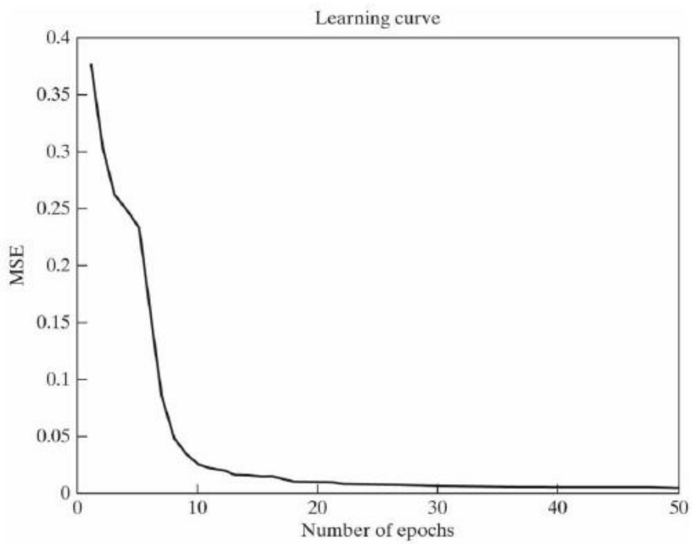
# Learning Curve



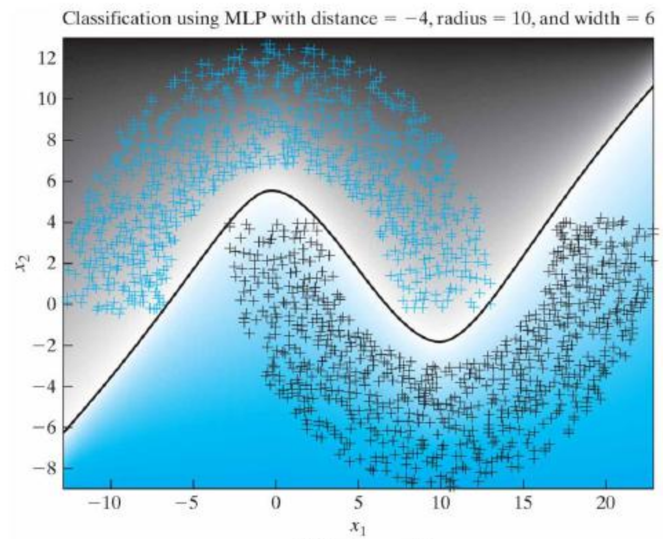
# Network Performance



# Pattern Classification Example

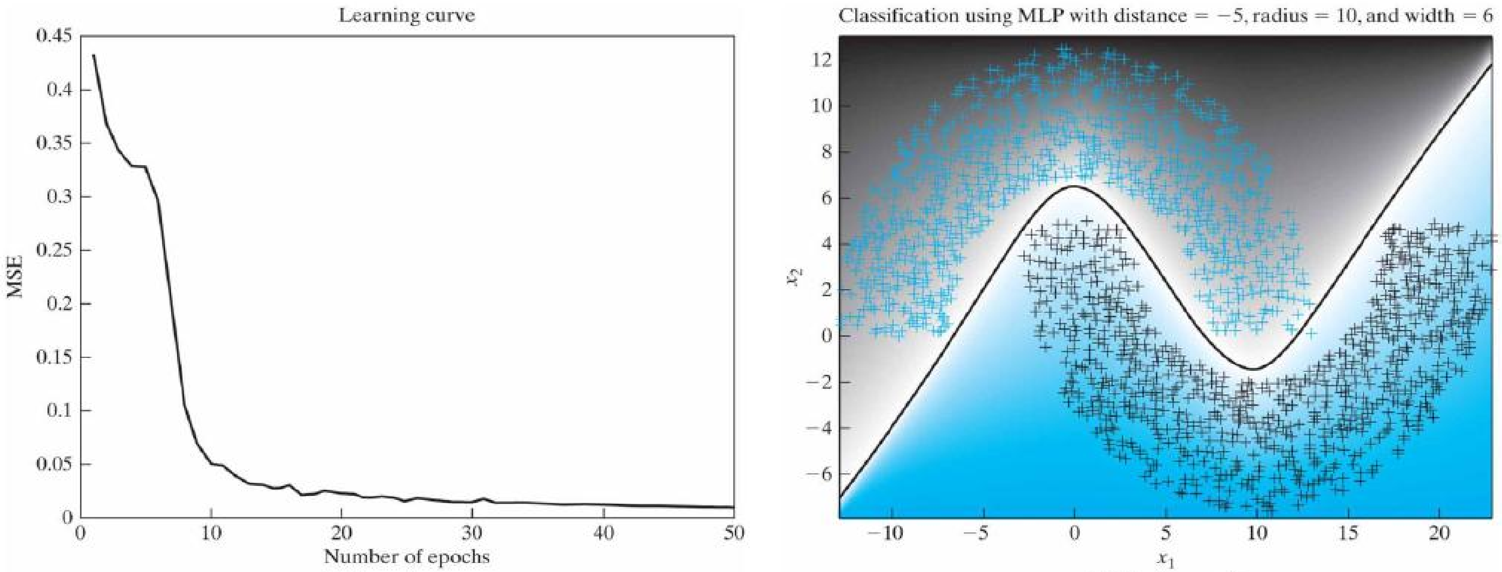


(a) Learning curve



(b) Testing result

Results of the computer experiment on the backpropagation algorithm applied to the MLP with distance $d = -4$.




(a) Learning curve (b) Testing result

Results of the computer experiment on the back-propagation algorithm applied to the MLP with distance $d = -5$

# Heuristics for Improving the Performance of the Backpropagation Algorithm

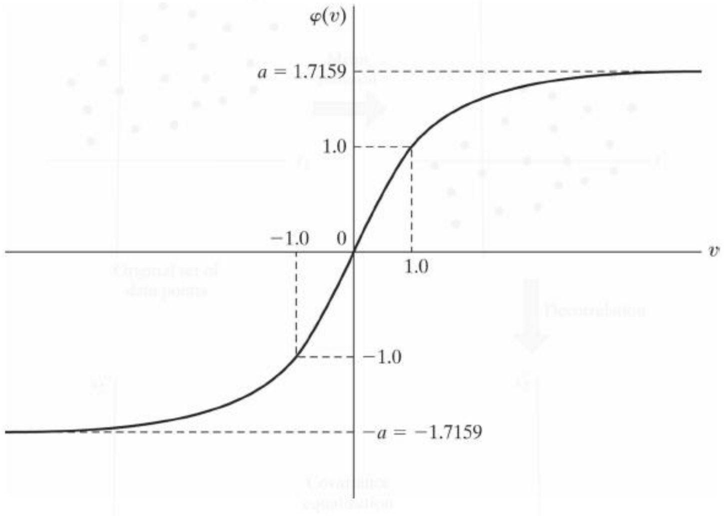
- Update scheme: Batch vs Sequential training is faster and less computationally intensive.

- Maximize information content Choose training samples that is different from previous samples and results in maximum training error.

洗牌

- Shuffle inputs before presenting to the network; As far as possible make sure each successive input belongs to a different class.

# Heuristics: Activation Function




Use an activation that is an odd function of its argument,

$\phi(-v) = - \phi(v)$

This condition is met by the hyperbolic tangent function,

$\phi(v) = = a \tanh(bv)$

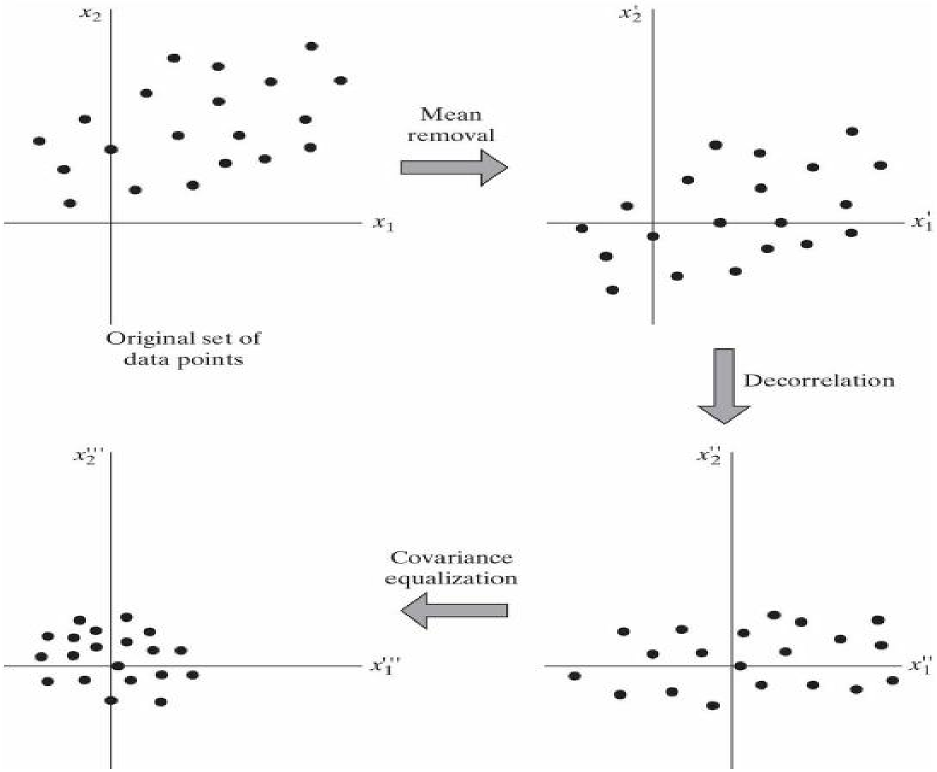
# Heuristics: Setting the Target Values

Use an offset to move the desired response away from the limiting values of the sigmoid function.

$$d_j = a - \varepsilon$$
$$d_j = -a + \varepsilon$$

where ε is a positive constant.

# Heuristics: Normalizing Inputs



# Heuristics: Other Suggestions

- Initialization: Avoid very large and very small values of initial weights. A safe range is [-0.5, 0.5].
- Learning rates: Ideally all neurons in the network should learn at the same rate. Neurons with many inputs should have a lower learning rate than neurons with fewer inputs, to ensure a uniform learning time.