COMP3411/9414: Artificial Intelligence

Module 3

Perceptrons

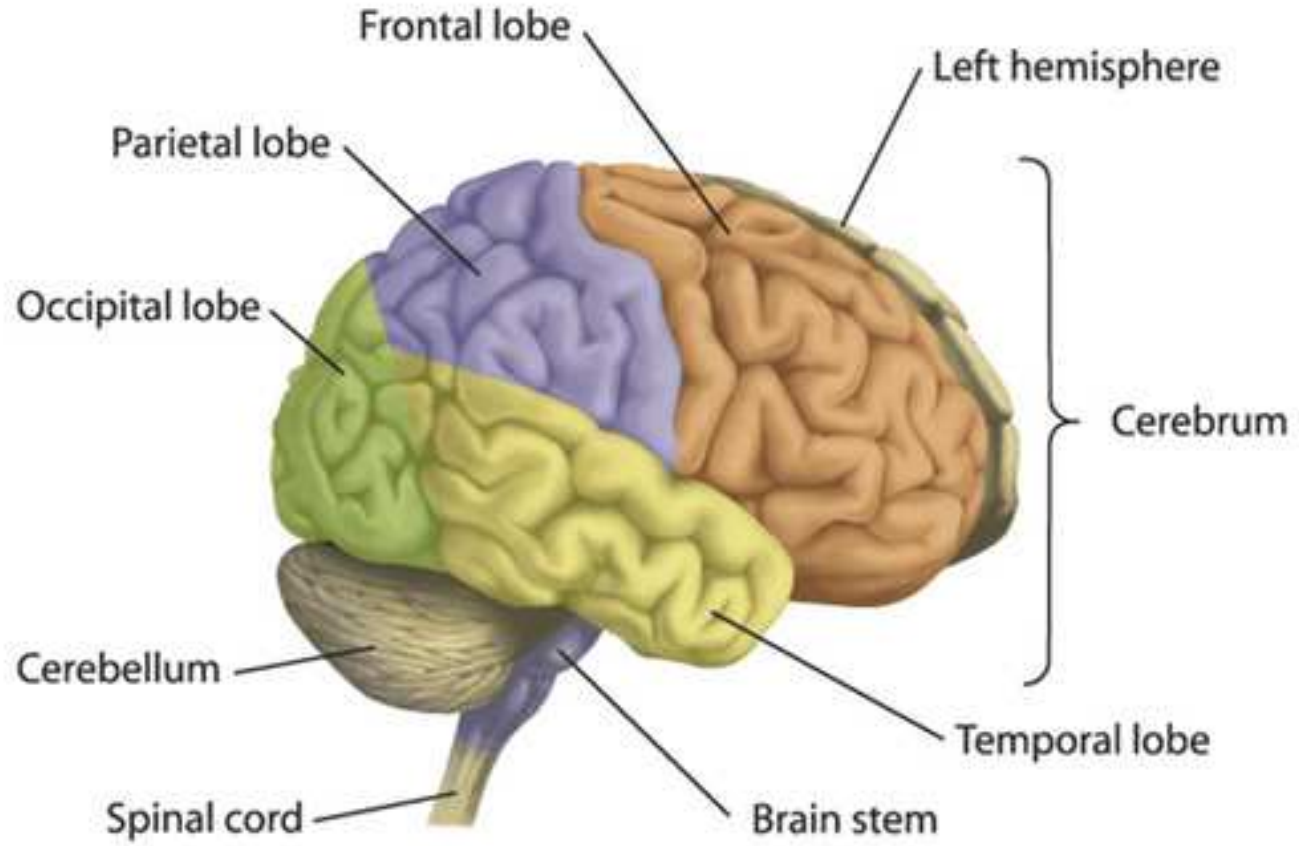
Russell & Norvig, Chapter 18.6, 18.7

Outline

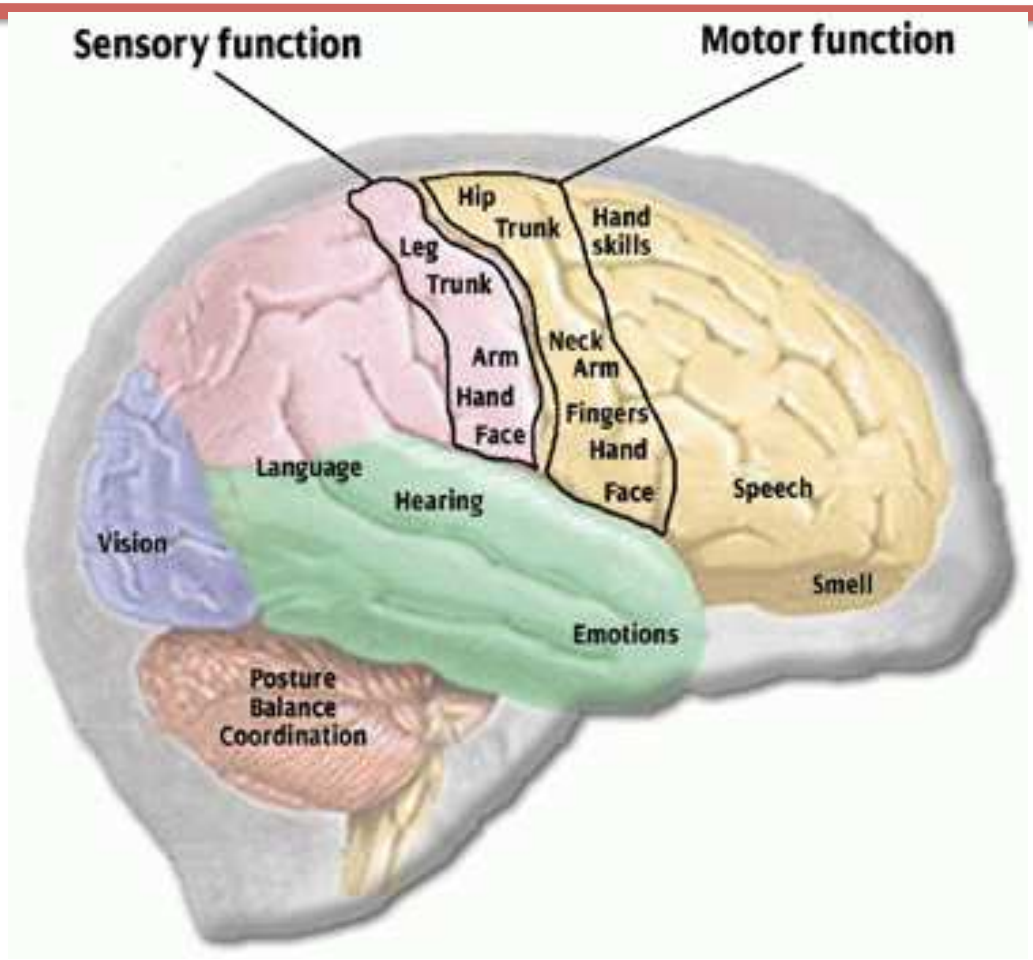
- Neurons – Biological and Artificial
- Perceptron Learning
- Linear Separability
- Multi-Layer Networks

Sub-Symbolic Processing

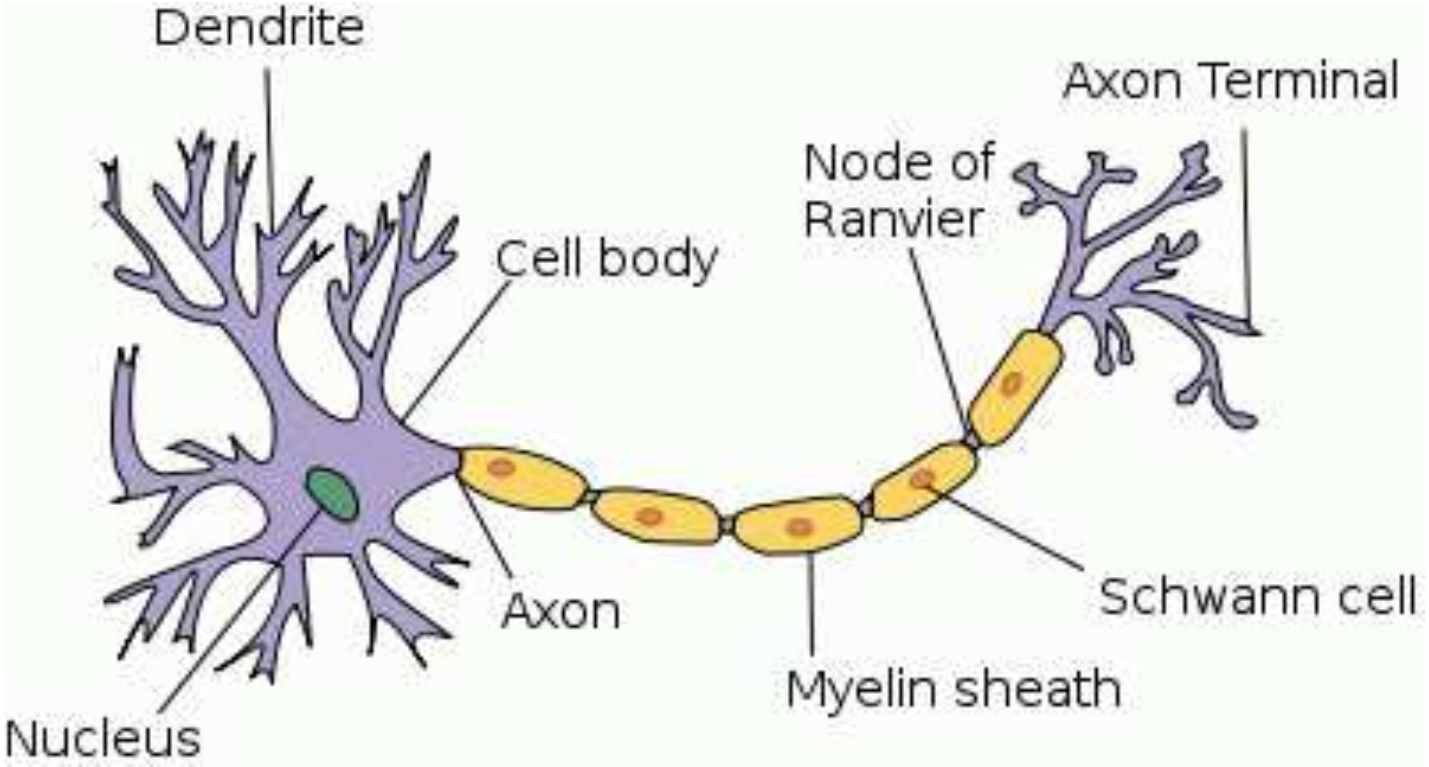
Brain Regions



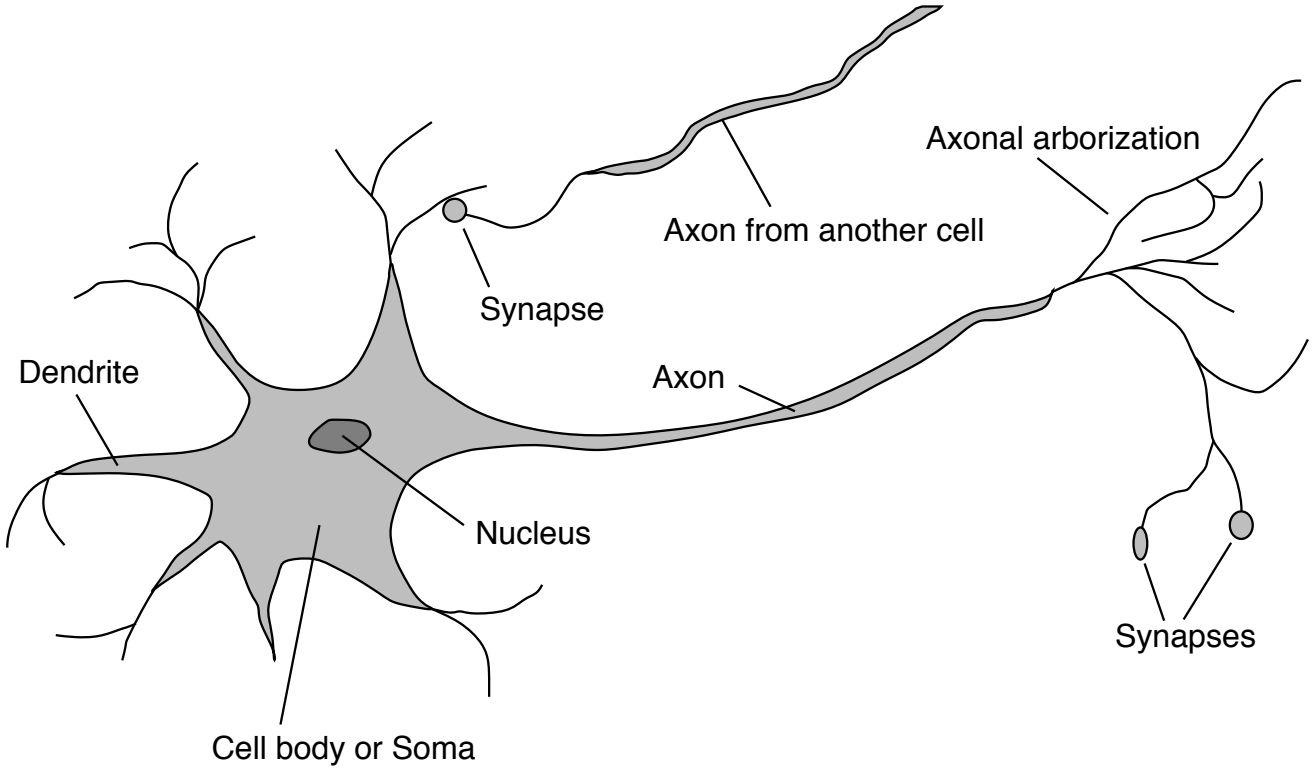
Brain Functions



Structure of a Typical Neuron



Brains



10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential

Biological Neurons

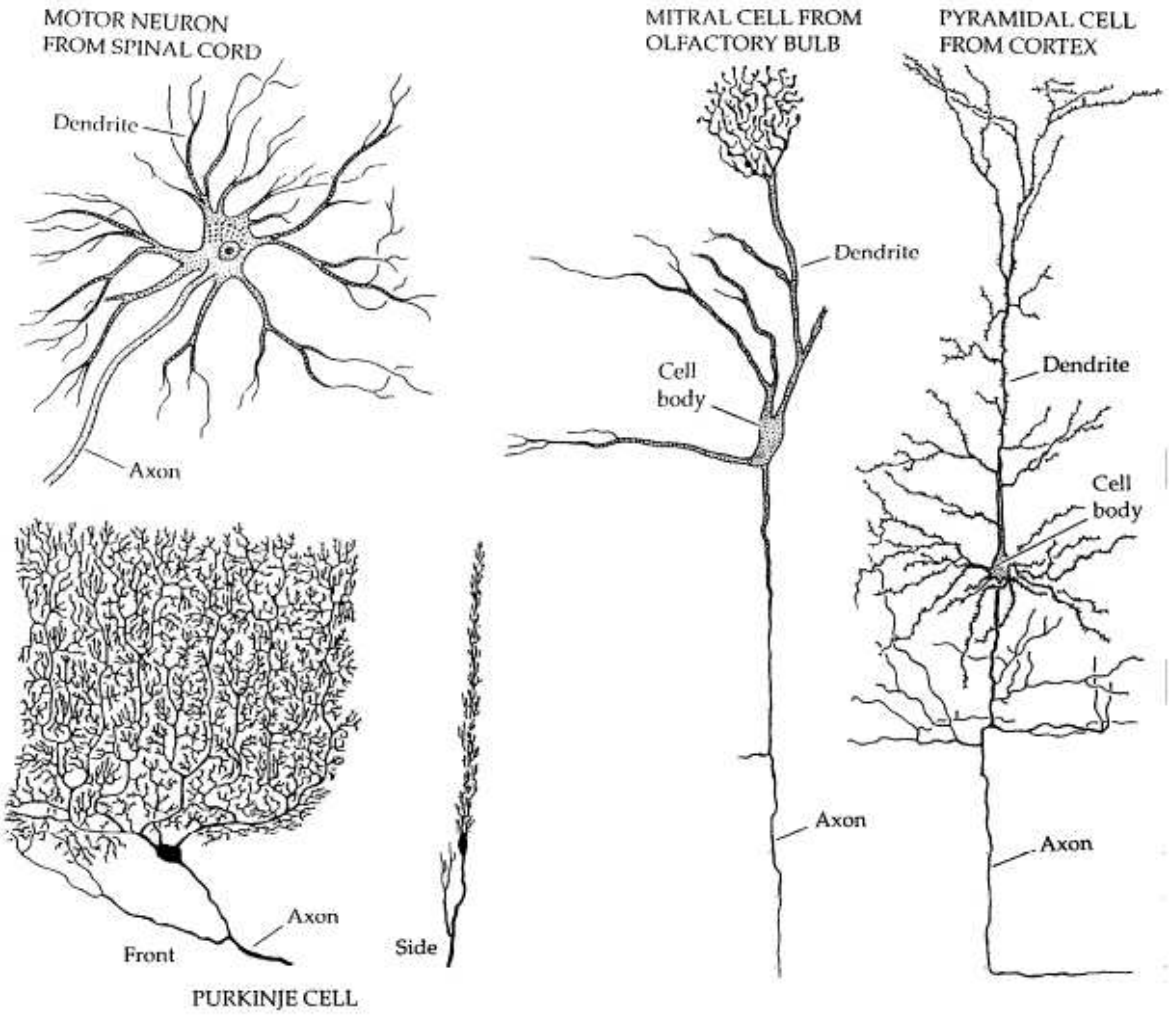
The brain is made up of neurons (nerve cells) which have a cell body (soma)

- dendrites (inputs)
- an axon (outputs)
- synapses (connections between cells)

Synapses can be exitatory or inhibitory and may change over time.

When the inputs reach some threshhold an action potential (electrical pulse) is sent along the axon to the outputs.

Variety of Neuron Types



The Big Picture

Human brain has 100 billion neurons with an average of 10,000 synapses each

Latency is about 3-6 milliseconds

At most a few hundred “steps” in any mental computation, but massively parallel

Artificial Neural Networks

(Artificial) Neural Networks are made up of nodes which have

- inputs edges, each with some weight
- outputs edges (with weights)
- an activation level (a function of the inputs)

- Weights can be positive or negative and may change over time (learning).
- The input function is the weighted sum of the activation levels of inputs.
- The activation level is a non-linear transfer function g of this input:

$$\text{activation}_i = g(s_i) = g(\sum_j w_{ij} x_j)$$

- Some nodes are inputs (sensing), some are outputs (action)

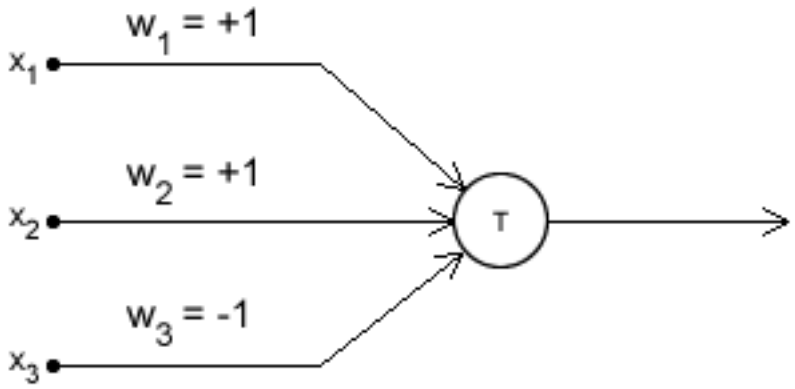
First artificial neurons: The McCulloch-Pitts model

- The McCulloch-Pitts model was an extremely simple artificial neuron.
  - The inputs could be either a zero or a one.
  - And the output was a zero or a one.
  - And each input could be either excitatory or inhibitory.

- Now the whole point was to sum the inputs.
  - If an input is one, and is excitatory in nature, it added one.
  - If it was one, and was inhibitory, it subtracted one from the sum.
  - This is done for all inputs, and a final sum is calculated.

- Now, if this final sum is less than some value (which you decide, say T = threshold), then the output is zero. Otherwise, the output is a one.

McCulloch & Pitts Model of a Single Neuron

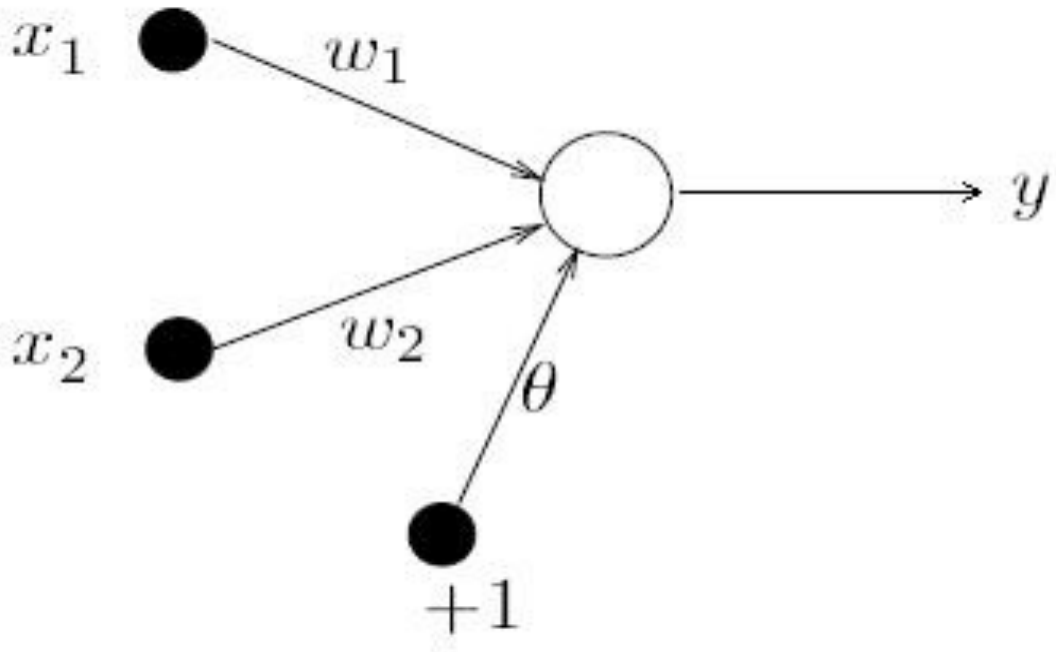


$$sum = x1w1 + x2w2 + x3w3 + \ldots$$

if sum < T or not ?
if sum < T , then the output is made zero.
Otherwise, it is made a one.

Simple Perceptron

The perceptron is a single layer feed-forward neural network.

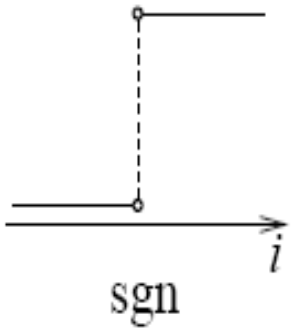


Simple Perceptron

- Simplest output function

$$y = \text{sgn}\left(\sum_{i=1}^{2} w_i x_i + \theta\right)$$

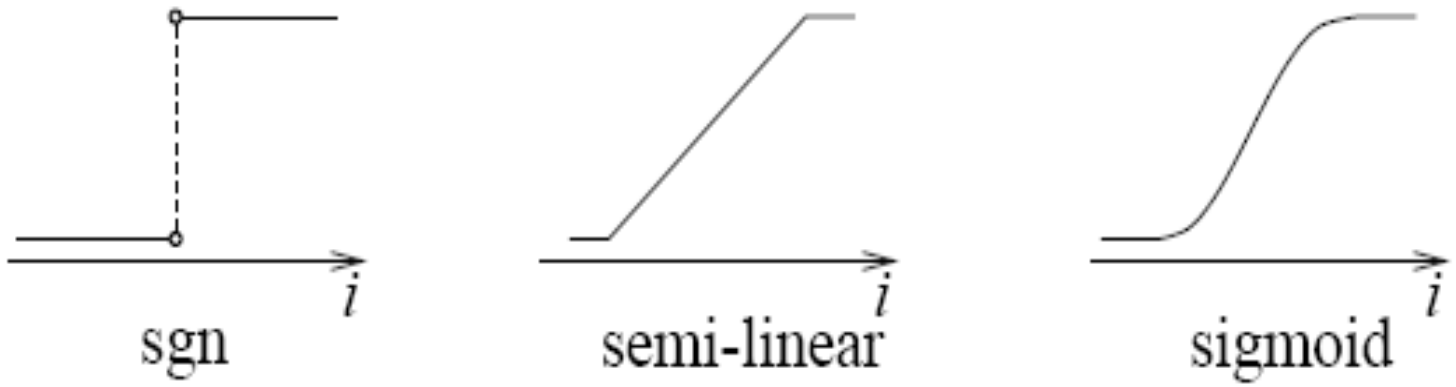
$$\text{sgn}(s) = \begin{cases} 1 & \text{if } s > 0 \\ -1 & \text{otherwise.} \end{cases}$$



- Used to classify patterns said to be linearly separable

Activation Functions

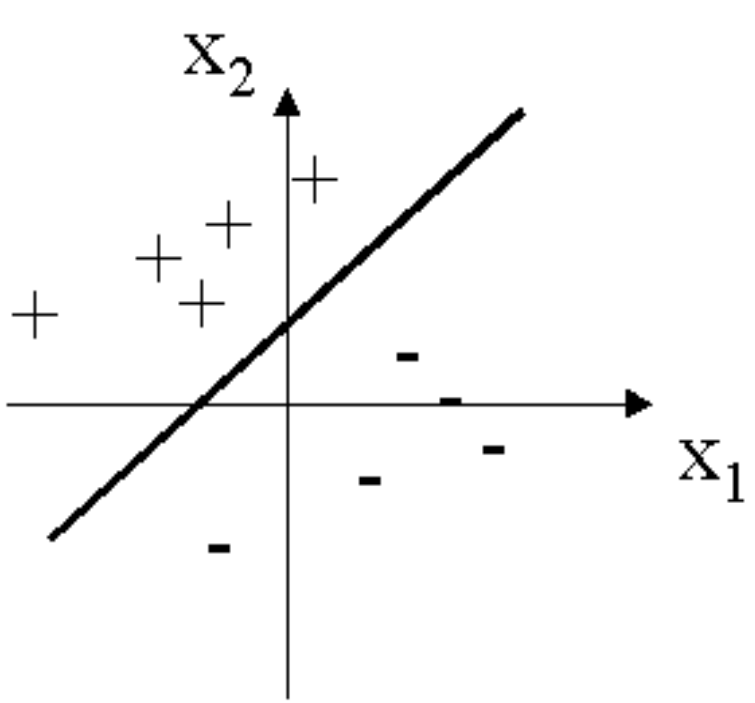
- Function which takes the total input and produces an output for the node given some threshold.

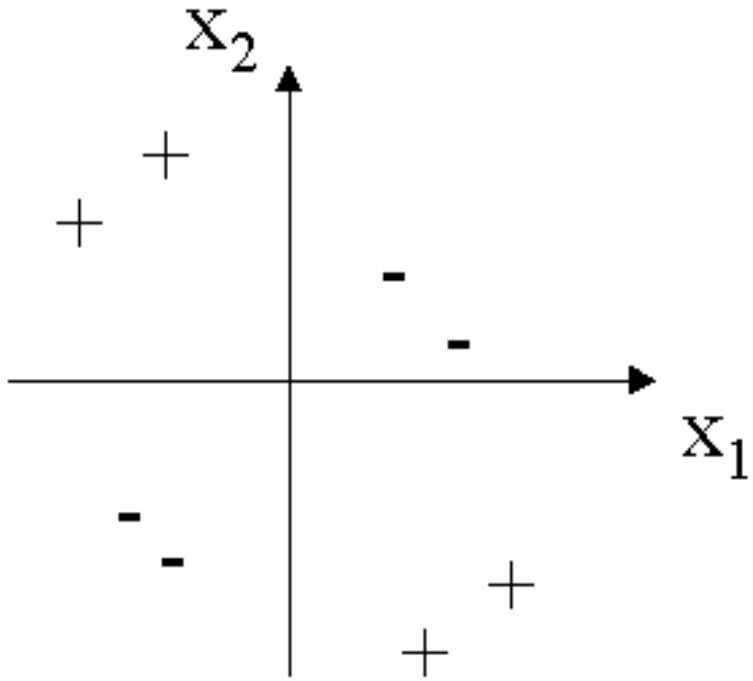


Linear Separability

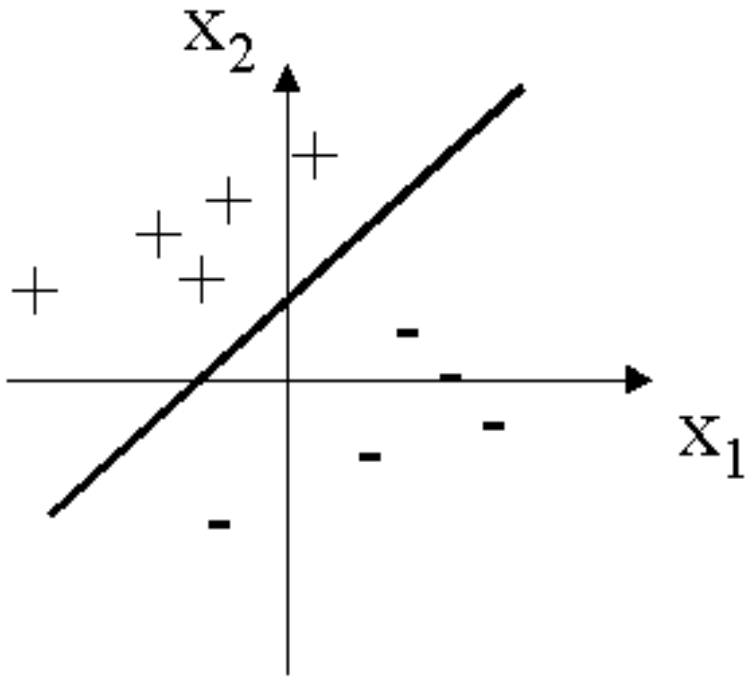
Q: what kind of functions can a perceptron compute?

Linearly Separable

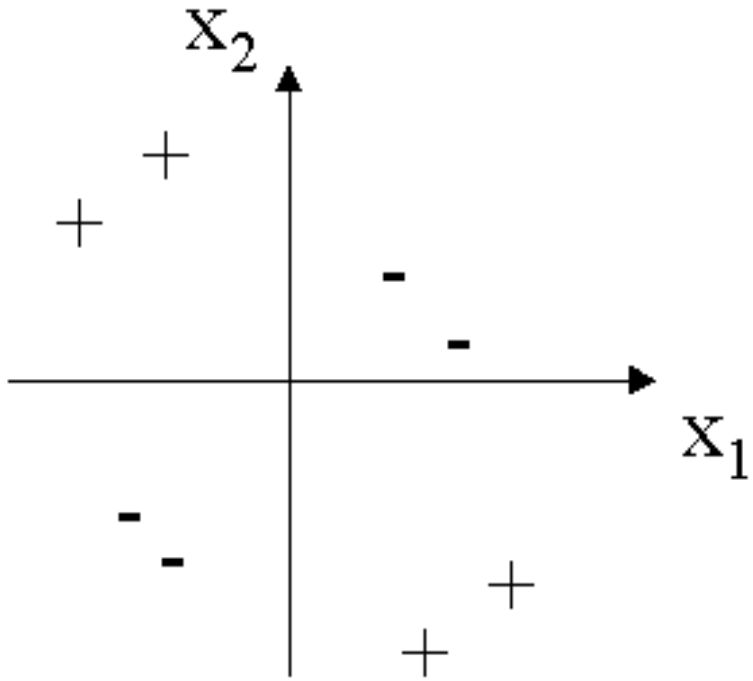




Linearly Separable



Linearly Separable



Not Linearly Separable

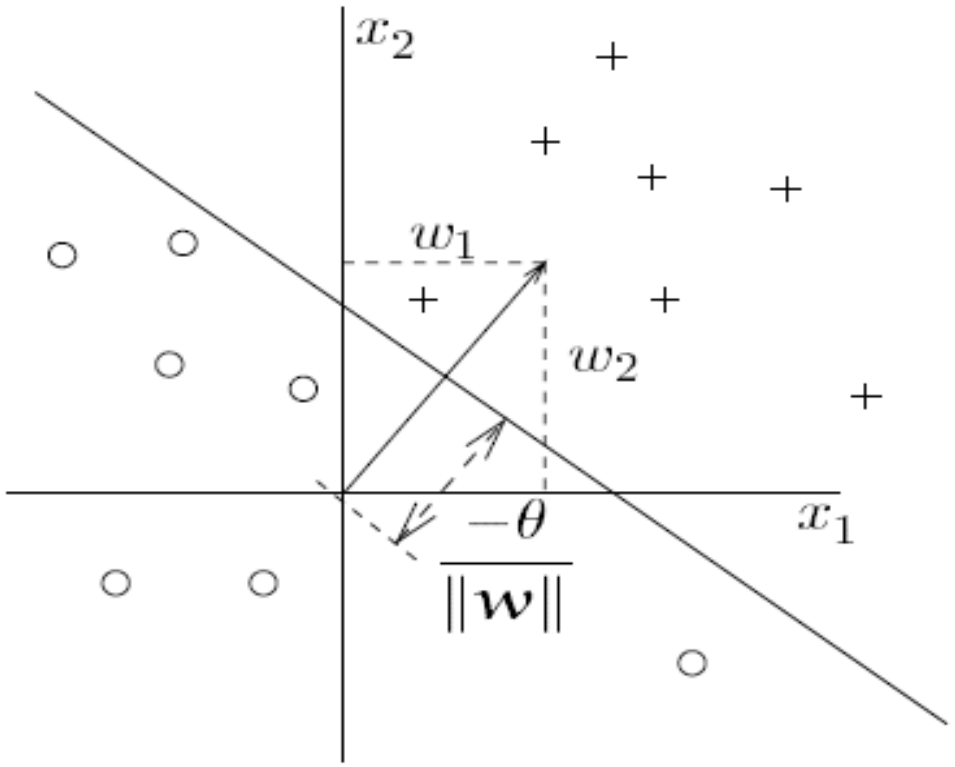
$$w_1 x_1 + w_2 x_2 + \theta = 0$$

Linearly Separable

The bias is proportional to the offset of the plane from the origin

The weights determine the slope of the line

The weight vector is perpendicular to the plane



$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{\theta}{w_2}$$

Perceptron Learning Algorithm

We want to train the perceptron to classify inputs correctly

Accomplished by adjusting the connecting weights and the bias

Can only properly handle linearly separable sets

Perceptron Learning Algorithm

- We have a "training set" which is a set of input vectors used to train the perceptron.
- During training both w_i and θ *(bias)* are modified for convenience, let $w_0 = \theta$ and $x_0 = 1$
- *Let*, η, the *learning rate*, be a small positive number (small steps lessen the possibility of destroying correct classifications)
- Initialise w_i to some values

Perceptron Learning Algorithm

$$\textit{Desired output} \qquad d(n) = \begin{cases} +1 & \textit{if } x(n) \in \textit{set } A \\ -1 & \textit{if } x(n) \in \textit{set } B \end{cases}$$

1. Select random sample from training set as input
2. If classification is correct, do nothing
3. If classification is incorrect, modify the weight vector w using

$$w_i = w_i + \eta d(n) x_i(n)$$

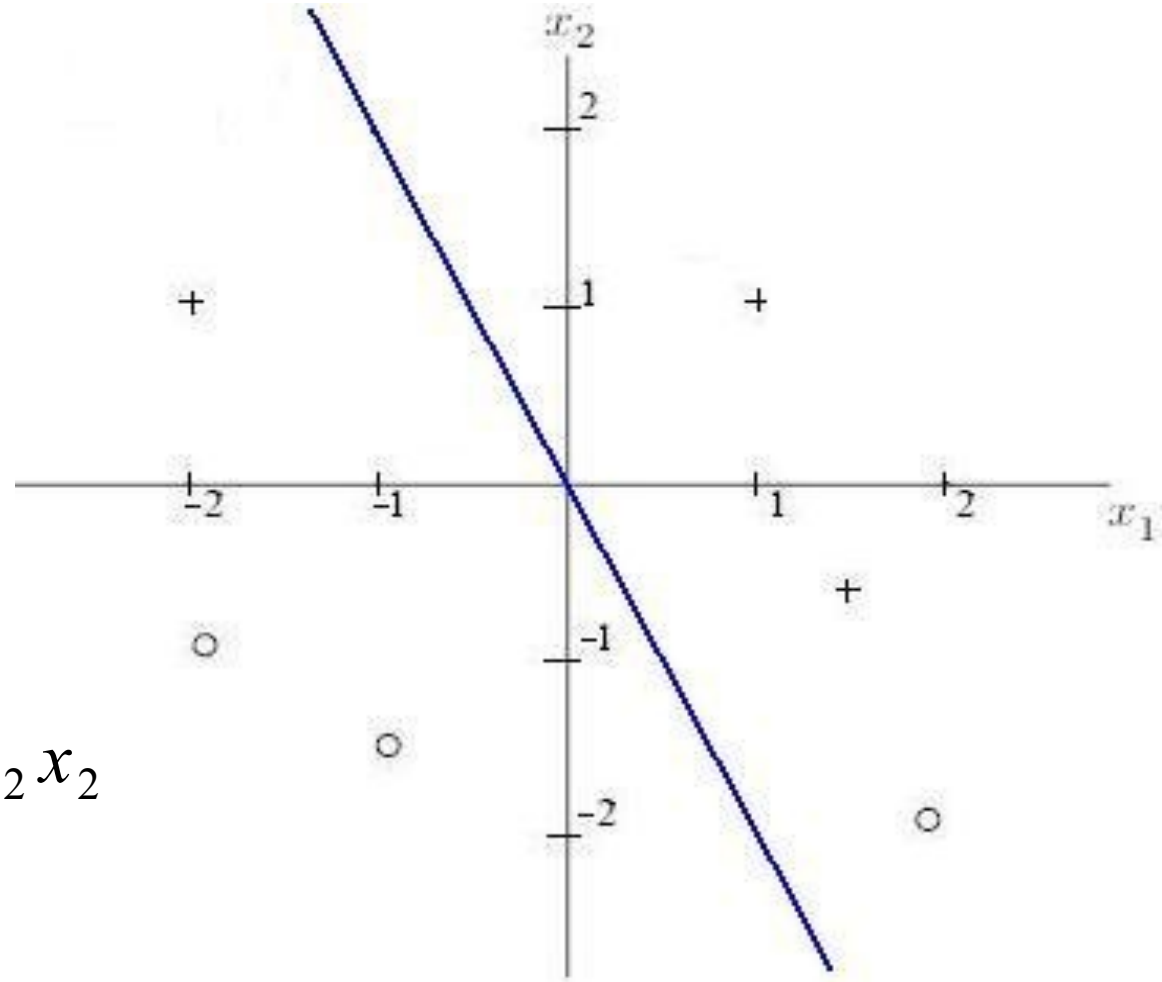
Repeat this procedure until the entire training set is classified correctly

Learning Example

Initial Values:

$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$



$$0 = w_0 + w_1 x_1 + w_2 x_2$$
$$= 0 + x_1 + 0.5x_2$$
$$\Rightarrow \quad x_2 = -2x_1$$

Learning Example

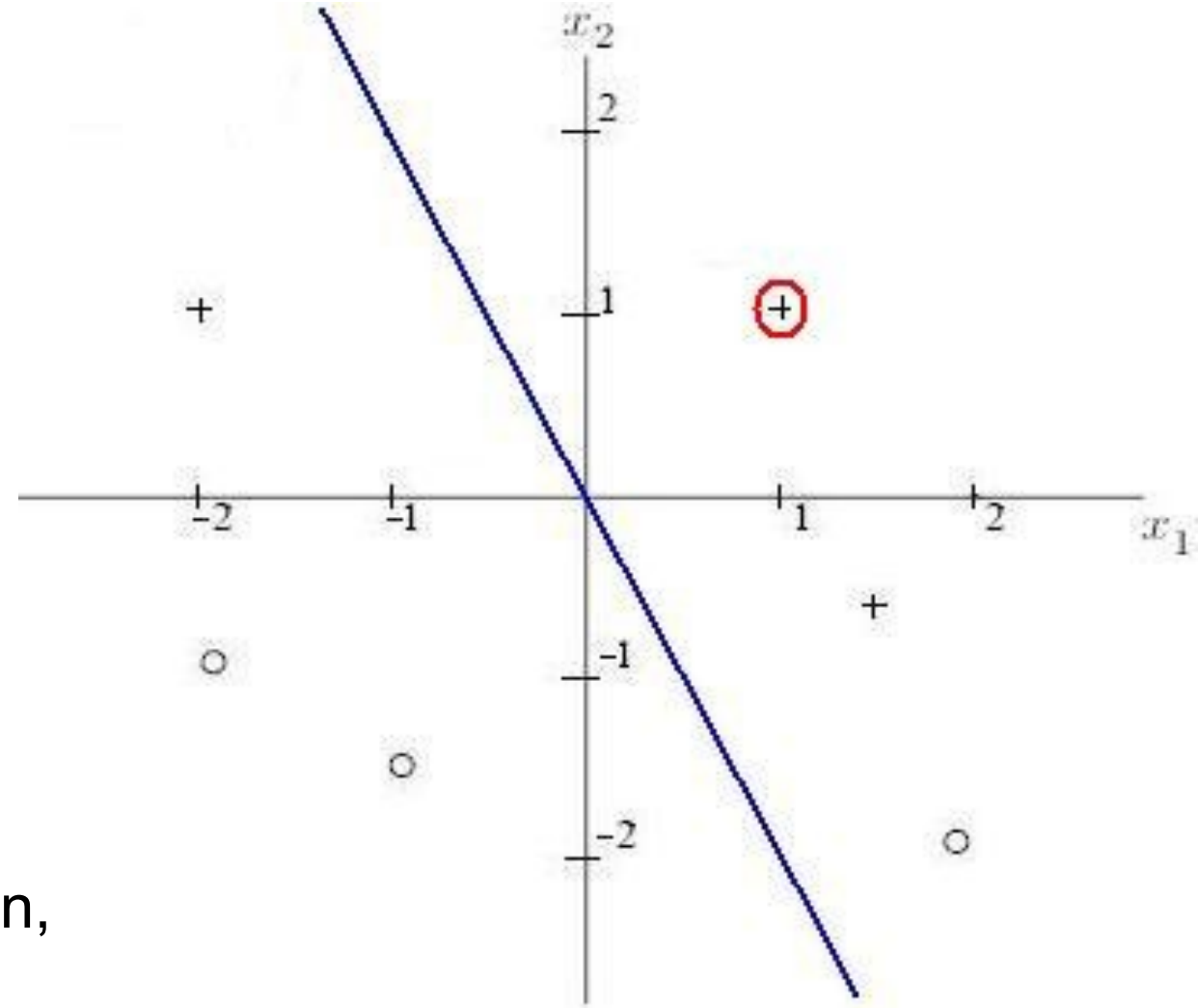
$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$x_1 = 1, x_2 = 1$

$w^T x > 0$

Correct classification, no action



Learning Example

$\eta = 0.2$

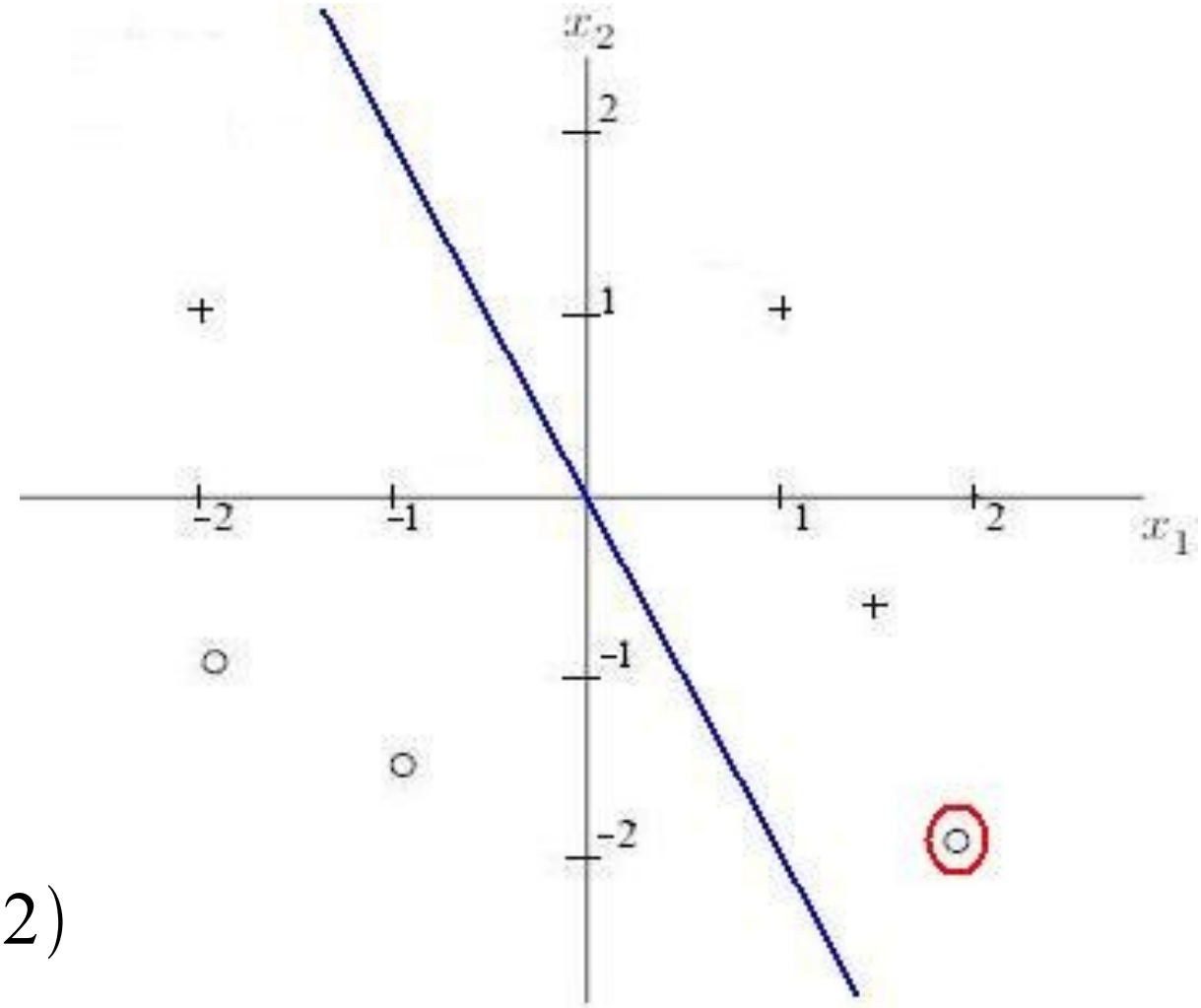
$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$x_1 = 2, x_2 = -2$

$$w_0 = w_0 - 0.2 * 1$$

$$w_1 = w_1 - 0.2 * 2$$

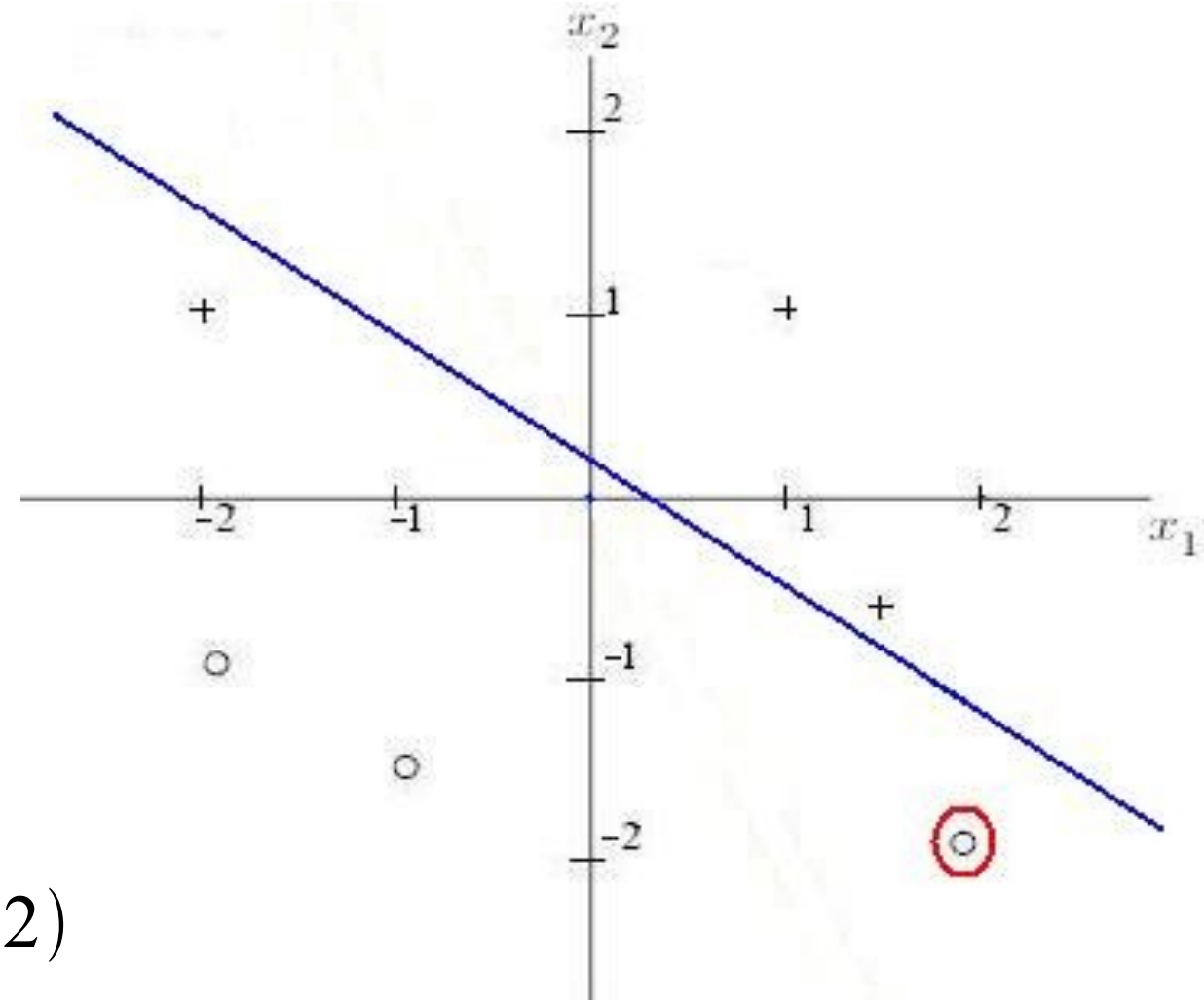
$$w_2 = w_2 - 0.2 * (-2)$$



Learning Example

$\eta = 0.2$

$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$



$x_1 = 2, x_2 = -2$

$$w_0 = w_0 - 0.2 * 1$$
$$w_1 = w_1 - 0.2 * 2$$
$$w_2 = w_2 - 0.2 * (-2)$$

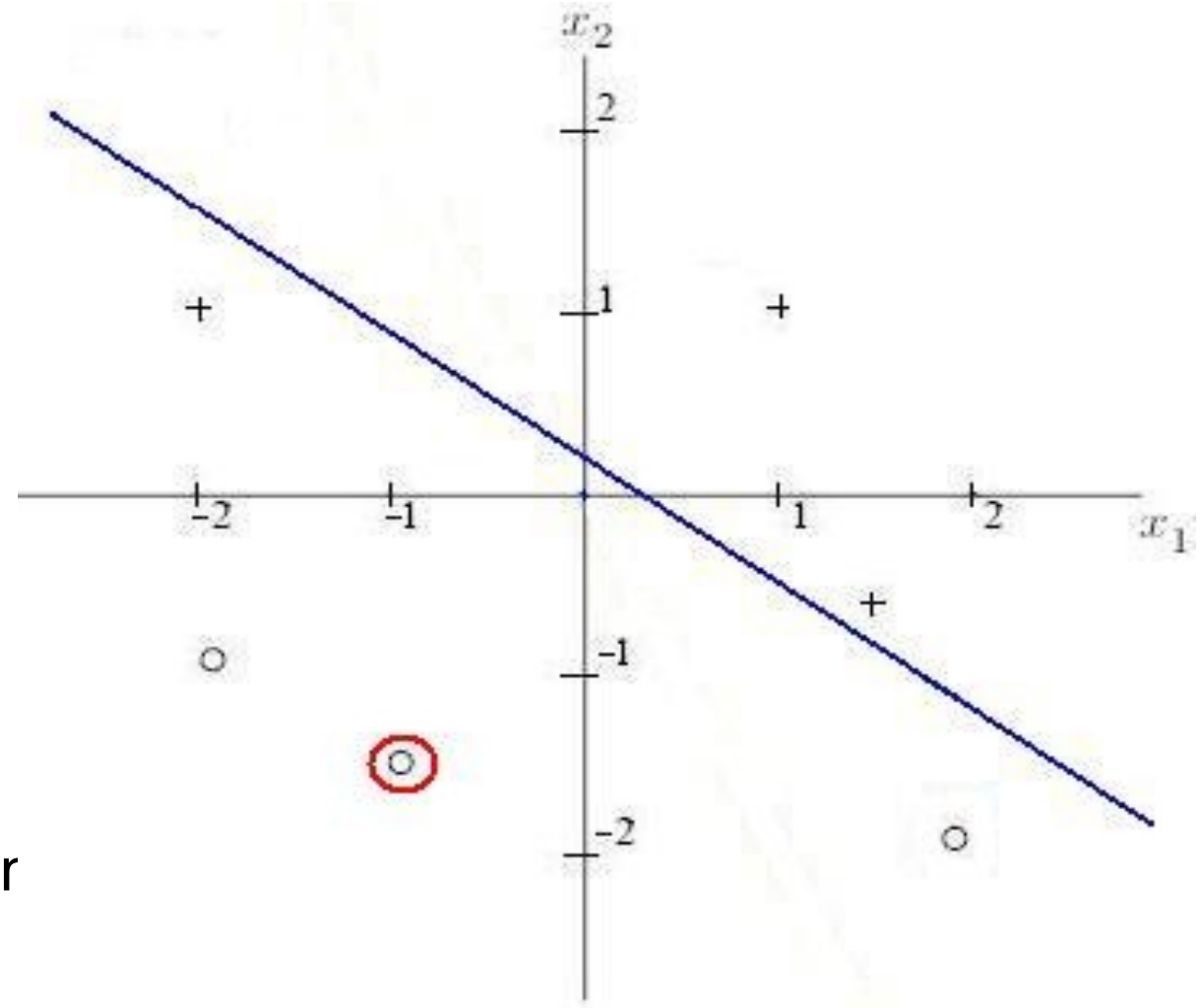
Learning Example

$\eta = 0.2$

$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$x_1 = -1, x_2 = -1.5$
$w^T x < 0$

Correct classificatior
no action



Learning Example

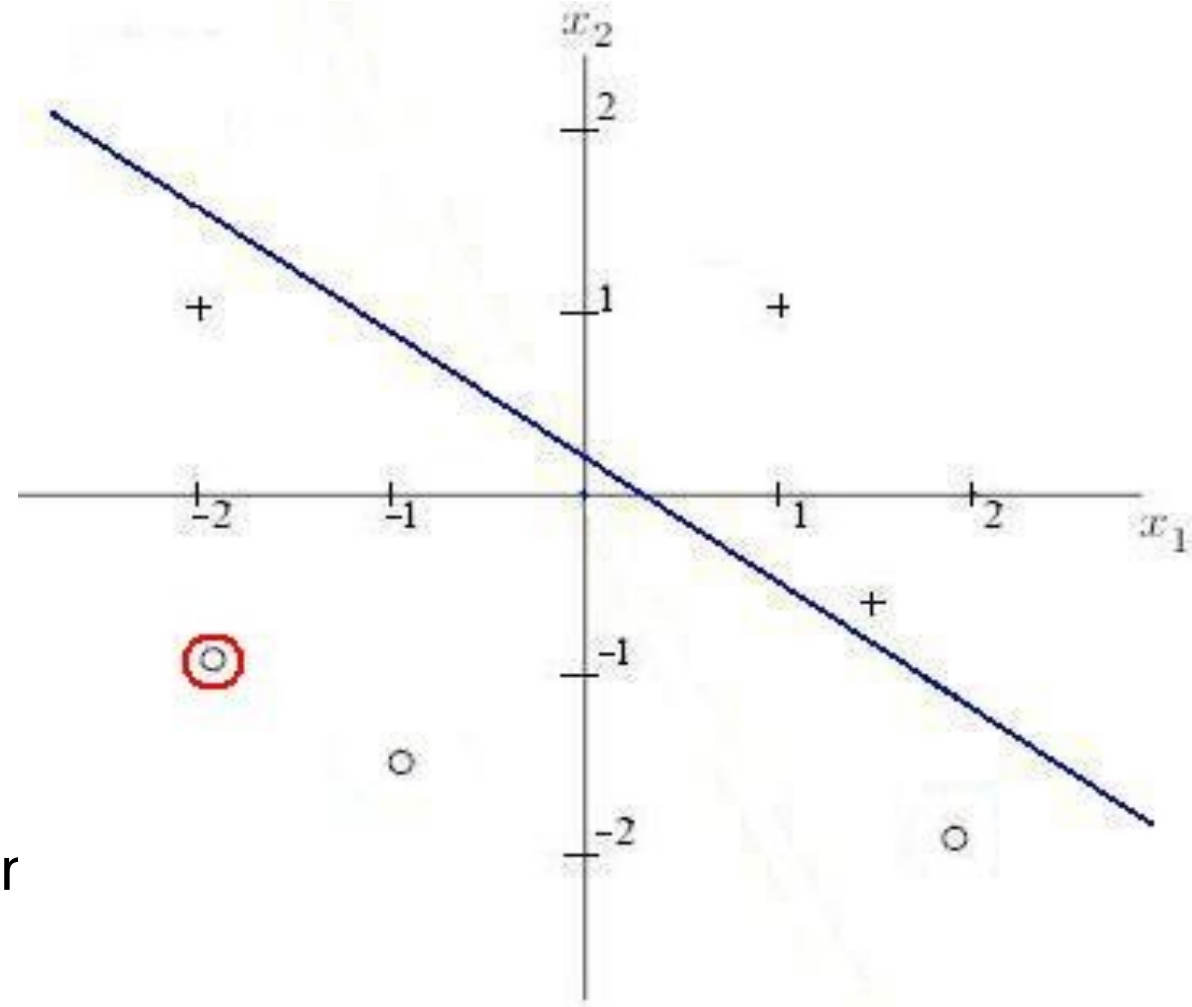
$\eta = 0.2$

$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$x_1 = -2, x_2 = -1$

$w^T x < 0$

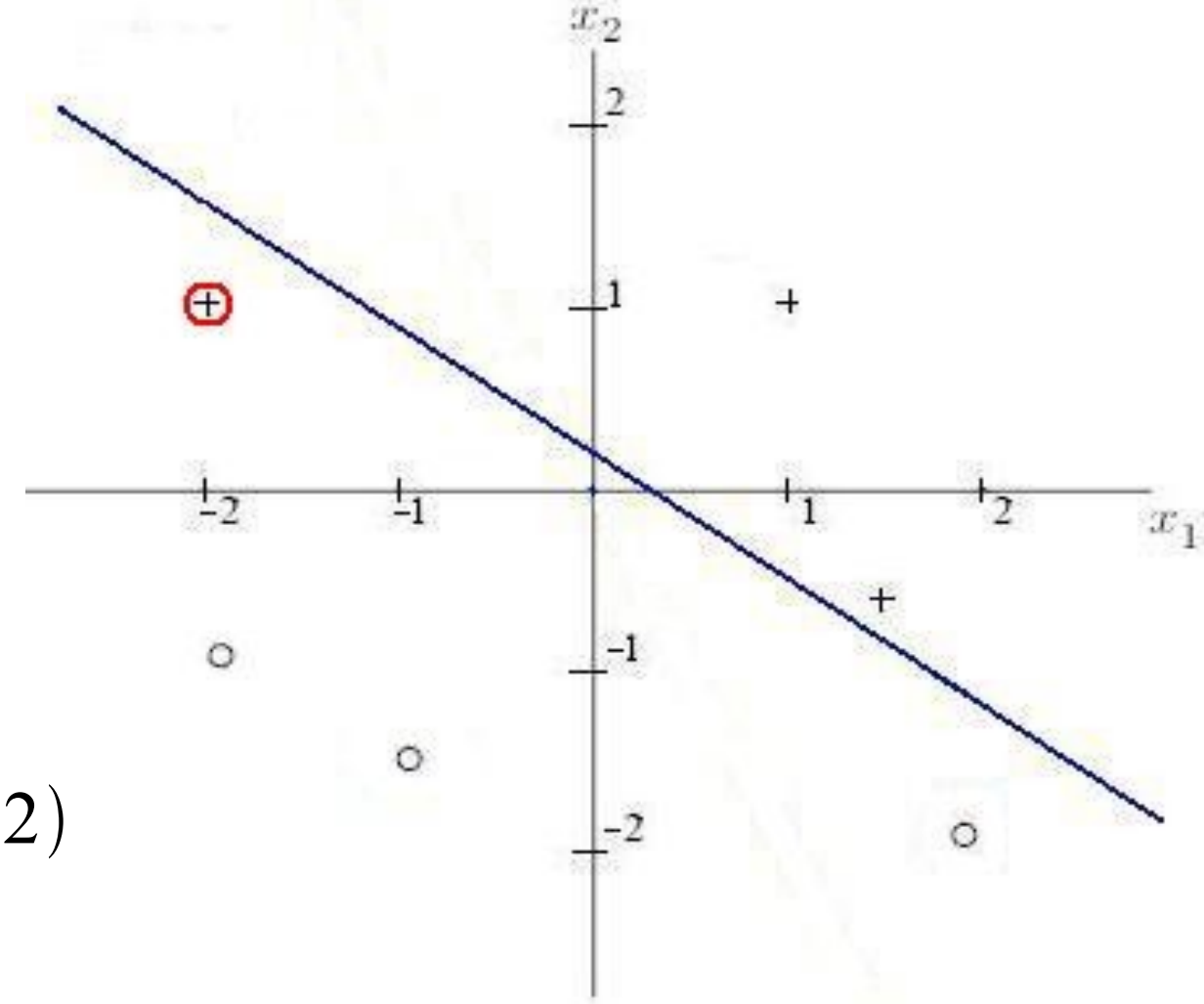
Correct classificatior
no action



Learning Example

$\eta = 0.2$

$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$



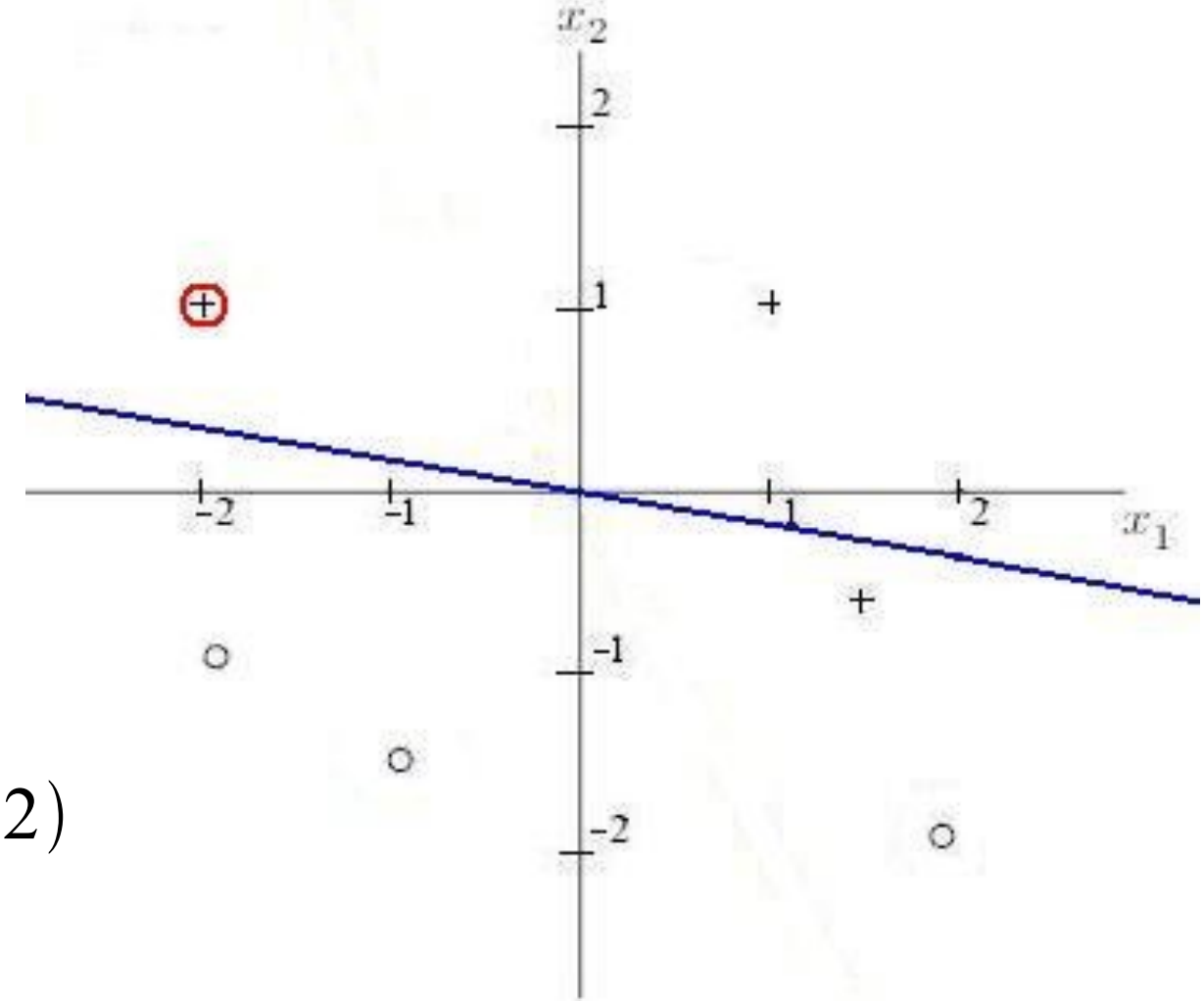
$x_1 = -2,\ x_2 = 1$

$$w_0 = w_0 + 0.2 * 1$$
$$w_1 = w_1 + 0.2 * (-2)$$
$$w_2 = w_2 + 0.2 * 1$$

Learning Example

$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 0.2 \\ 1.1 \end{pmatrix}$$



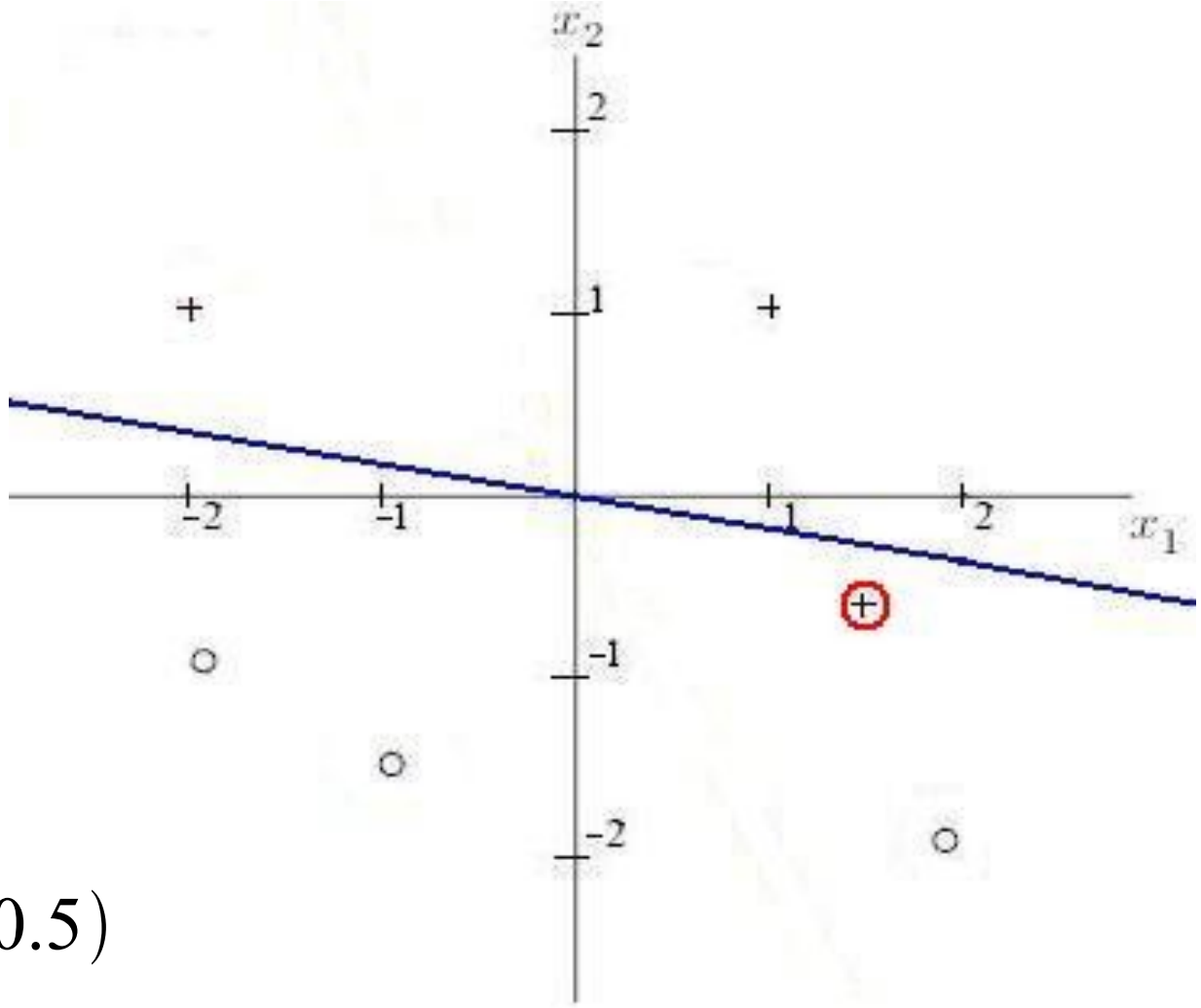
$x_1 = -2, x_2 = 1$

$$w_0 = w_0 + 0.2 * 1$$
$$w_1 = w_1 + 0.2 * (-2)$$
$$w_2 = w_2 + 0.2 * 1$$

Learning Example

$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 0.2 \\ 1.1 \end{pmatrix}$$



$x_1 = 1.5, x_2 = -0.5$

$$w_0 = w_0 + 0.2 * 1$$

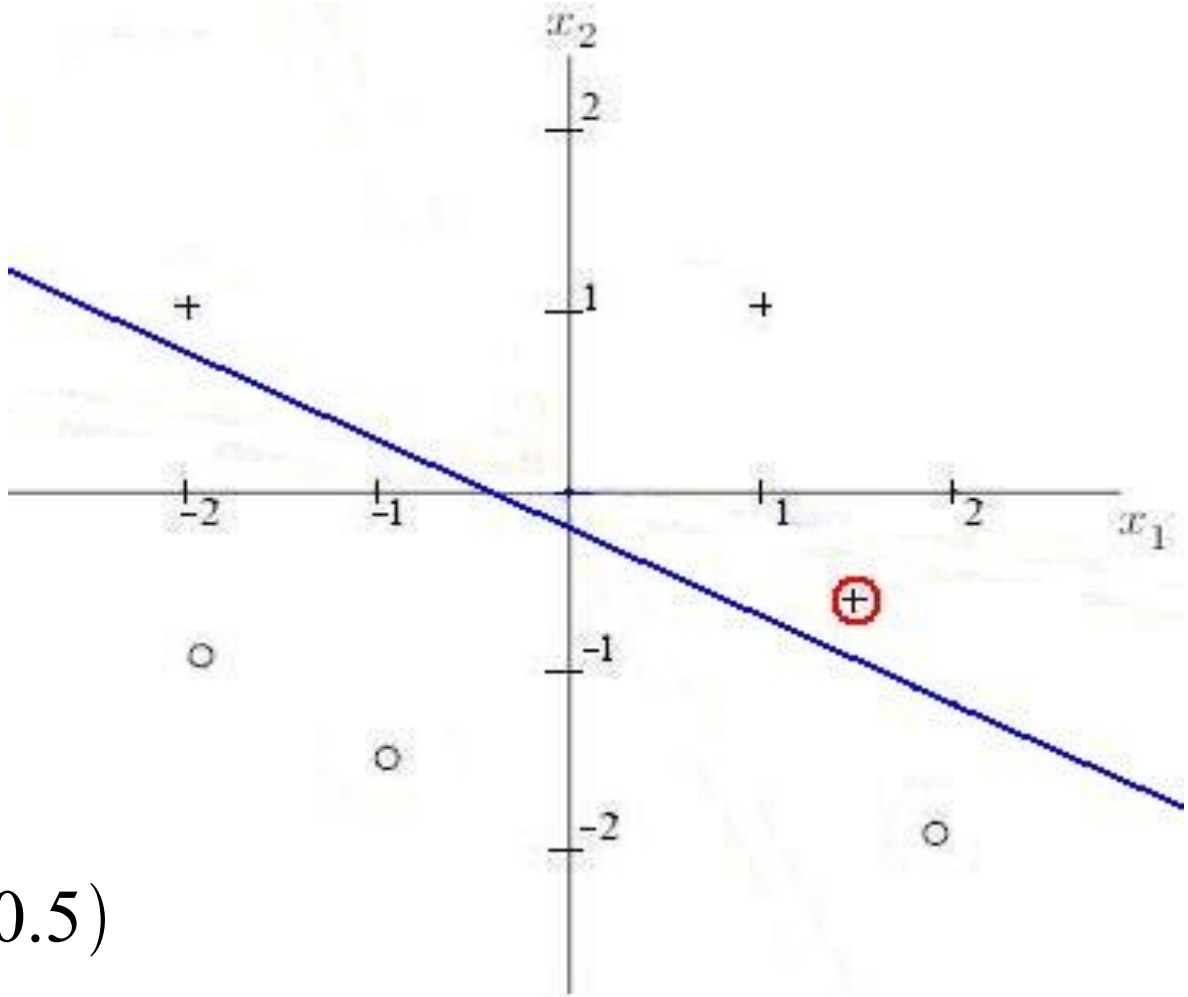
$$w_1 = w_1 + 0.2 * 1.5$$

$$w_2 = w_2 + 0.2 * (-0.5)$$

Learning Example

$\eta = 0.2$

$$w = \begin{pmatrix} 0.2 \\ 0.5 \\ 1 \end{pmatrix}$$



$x_1 = 1.5, x_2 = -0.5$

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * 1.5$$

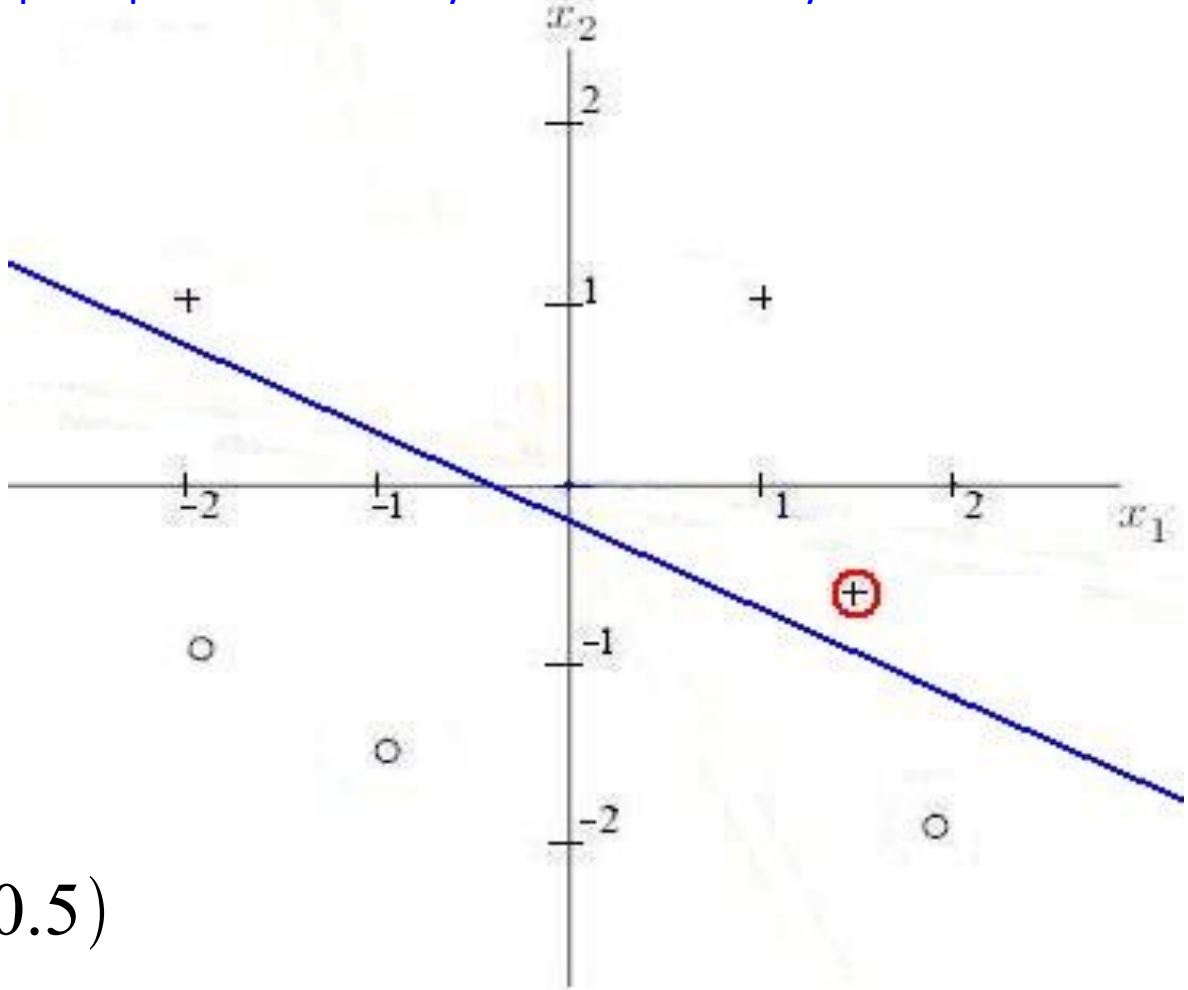
$$w_2 = w_2 + 0.2 * (-0.5)$$

Learning Example

Final weights – learned - the perceptron can classify the data correctly

$\eta = 0.2$

$$w = \begin{pmatrix} 0.2 \\ 0.5 \\ 1 \end{pmatrix}$$



$x_1 = 1.5, x_2 = -0.5$

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * 1.5$$

$$w_2 = w_2 + 0.2 * (-0.5)$$

Perceptron Learning Rule

Adjust the weights as each input is presented.

recall: $s = w_1 x_1 + w_2 x_2 + w_0$

if $g(s) = 0$ but should be 1,

$$\begin{aligned} w_k &\leftarrow w_k + \eta x_k \\ w_0 &\leftarrow w_0 + \eta \\ \text{so} \quad s &\leftarrow s + \eta (1 + \sum_k x_k^2) \end{aligned}$$

if $g(s) = 1$ but should be 0,

$$\begin{aligned} w_k &\leftarrow w_k - \eta x_k \\ w_0 &\leftarrow w_0 - \eta \\ \text{so} \quad s &\leftarrow s - \eta (1 + \sum_k x_k^2) \end{aligned}$$

otherwise, weights are unchanged. ($\eta > 0$ is called the **learning rate**)

Theorem: This will eventually learn to classify the data correctly, as long as they are **linearly separable**.

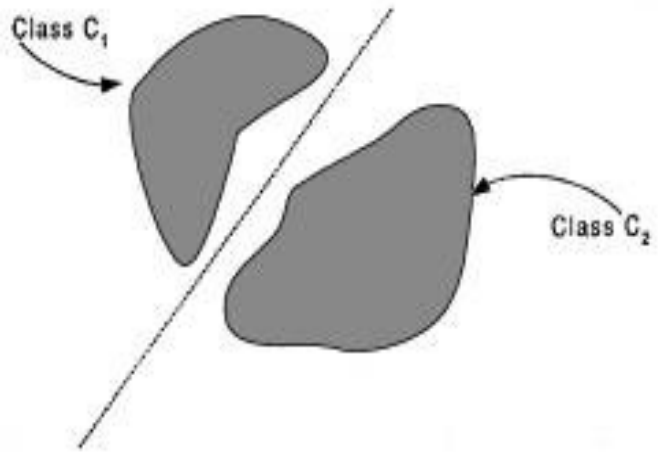
Perceptron Convergence Theorem

The theorem states that for any data set which is linearly separable, the perceptron learning rule is guaranteed to find a solution in a finite number of iterations.

Idea behind the proof: Find upper & lower bounds on the length of the weight vector to show finite number of iterations.

Perceptron Convergence Theorem

Let's assume that the input variables come from two linearly separable classes *C1 & C2*.



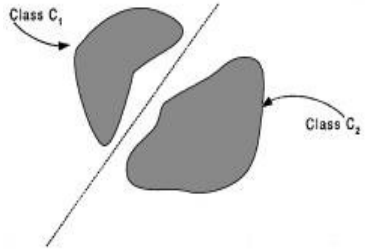
Let T_1 & T_2 be subsets of training vectors which belong to the classes C_1 & C_2 respectively. Then T_1 U T_2 is the complete training set.

Perceptron Convergence Theorem

As we have seen, the learning algorithms purpose is to find a weight vector w such that

$$w \cdot x > 0 \quad \forall x \in C_1$$
$$w \cdot x \leq 0 \quad \forall x \in C_2$$

(***x*** *is an input vector)*



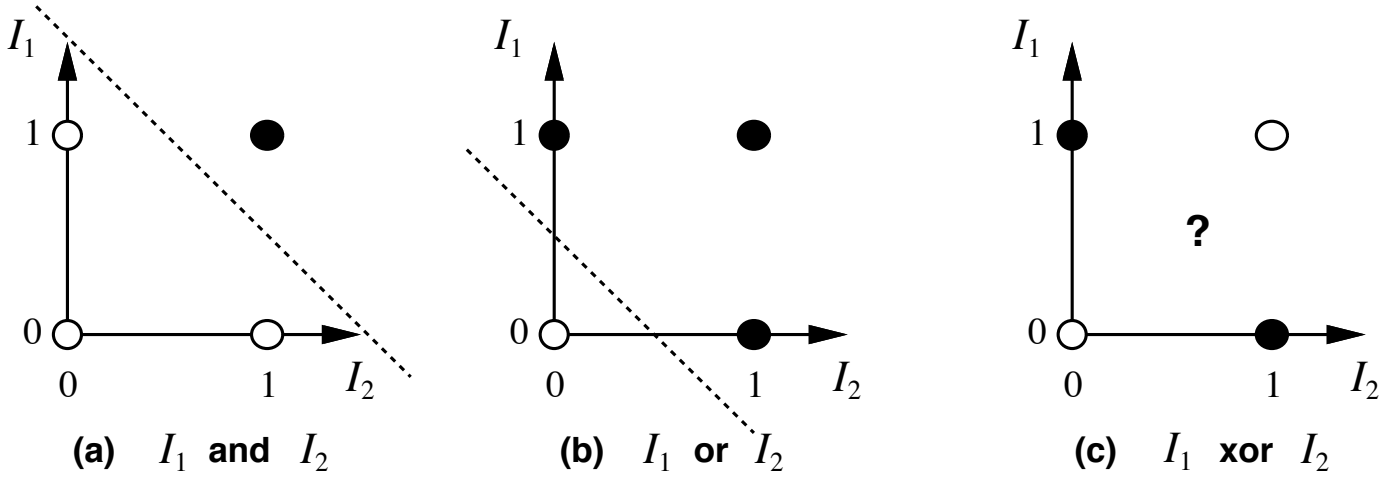
If the kth member of the training set, $x(k)$, is correctly classified by the weight vector $w(k)$ computed at the kth iteration of the algorithm, then we do not adjust the weight vector.

However, if it is incorrectly classified, we use the modifier

$$w(k+1) = w(k) + \eta d(k) x(k)$$

Limitations of Perceptrons

Problem: many useful functions are not linearly separable (e.g. XOR)

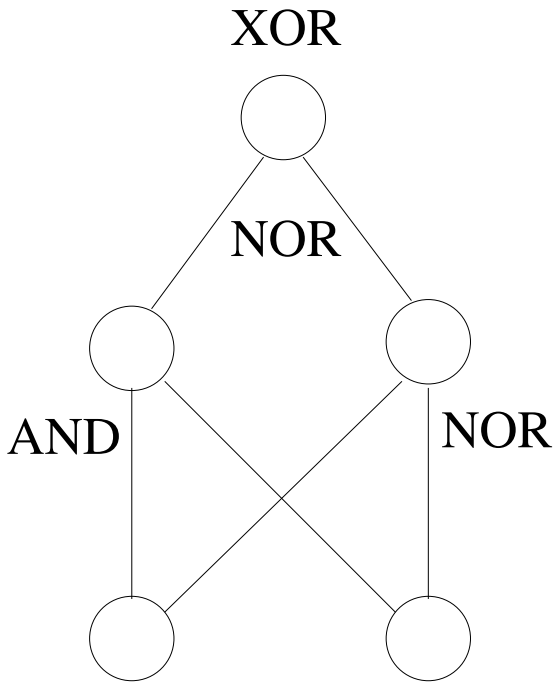


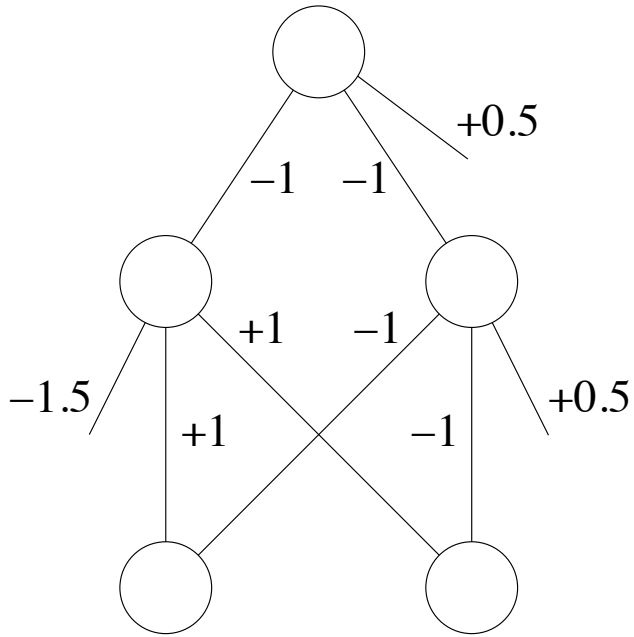
Possible solution:

$x1$ XOR $x2$ can be written as: ($x1$ AND $x2$) NOR ($x1$ NOR $x2$)

Recall that AND, OR and NOR can be implemented by perceptrons.

Multi-Layer Neural Networks





Given an explicit logical function, we can design a multi-layer neural network by hand to compute that function. But, if we are just given a set of training data, can we train a multi-layer network to fit these data?

Historical Context

- In 1969, Minsky and Papert published a book highlighting the limitations of Perceptrons, and lobbied various funding agencies to redirect funding away from neural network research, preferring instead logic-based methods such as expert systems.
- It was known as far back as the 1960's that any given logical function could be implemented in a 2-layer neural network with step function activations. But, the the question of how to learn the weights of a multi-layer neural network based on training examples remained an open problem. The solution, which we describe in the next section, was found in 1976 by Paul Werbos, but did not become widely known until it was rediscovered in 1986 by Rumelhart, Hinton and Williams.