

# **CS 461**

# **Artificial Intelligence**

Dr. Hashim Yasin

# Perceptron Training Rule

- ▶ The ***perceptron training rule***, which revises the weight  $w_i$  associated with input  $x_i$  according to the rule:

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t$  is target value
- $o$  is perceptron output
- $\eta$  is small constant (e.g., 0.1) called *learning rate*

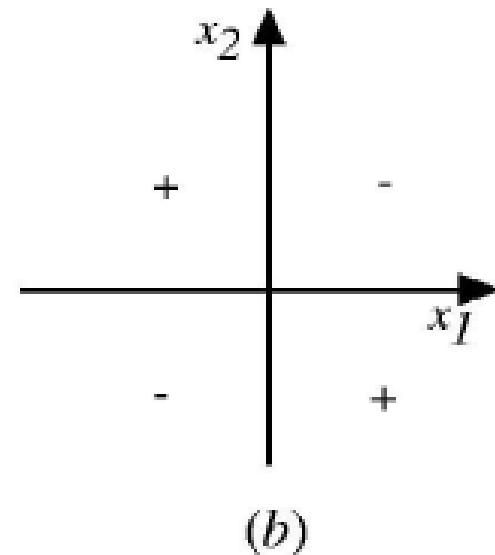
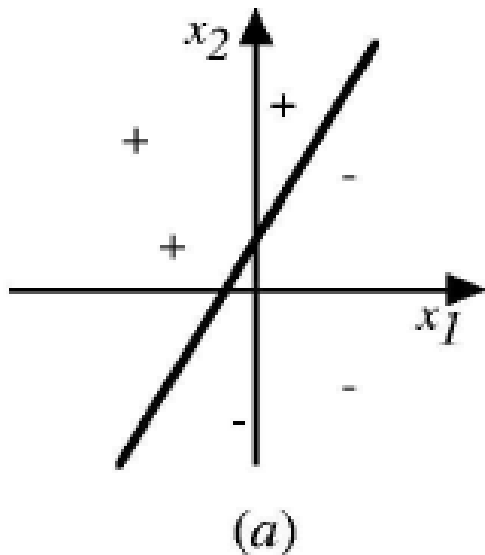
# Perceptron Training Rule

- ▶ The **perceptron rule** finds a successful weight vector when the training examples are **linearly separable**,
- ▶ It fails to converge if the examples are **not linearly separable**.
- ▶ The solution is ... **Delta Rule** also known as (**Widrow-Hoff Rule**)

## Delta Rule

- ▶ use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

# Perceptron



The **decision surface** represented by a **two-input perceptron  $x_1$  and  $x_2$** .  
(a) A set of training examples and the decision surface of a perceptron that classifies them correctly. (b) A set of training examples that is not linearly separable.

# Delta Rule

# Delta Rule

- ▶ In perceptron training rule we employ *thresholded perceptron*

$$o(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

- ▶ The delta training rule is the task of training an *unthresholded perceptron*;
  - a *linear unit* for which the output  $o$  is given by

$$o(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$$

- A linear unit corresponds to the *first stage* of a perceptron, *without* the threshold.

# Delta Rule

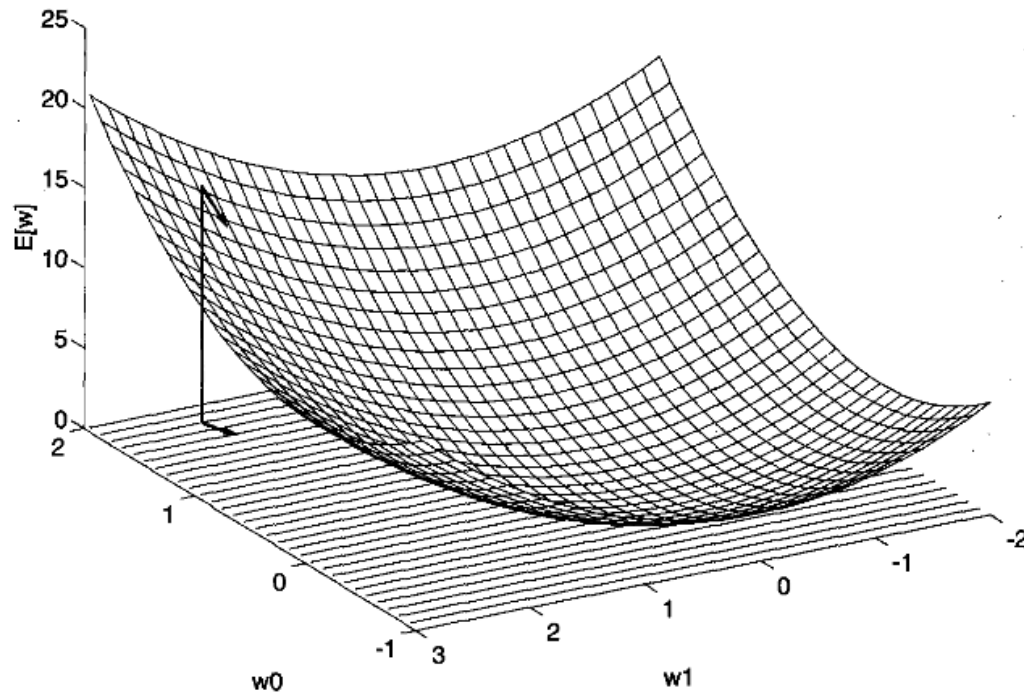
- ▶ In order to derive **a weight learning rule for linear units**,
  - Specify a measure for the **training error** of a hypothesis (weight vector), relative to the training examples.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- $D$  is the set of training examples,
- $t_d$  is the target output for training example  $d$ ,
- $o_d$  is the output of the linear unit for training example  $d$
- $E$  is characterized as a function of  $\mathbf{w}$ , because the linear unit output  $o$  depends on this weight vector.

# Hypothesis Space

- ▶ For a linear unit with two weights, the hypothesis space  $H$  is the  $w_0, w_1$  plane.



Error of different hypotheses.



# Gradient Descent

- ▶ Gradient descent search determines *a weight vector that minimizes  $E$*  by
  - ❑ Starting with an arbitrary initial weight vector,
  - ❑ Repeatedly modifying it in small steps.
  - ❑ At each step, the ***weight vector is altered in the direction*** that produces the **steepest descent** along the error surface,
  - ❑ This process continues until the **global minimum error** is reached.

# Gradient Descent

*How can we calculate the direction of steepest descent along the error surface?*

- ▶ This direction can be found by computing the derivative of  $E$  with respect to each component of the vector  $\mathbf{w}$ .

$$\nabla E(\vec{w}) \equiv \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- ▶ Notice  $\nabla E(\vec{w})$  is itself a vector, whose components are the partial derivatives of  $E$  with respect to each of the  $w_i$ .

# Gradient Descent

- ▶ The **gradient specifies the direction** that produces the steepest increase in  $E$ . The training rule for gradient descent is,

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w}$$

where,

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

- ▶ The negative sign is present because we want to move the weight vector in the direction that ***decreases***  $E$ .

# Gradient Descent

- ▶ This training rule can also be written in its **component form**,

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w}$$

$$w_i \leftarrow w_i + \Delta w_i$$

where,

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

- ▶ The steepest descent is achieved by altering each component  $w_i$  in proportion to  $\frac{\partial E}{\partial w_i}$

# Gradient Descent

- ▶ The vector of derivatives  $\frac{\partial E}{\partial w_i}$  that form the gradient can be obtained by differentiating  $E$  from delta rule

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$o(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$$

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \end{aligned}$$

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d) (-x_{id})$$

# Gradient Descent

- ▶ We now have an equation that gives  $\frac{\partial E}{\partial w_i}$  in terms of
  - the linear unit inputs  $x_{id}$ ,
  - outputs  $o_d$ ,
  - target values  $t_d$  associated with the training examples
- ▶ The weight update rule for gradient descent becomes,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

# Gradient Descent

GRADIENT-DESCENT(*training\_examples*,  $\eta$ )

*Each training example is a pair of the form  $\langle \vec{x}, t \rangle$ , where  $\vec{x}$  is the vector of input values, and  $t$  is the target output value.  $\eta$  is the learning rate (e.g., .05).*

- Initialize each  $w_i$  to some small random value
- Until the termination condition is met, Do
  - Initialize each  $\Delta w_i$  to zero.
  - For each  $\langle \vec{x}, t \rangle$  in *training\_examples*, Do
    - Input the instance  $\vec{x}$  to the unit and compute the output  $o$
    - For each linear unit weight  $w_i$ , Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

- For each linear unit weight  $w_i$ , Do

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

# Gradient Descent

- ▶ Gradient descent is an important general paradigm for learning.
- ▶ It is a *strategy for searching through a large or infinite hypothesis space* that can be applied whenever
  - 1) the hypothesis space contains **continuously parameterized hypotheses** (e.g., the weights in a linear unit),
  - 2) the **error can be differentiated** with respect to these hypothesis parameters



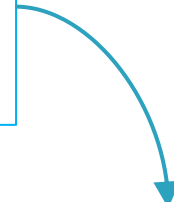
# Gradient Descent

- ▶ The **key practical difficulties** in applying gradient descent are:
  - a) **converging to a local minimum*** can sometimes be **quite slow** (i.e., it can require many thousands of gradient descent steps),
  - b) **if there are multiple local minima*** in the error surface, then there is no guarantee that the procedure will find the global minimum.

# Stochastic Gradient Descent

- ▶ The idea behind stochastic gradient descent is to approximate the gradient descent search by **updating weights incrementally**, following the calculation of the error for **each individual example**.

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$


$$\Delta w_i = \eta (t - o) x_i$$

# Stochastic Gradient Descent

- ▶ One way to view this stochastic gradient descent is to consider a **distinct error function** defined for *each individual training example  $d$  as follows*.

$$E_d(\vec{w}) = \frac{1}{2}(t_d - o_d)^2$$

- where  $t$ , and  $o_d$  are the target value and the unit output value for training example  $d$ .
- Stochastic gradient descent iterates over the training examples  $d$  in  $D$ ,
- at each iteration altering weights according to the gradient

# Stochastic Gradient Descent

GRADIENT-DESCENT(*training\_examples*,  $\eta$ )

*Each training example is a pair of the form  $\langle \vec{x}, t \rangle$ , where  $\vec{x}$  is the vector of input values, and  $t$  is the target output value.  $\eta$  is the learning rate (e.g., .05).*

- Initialize each  $w_i$  to some small random value
- Until the termination condition is met, Do
  - Initialize each  $w_i$  to zero.
  - For each  $\langle \vec{x}, t \rangle$  in *training\_examples*, Do
    - Input the instance  $\vec{x}$  to the unit and compute the output  $o$
    - For each linear unit weight  $w_i$ , Do

$$w_i \leftarrow w_i + \eta(t - o)x_i$$

# The Key Difference

- ▶ In **stochastic gradient descent**, weights are updated upon examining *each* training example.
- ▶ Whereas in **standard gradient descent**, the error is summed over *all* examples before updating weights,
  - ❑ **Standard gradient descent** requires **more computation** per weight update step.
  - ❑ **Standard gradient descent** is often used with a **larger step size** per weight update than stochastic gradient descent.

# The Key Difference

- ▶ When there are multiple local minima with respect to  $E(\mathbf{w})$ ,
  - ❑ The **stochastic gradient descent** can sometimes *avoid falling into these local minima*,
  - ❑ It is due to the reason that it uses various  $\nabla E_d(\vec{\mathbf{w}})$  rather than  $\nabla E(\vec{\mathbf{w}})$  to guide its search.

# Training Rules

- ▶ **Perceptron Training Rule:** guarantee to succeed if
  - training examples are linearly separable
  - Sufficiently small learning rate
- ▶ **Delta Rule:**
  - use gradient descent
  - converges only asymptotically toward the minimum error hypothesis,
  - converges regardless of whether the training data are linearly separable.

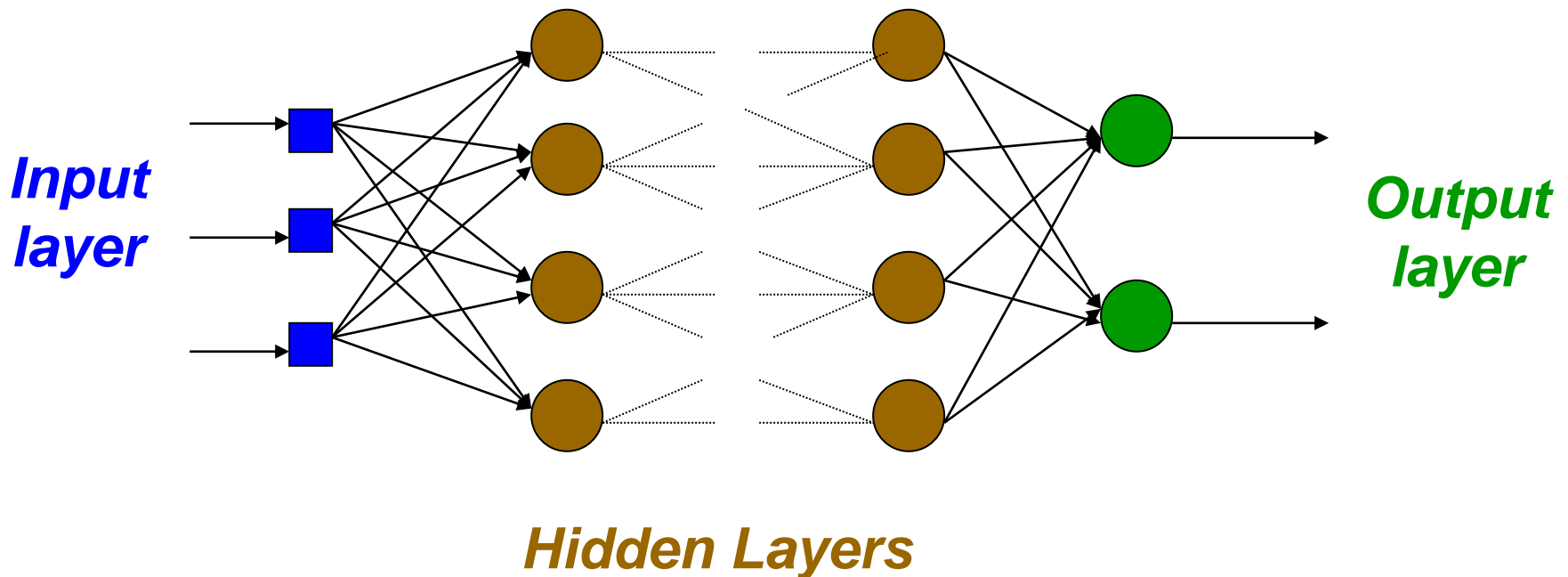
# Multilayer Networks





# Multilayer Perceptron Architecture

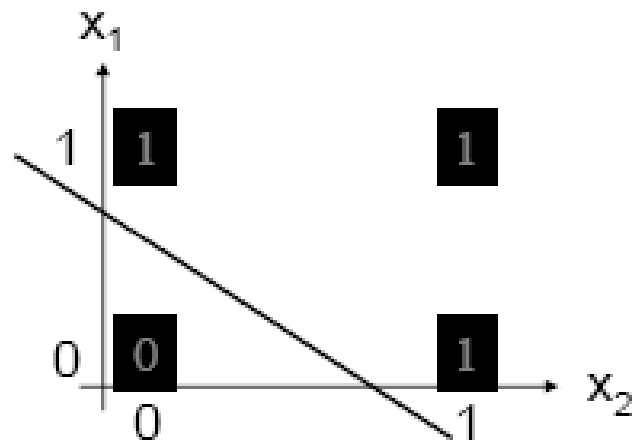
MLP used to describe any general feedforward (no recurrent connections) network



# Multilayer Networks

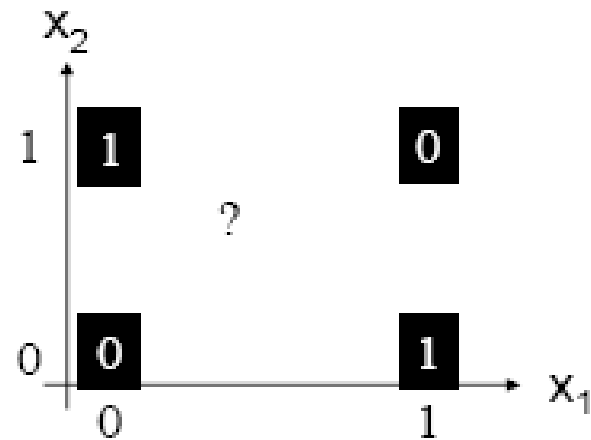
## OR function

$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1

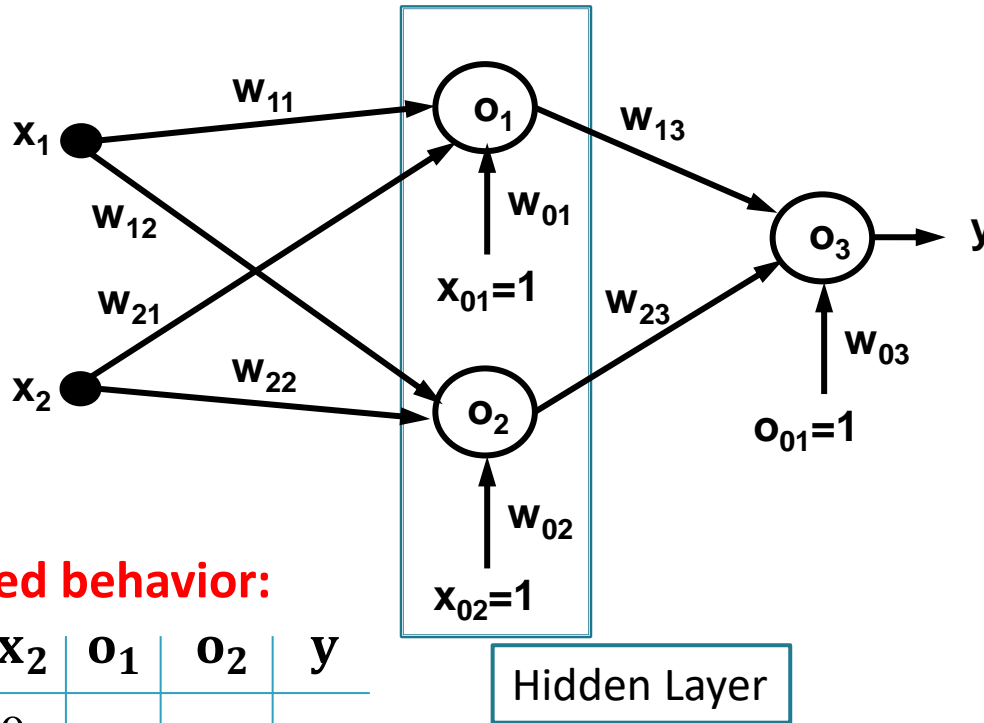


## XOR function

$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0



# Multilayer Networks



**Network Topology:**

2 hidden nodes

1 output

**Weights:**

$$w_{11} = w_{12} = 1$$

$$w_{21} = w_{22} = 1$$

$$w_{01} = -1.5$$

$$w_{02} = -0.5$$

$$w_{13} = -1$$

$$w_{23} = 1$$

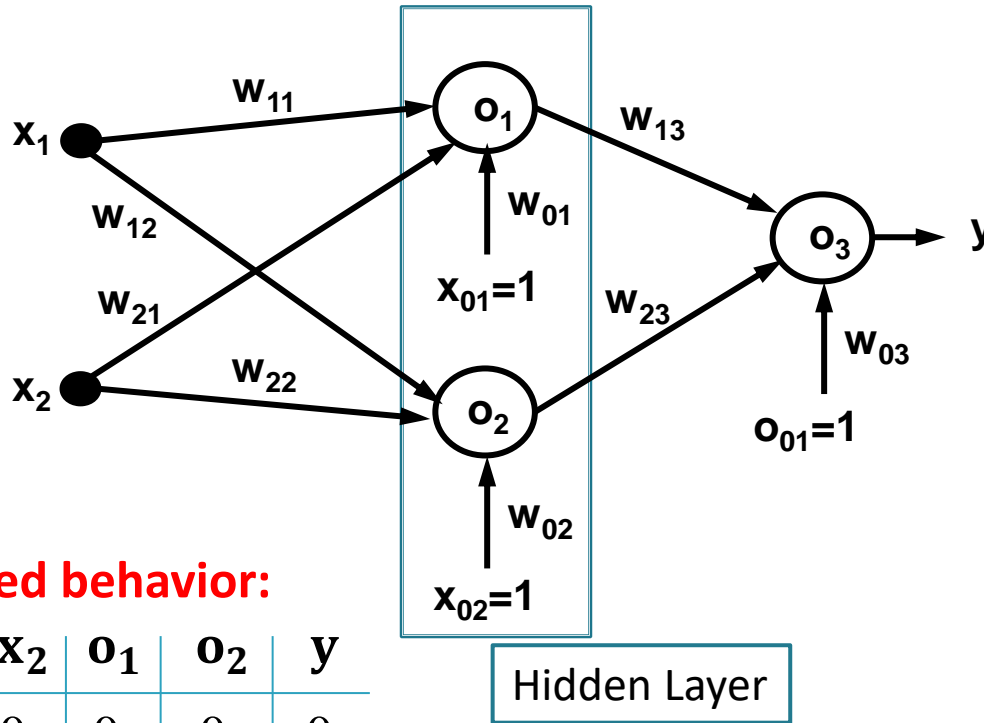
$$w_{03} = -0.5$$

**Desired behavior:**

$x_1$	$x_2$	$o_1$	$o_2$	$y$
0	0			
1	0			
0	1			
1	1			

Piecewise linear classification using an MLP with threshold (perceptron) units

# Multilayer Networks



**Network Topology:**

2 hidden nodes

1 output

**Weights:**

$$w_{11} = w_{12} = 1$$

$$w_{21} = w_{22} = 1$$

$$w_{01} = -1.5$$

$$w_{02} = -0.5$$

$$w_{13} = -1$$

$$w_{23} = 1$$

$$w_{03} = -0.5$$

**Desired behavior:**

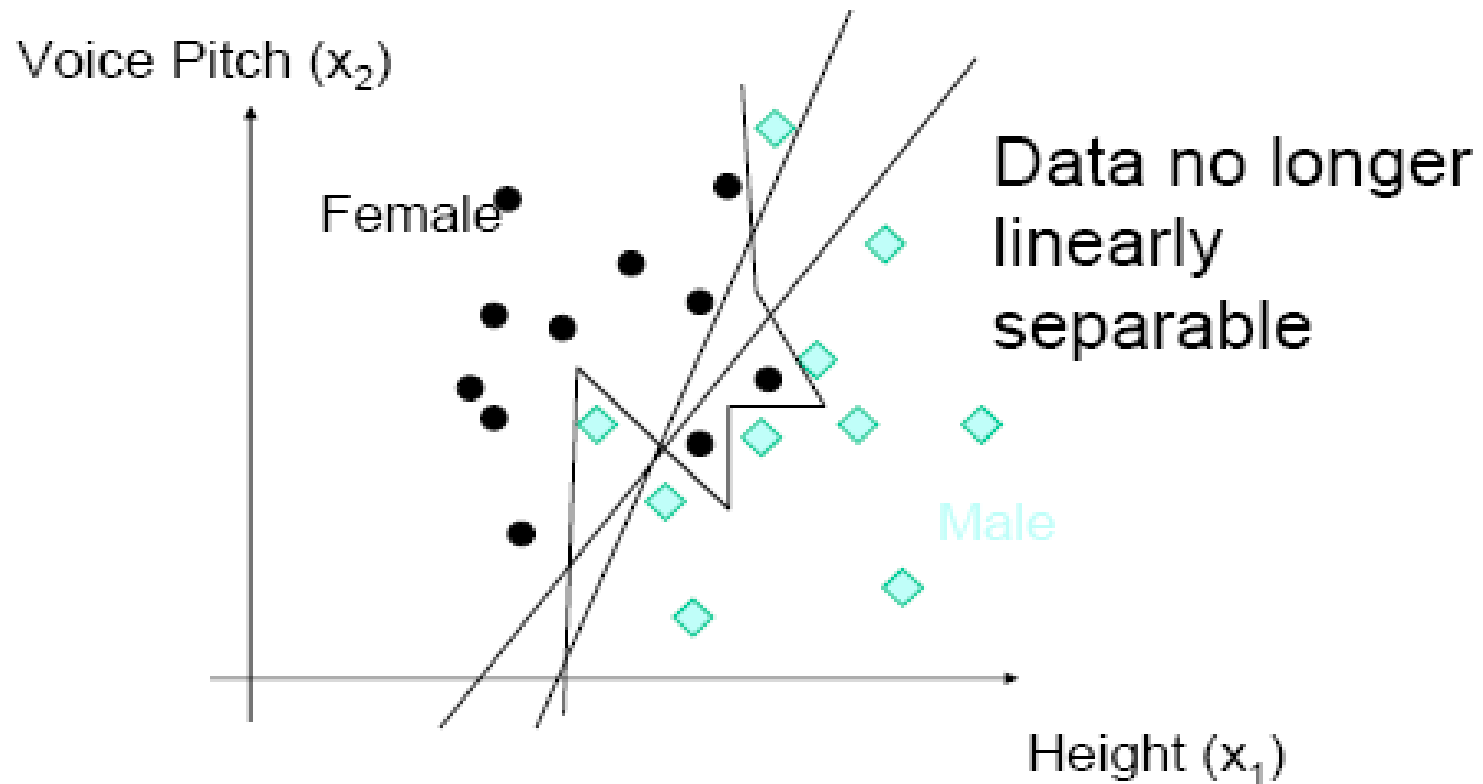
$x_1$	$x_2$	$o_1$	$o_2$	$y$
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

Piecewise linear classification using an MLP with threshold (perceptron) units

# Multilayer Networks

- ▶ The single perceptron can only express linear decision surfaces.
- ▶ The kind of **multilayer networks** learned by the **back propagation** algorithm are capable of expressing a rich variety of nonlinear decision surfaces.

# Multilayer Networks... Example



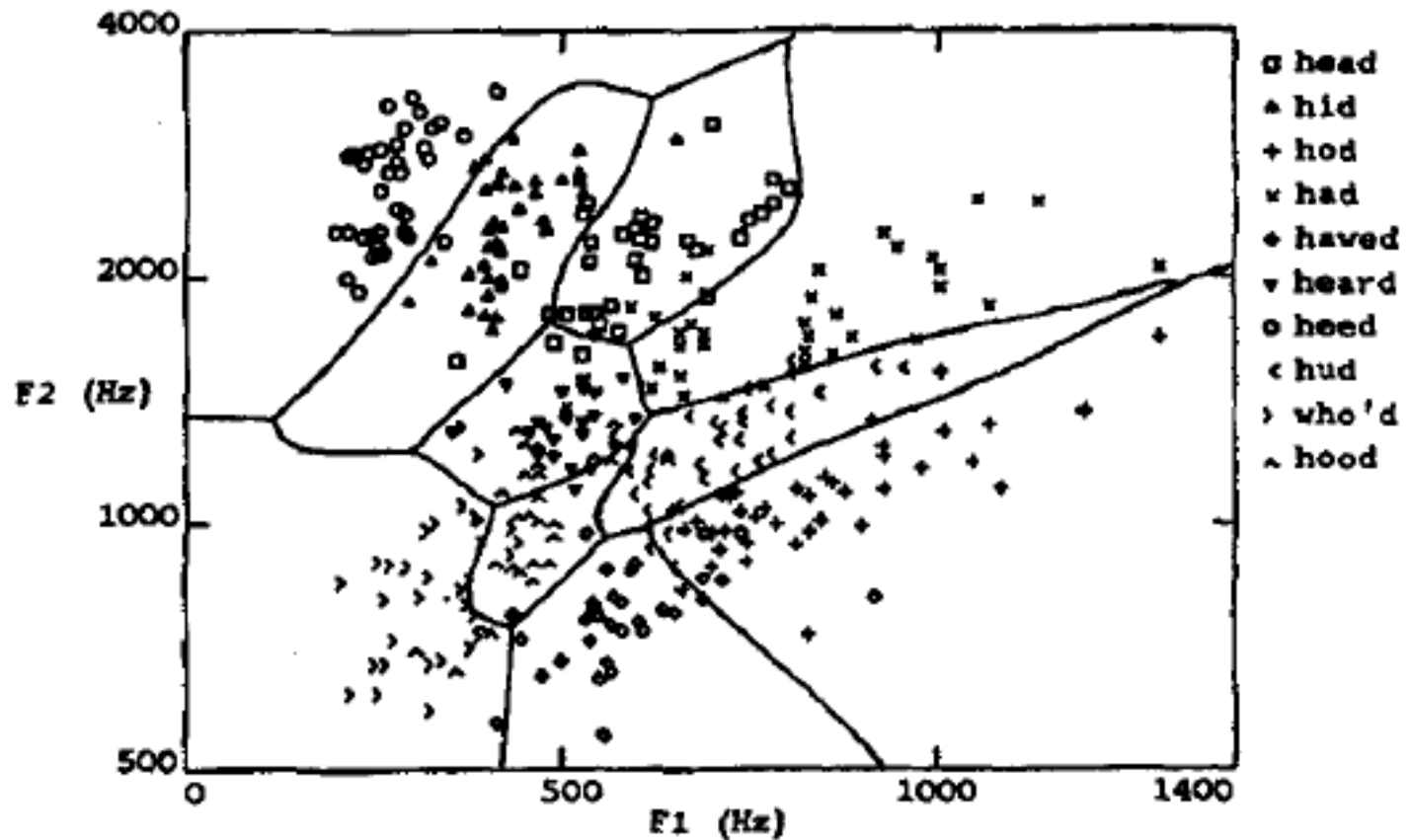
**What is a good decision boundary ?**

# Multilayer Networks... Example

## Example:

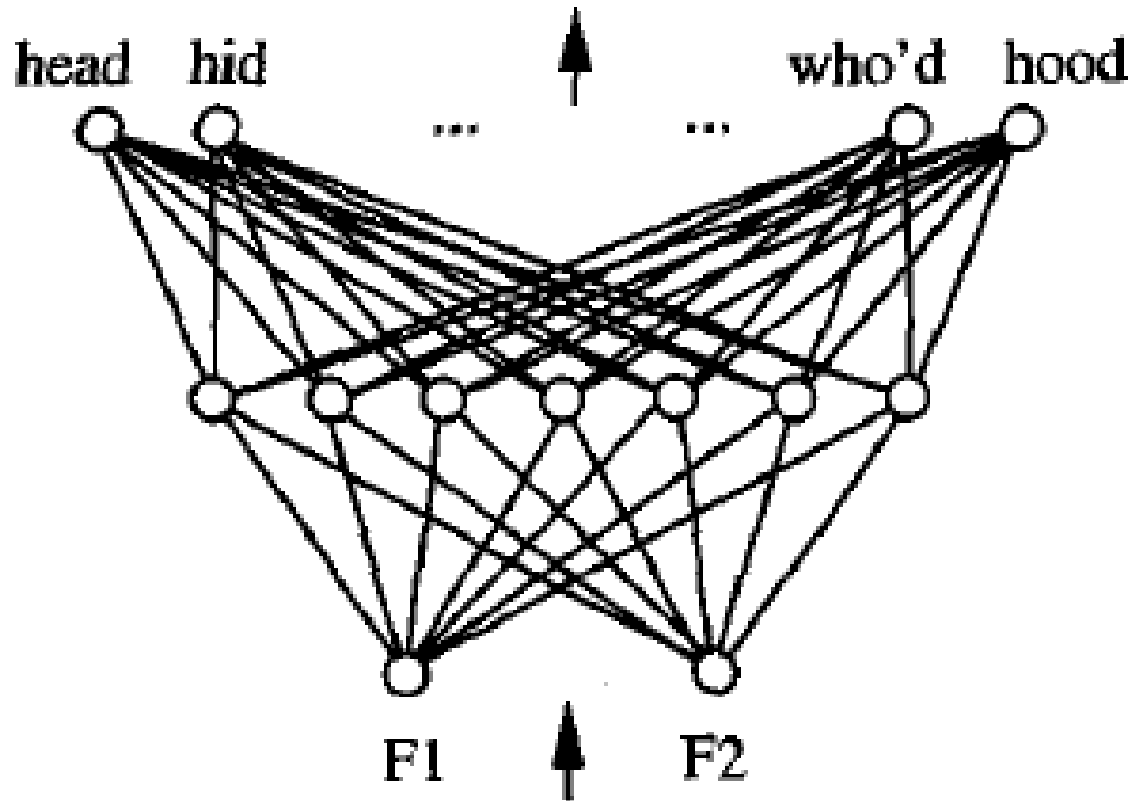
- ▶ The speech recognition task involves distinguishing among 10 possible vowels, all spoken in the context of "h-d" (i.e., "hid," "had," "head," "hood," etc.).

# Multilayer Networks... Example





# Multilayer Networks... Example



# Reading Material

- ▶ **Artificial Intelligence, A Modern Approach**  
**Stuart J. Russell and Peter Norvig**
  - Chapter 18.
- ▶ **Machine Learning**  
**Tom M. Mitchell**
  - Chapter 4.

