Introduction to the

Introduction to Artificial Neural Network

Vuong Le

with Hao Tang's slides

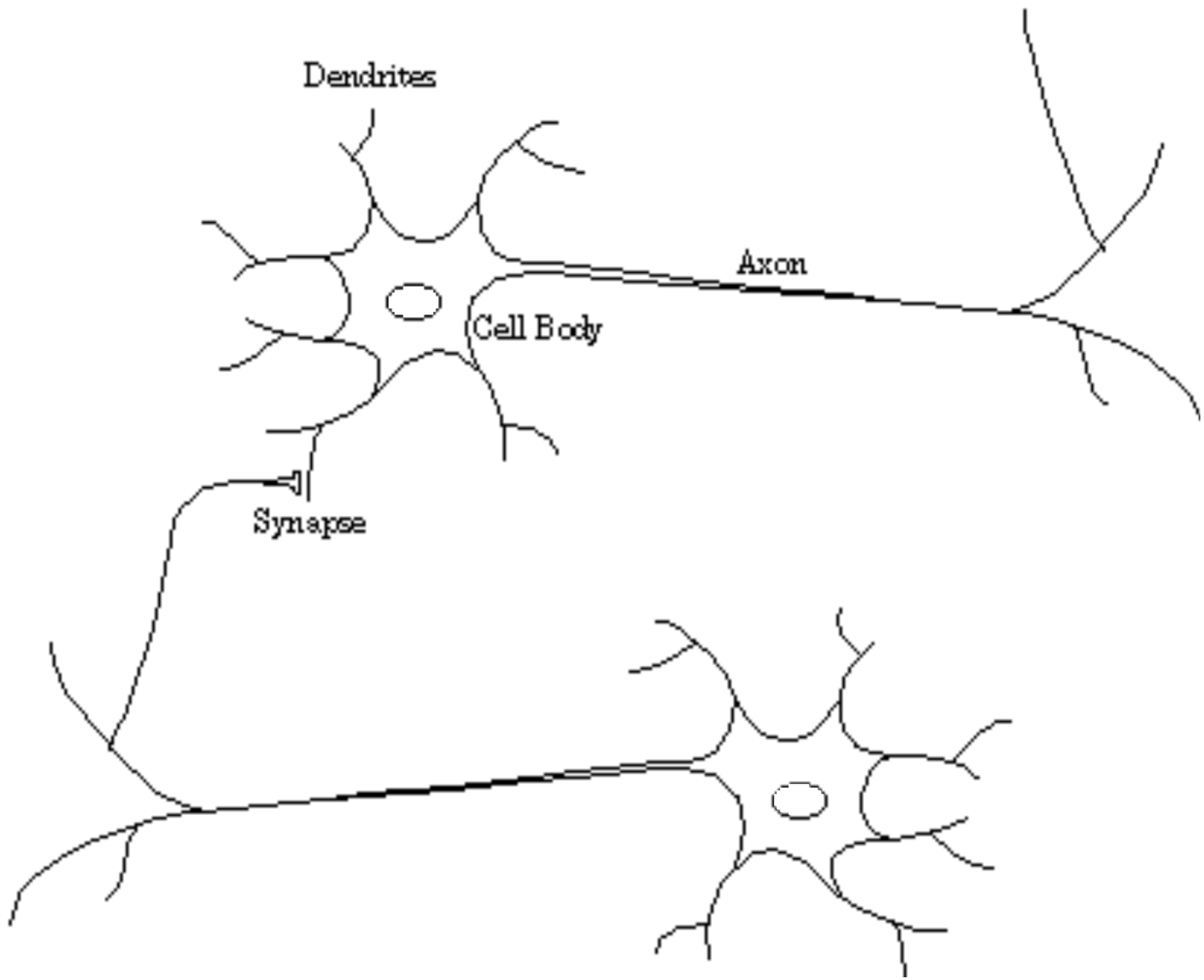
Outline

- Biological Inspirations
- Applications & Properties of ANN
- Perceptron
- Multi-Layer Perceptrons
- Error Backpropagation Algorithm
- Remarks on ANN

Biological Inspirations

- Humans perform complex tasks like vision, motor control, or language understanding very well
- One way to build intelligent machines is to try to imitate the (organizational principles of) human brain

Biological Neuron

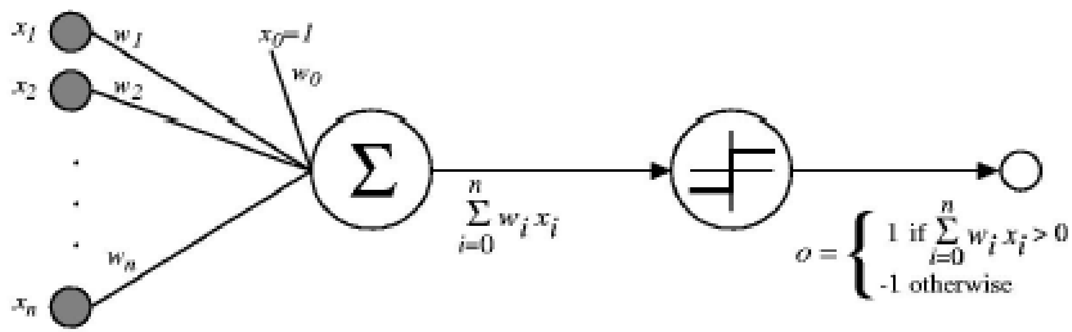


Artificial Neural Networks

- ANNs have been widely used in various domains for:
  - Pattern recognition
  - Function approximation
  - Etc.

Perceptron (Artificial Neuron)

- A perceptron
  - takes a vector of real-valued inputs
  - calculates a linear combination of the inputs
  - outputs +1 if the result is greater than some threshold and -1 (or 0) otherwise



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Perceptron

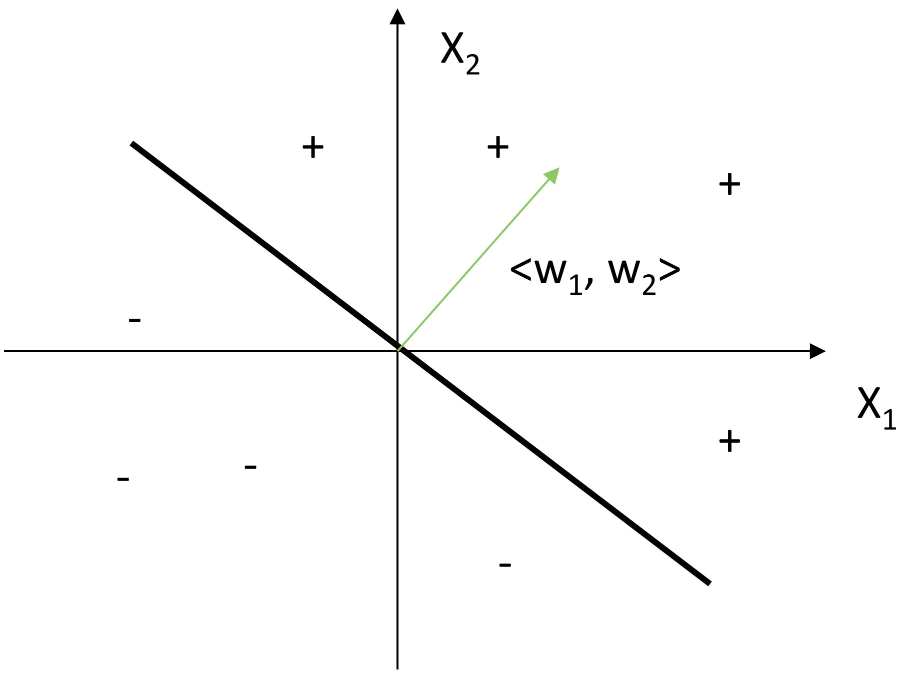
- To simplify notation, assume an additional constant input x_0=1. We can write the perceptron function as

$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

$$sgn(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise} \end{cases}$$

Representational Power of Perceptron

- The perceptron ~ a hyperplane decision surface in the n-dimensional space of instances

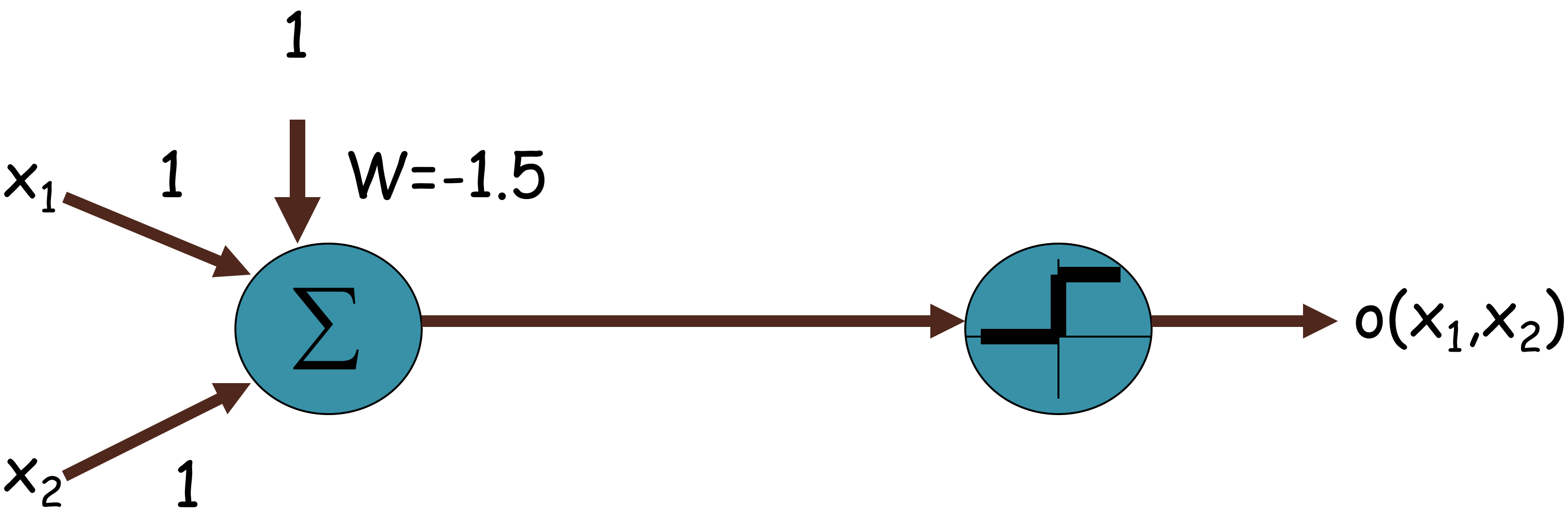


Linearly separable data

Boolean Functions

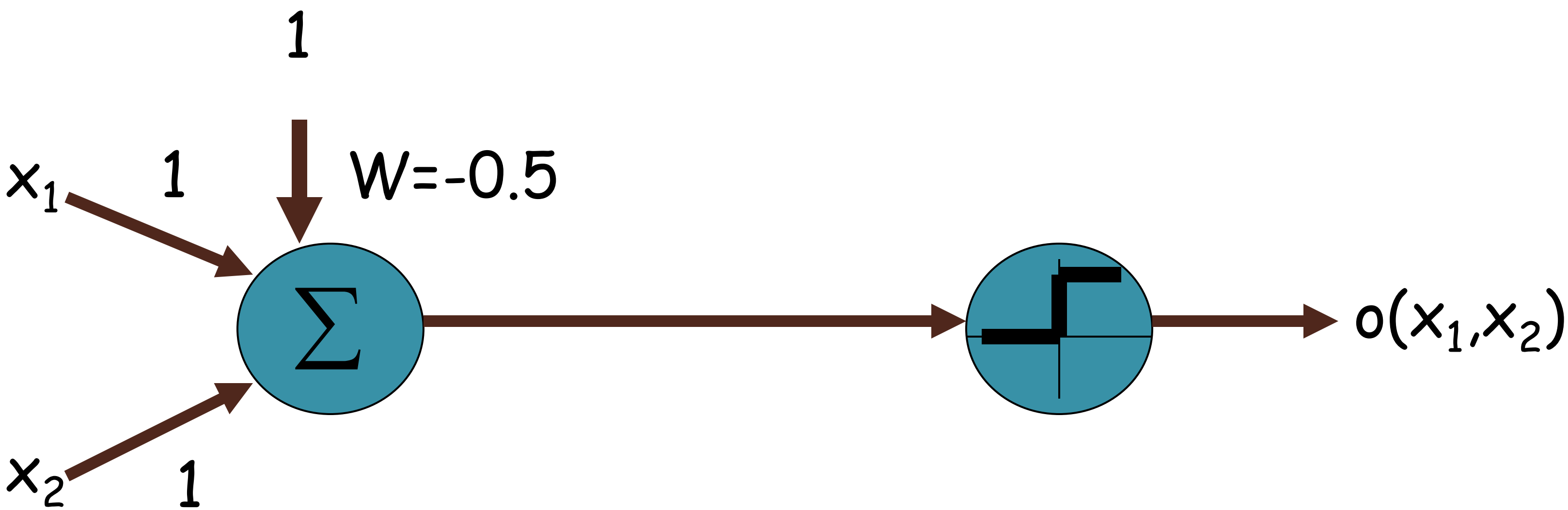
- A single perceptron can be used to represent many boolean functions
  - 1 (true); 0 (false)
- Perceptrons can represent all of the primitive boolean functions
  - AND, OR, NOT

Implementing AND



$$o(x_1, x_2) = 1 \text{ if } -1.5 + x_1 + x_2 > 0$$
$$= 0 \text{ otherwise}$$

Implementing OR



$o(x1,x2) = 1$ if $-0.5 + x1 + x2 > 0$

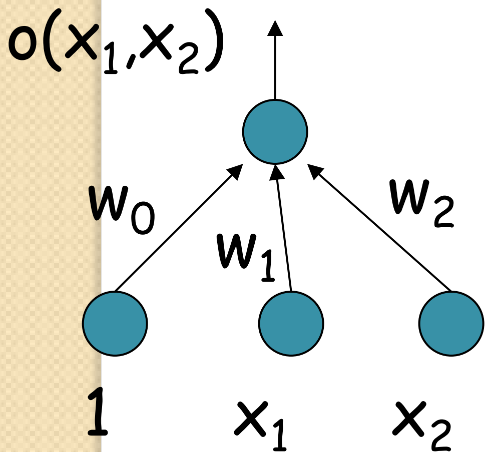
$= 0$ otherwise

Implementing NOT

$$o(x_1) = 1 \text{ if } 0.5 - x_1 > 0$$
$$= 0 \text{ otherwise}$$

The XOR Function

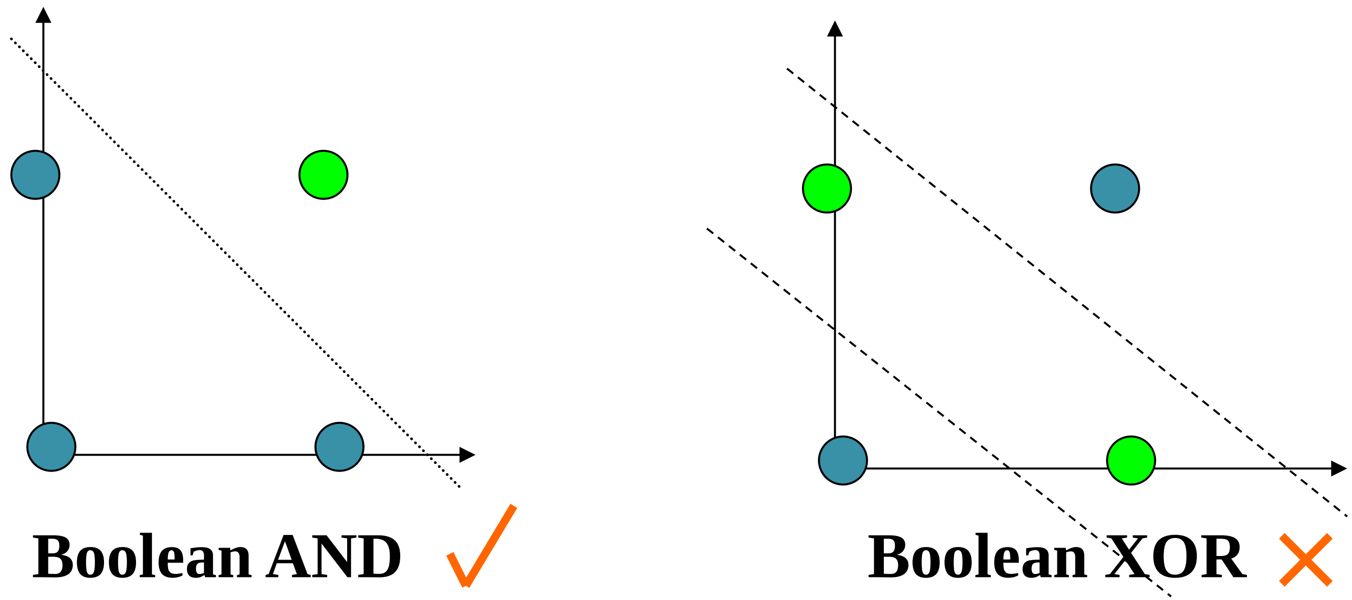
- Unfortunately, some Boolean functions cannot be represented by a single perceptron



$$\left.\begin{array}{l} w_0 + 0\cdot w_1 + 0\cdot w_2 \le 0 \\ w_0 + 0\cdot w_1 + 1\cdot w_2 > 0 \\ w_0 + 1\cdot w_1 + 0\cdot w_2 > 0 \\ w_0 + 1\cdot w_1 + 1\cdot w_2 \le 0 \end{array}\right\} \text{XOR}(x_1,x_2)$$

There is no assignment of values to w_0,w_1 and w_2 that satisfies above inequalities. XOR cannot be represented!

Linear Seprability

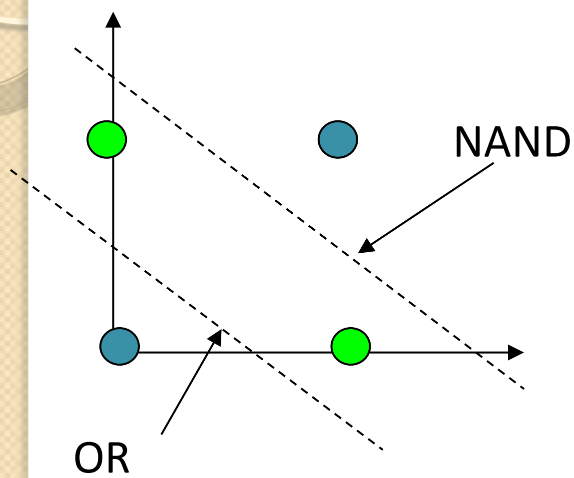


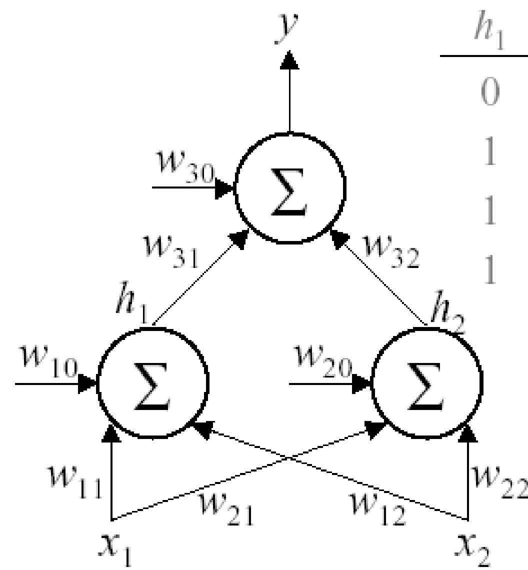
<u>Representation Theorem</u>: Perceptrons can only represent linearly separable functions. That is, the decision surface separating the output values has to be a plane. (Minsky & Papert, 1969)

Remarks on perceptron

- Perceptrons can represent all the primitive Boolean functions
  - AND, OR, and NOT
- Some Boolean functions cannot be represented by a single perceptron
  - Such as the XOR function
- Every Boolean function can be represented by some combination of
  - AND, OR, and NOT
- We want networks of the perceptrons…

Implementing XOR by Multi-layer perceptron (MLP)





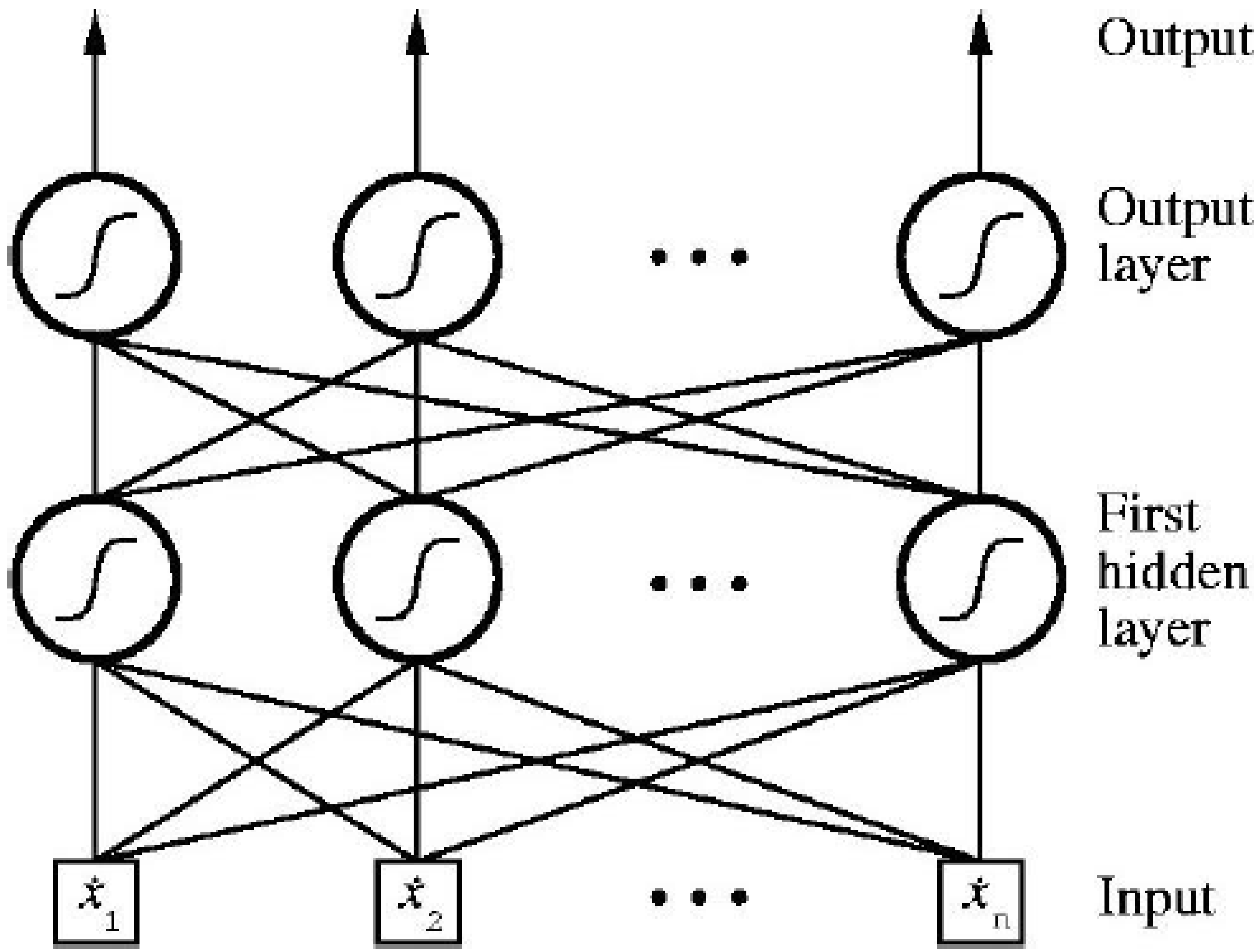
h_1	w_{31}	h_2	w_{32}	Σ	w_{30}	y
0	1	0	-1	0	0.5	0
1	1	0	-1	1	0.5	1
1	1	0	-1	1	0.5	1
1	1	1	-1	0	0.5	0

x_1	w_{11}	x_2	w_{12}	Σ	w_{10}	h_1
0	0.5	0	0.5	0	0.3	0
0	0.5	1	0.5	0.5	0.3	1
1	0.5	0	0.5	0.5	0.3	1
1	0.5	1	0.5	1	0.3	1

x_1	w_{21}	x_2	w_{22}	Σ	w_{20}	h_2
0	0.5	0	0.5	0	0.7	0
0	0.5	1	0.5	0.5	0.7	0
1	0.5	0	0.5	0.5	0.7	0
1	0.5	1	0.5	1	0.7	1

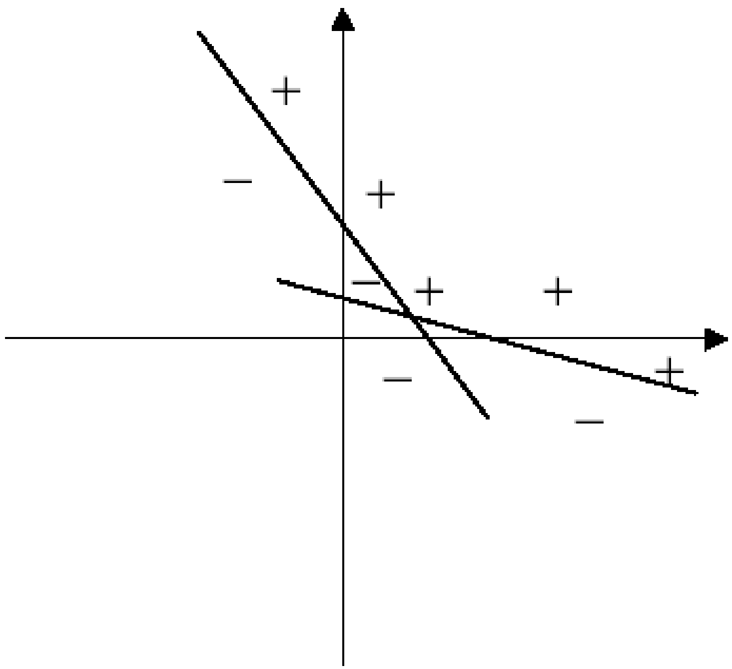
x1 XOR x2 = (x1 OR x2) AND (NOT(x1 AND x2))

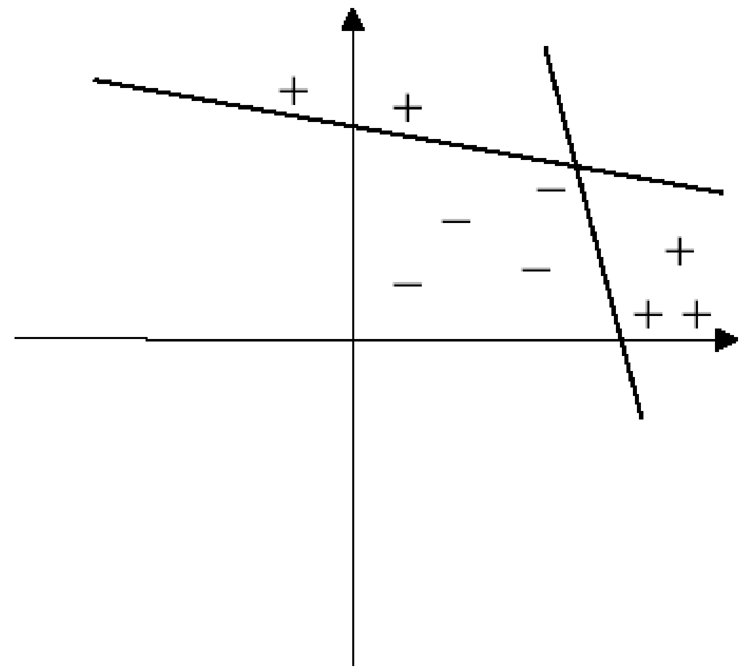
Multi-Layer Perceptrons (MLP)



Representation Power of MLP

- Conjunction of piece-wise hyperplanes

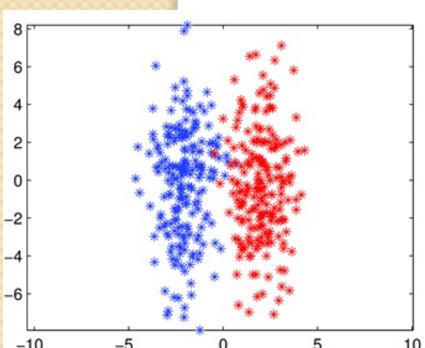




Representation Power of ANN

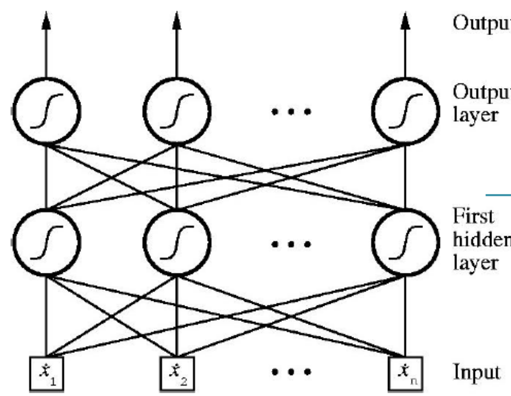
- Boolean functions: Every Boolean function can be represented exactly by some network with two layers of units
- Continuous functions: Every bounded continuous function can be approximated with arbitrarily small error (under a finite norm) by a network with two layers of units
- Arbitrary functions: Any function can be approximated to arbitrary accuracy by a network with three layers of units

Neuron network design for non-closed form problem



Training data $\{x_i, y_i\}$

Training process



Neuron network

Testing data x^*

Testing process

Result y^*

Definition of Training Error

- Training error E: a function of weight vector over the training data set D

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

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

Unthresholded perceptron or linear unit

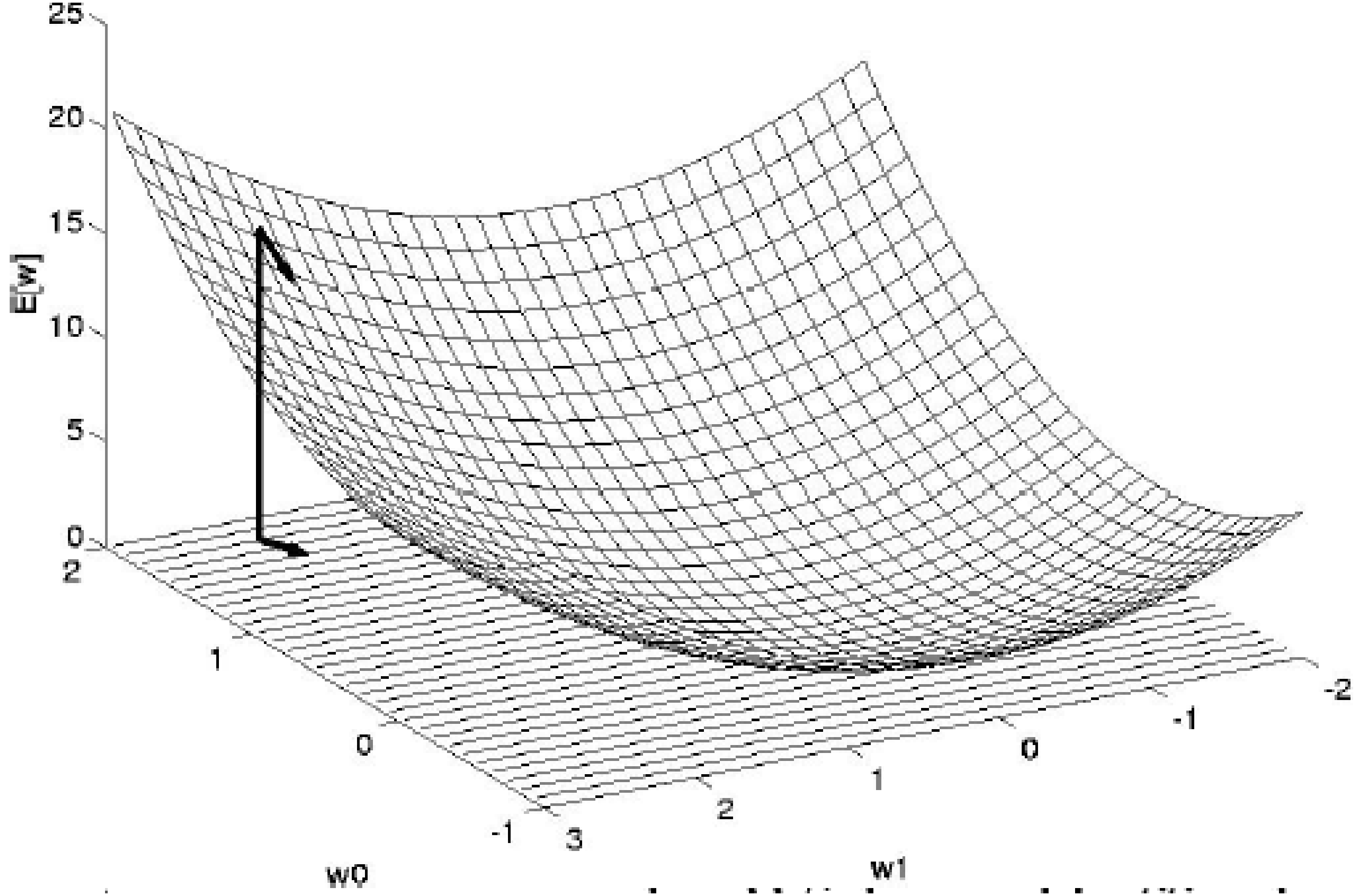
Gradient Descent

- To reduce error E, update the weight vector w in the direction of steepest descent along the error surface

$$\nabla E(w) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \cdots, \frac{\partial E}{\partial w_n} \right]$$

$$w \leftarrow w + (-\eta \nabla E(w))$$

Gradient Descent



Weight Update Rule

$$w \leftarrow w + (-\eta \nabla E(w))$$

$$w_i \leftarrow w_i + (-\eta \frac{\partial E}{\partial w_i}),$$

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

Gradient Descent Search Algorithm

repeat

  $\Delta w \leftarrow 0$

  for each training example <x, t(x)>

    $o(x)=w \cdot x$

    for each w_i

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

  for each wi

    $w_i \leftarrow w_i + \Delta w_i$

until (termination condition)

Perceptron Learning Rule vs Delta Rule

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

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

Perceptron learning rule

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

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

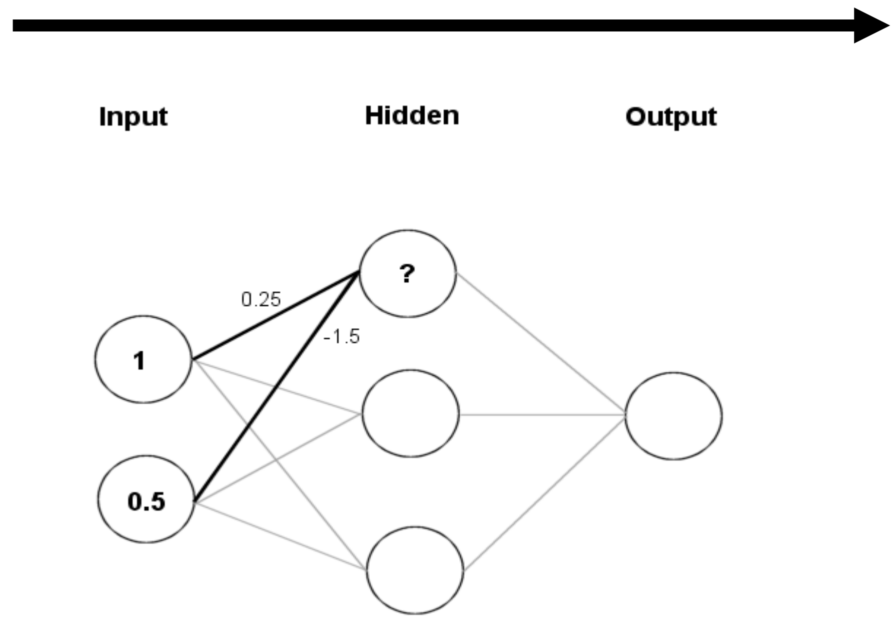
Delta rule

The perceptron learning rule uses the output of the threshold function (either -1 or +1) for learning.
The delta-rule uses the net output without further mapping into output values -1 or +1

Perceptron Learning Algorithm

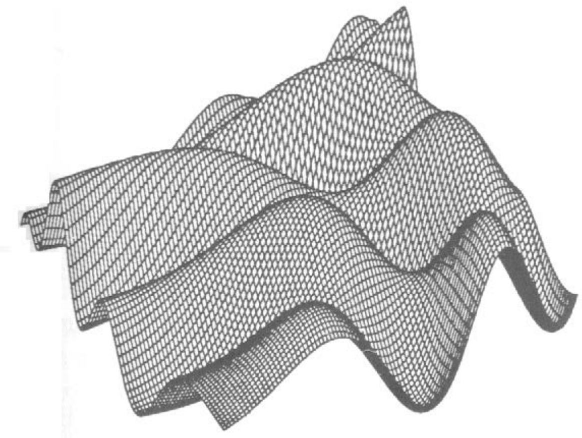
- Guaranteed to converge within a finite time if the training data is linearly separable and η is sufficiently small
- If the data are not linearly separable, convergence is not assured

Feed-forward Networks



Definition of Error for MLP

$$E(w) \equiv \frac{1}{2}\sum_{d \in D}\sum_{i}\left(t_i^{(d)} - o_i^{(d)}\right)^2$$



$$\nabla E(w) \equiv \left[\frac{\partial E}{\partial w_{11}^{(o)}}, \frac{\partial E}{\partial w_{12}^{(o)}}, \cdots, \frac{\partial E}{\partial w_{11}^{(h)}}, \frac{\partial E}{\partial w_{12}^{(h)}}, \cdots, \frac{\partial E}{\partial w_{ij}^{(h)}}\right]$$

$$w \leftarrow w + \left(-\eta \nabla E(w)\right)$$

Output Layer's Weight Update

$$\frac{\partial E^{(d)}}{\partial w_{ij}^{(d)}} = \frac{\partial \frac{1}{2}\sum_l (t_l - o_l)^2}{\partial w_{ij}}$$

$$= \frac{\partial \frac{1}{2} \sum_l \left(t_l - \Theta(\sum_m w_{lm} h_m) \right)^2}{\partial w_{ij}}$$

$$= \frac{\partial \frac{1}{2} \left(t_i - \Theta\left(\sum_m w_{im} h_m \right) \right)^2}{\partial w_{ij}}$$

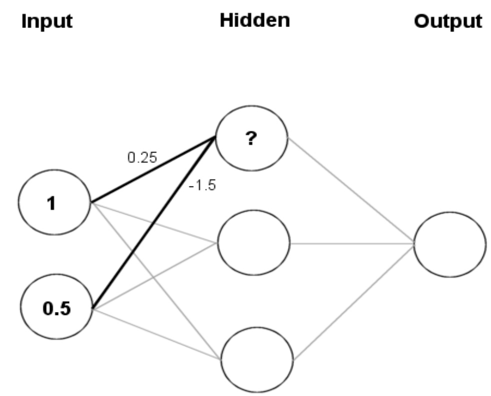
$$= \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial \sigma_i} \frac{\partial \sigma_i}{\partial w_{ij}} \qquad o_i = \Theta(\sigma_i) = \frac{1}{1+e^{-\sigma_i}} \text{ and } \sigma_i = \sum_m w_{im} h_m$$

$$= \frac{\partial \left[\frac{1}{2} (t_i - o_i)^2 \right]}{\partial o_i} \frac{\partial \left[\frac{1}{1+e^{-\sigma_i}} \right]}{\partial \sigma_i} \frac{\partial \left[\sum_m w_{im} h_m \right]}{\partial w_{ij}}$$

$$= -(t_i - o_i) o_i (1 - o_i) h_j$$

Hidden Layer's Weight Update

- Error of $h_j \propto \sum_i \frac{\partial E}{\partial w_{ij}} w_{ij}$
  - Distribute error to inputs proportional to weights



- Similar to output layer:

$$\frac{\partial E}{\partial w_{jk}} = \sum_i \left[-(t_i - o_i) o_i (1 - o_i) w_{ij} \right] h_j (1 - h_j) x_k$$

Error Back-propagation

Error Back Propagation Algorithm

initialize all weights to small random numbers

repeat

for each training example <x, t(x)>

for each hidden node $h_j \leftarrow \Theta(\sum_k w_{jk} x_k)$

for each output node $o_i \leftarrow \Theta(\sum_j w_{ij} h_j)$

for each output node's weight $\partial E / \partial w_{ij} = -o_i(1-o_i)(t_i - o_i)h_j$

for each hidden node's weight $\partial E / \partial w_{jk} = [\sum_i -o_i(1-o_i)(t_i - o_i)w_{ij}]h_j(1-h_j)x_k$

for each hidden node's weight $w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$

for each output node's weight $w_{jk} \leftarrow w_{jk} - \eta \frac{\partial E}{\partial w_{jk}}$

until (termination condition)

Generalization, Overfitting, etc.

- Artificial neural networks with a large number of weights tend to overfit the training data
- To increase generalization accuracy, use a validation set
  - Find the optimal number of perceptrons
  - Find the optimal number of training iterations
    - Stop when overfitting happens

Generalization, Overfitting, etc.