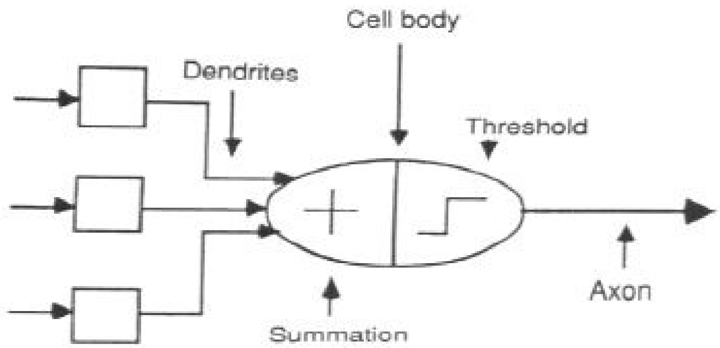
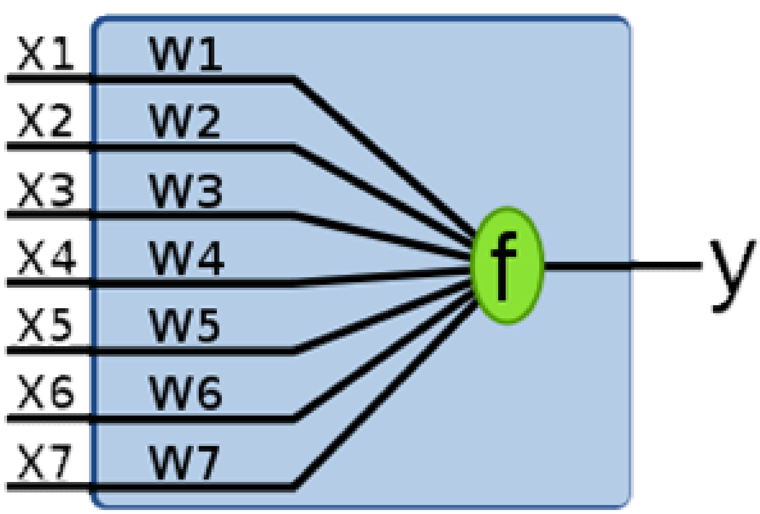
Coming back to Neuron Model



We relate this to A better term PERCEPTRON

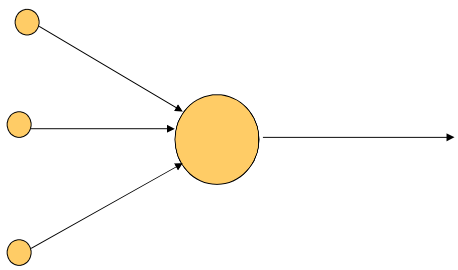
Perceptron



Neural Networks NN 1 210

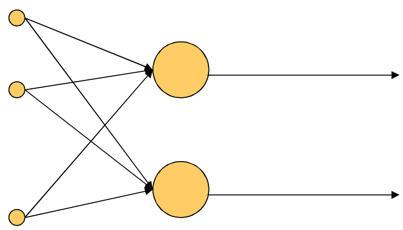
Perceptron: architecture

- We consider the architecture: feed-forward NN with one layer
- It is sufficient to study single layer Perceptron with just one neuron:



Single layer perceptrons

- Generalization to single layer Perceptrons with more neurons is easy because:

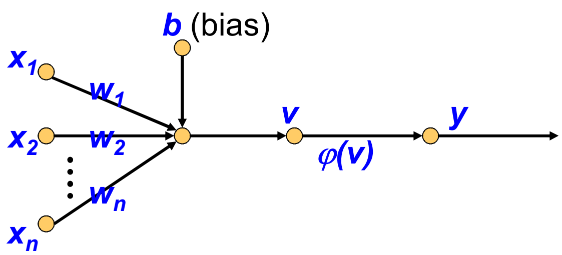


•The output units are independent among each other
•Each weight only affects one of the outputs

Perceptron: Neuron Model

- The (McCulloch-Pitts) Perceptron is a single layer NN with a non-linear φ, the sign function

$$\varphi(v) = \begin{cases} +1 \text{ if } v \geq 0 \\ -1 \text{ if } v < 0 \end{cases}$$



Perceptron for Classification

- The perceptron is used for binary classification
- Given training examples of classes C_1, C_2 train the Perceptron in such a way that it classifies correctly the training examples:
  - ***If the output of the Perceptron is +1 (>0) then the input is assigned to class C_1***
  - ***If the output is -1 (<0)then the input is assigned to C_2***

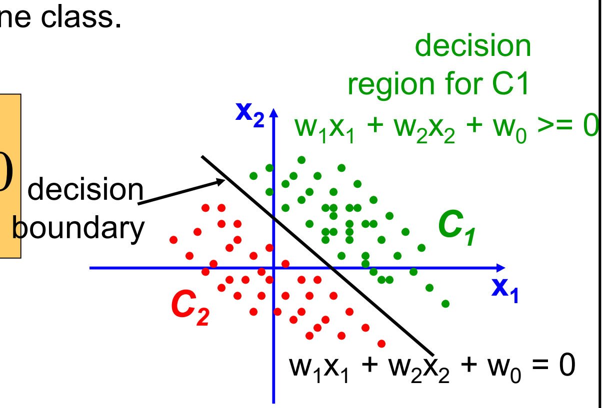
Perceptron Training

- How can we train a Perceptron for a classification task?
- We try to find suitable values for the weights in such a way that the training examples are correctly classified
- Geometrically, we try to find a hyper-plane that separates the examples of the two classes

Perceptron Geometric View

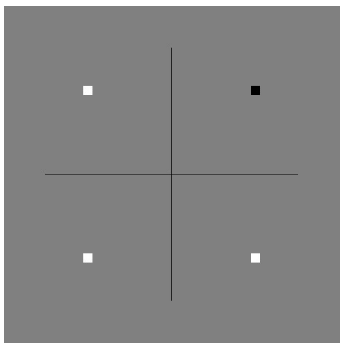
The equation below describes a (hyper-)plane in the input space consisting of real valued 2D vectors. The plane splits the input space into two regions, each of them describing one class.

$$\sum_{i=1}^{2} w_i x_i + w_0 = 0$$



Example: AND

- Here is a representation of the AND function
- White means *false*, black means *true* for output
- -1 means *false*, +1 means *true* for the input



-1 AND -1 = false

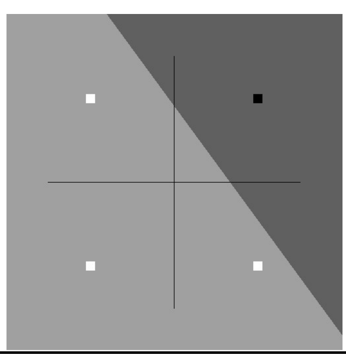
-1 AND +1 = false

+1 AND -1 = false

+1 AND +1 = true

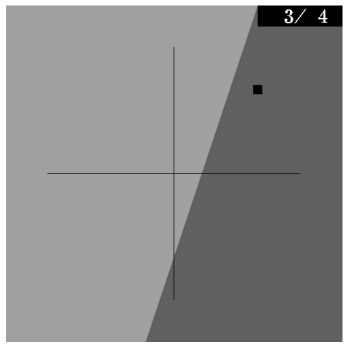
Example: AND continued

- A linear decision surface (a plane in 3D space) intersecting the feature space (the 2D plane where *z*=0) separates *false* from *true* instances



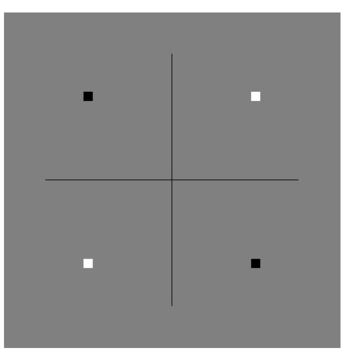
Example: AND continued

- Watch a Perceptron learn the AND function:



Example: XOR

- Here's the XOR function:



-1 XOR -1 = *false*

-1 XOR +1 = *true*

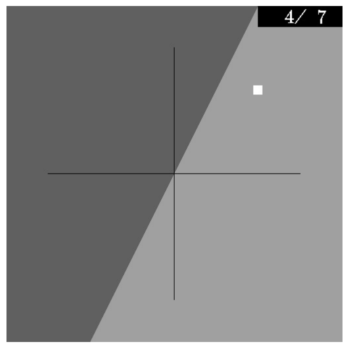
+1 XOR -1 = *true*

+1 XOR +1 = *false*

Perceptrons cannot learn such *linearly inseparable* functions

Example: XOR continued

- Watch a Perceptron try to learn XOR



FAILS?

- How to train the NEURON ?

Fixed increment learning algorithm

- Step 0: Initialize weight and bias.
  - (For simplicity, set weight & bias to zero.)
  - Set learning rate α ($0 < \alpha \leq 1$).
    - (For simplicity, α can be set to 1.)
- Step 1: While stopping condition is false,
  - do steps 2- 5.
- Step 2: For each input set (training pair, input & target), do steps 3-4

Continue…

- Step 3: Compute response of output unit:

$$y\_in = b + \sum_i x_i w_i;$$

$$y = \begin{cases} 1 & \text{if y\_in } \geq \theta \\ -1 & \text{if } < \theta \end{cases}$$

Continue….

- Step 4: Update weights and bias if an error occurred for this pattern.

$$
\begin{aligned}
&\mathit{if}\ \ y \neq t, \\
&\qquad w_i(new) = w_i(old) + \alpha t x_i \\
&\qquad b(new) = b(old) + \alpha\, t. \\
&\mathit{else} \\
&\qquad w_i(new) = w_i(old) \\
&\qquad b(new) = b(old)
\end{aligned}
$$

Continue..

- Step 5. **Test the stopping condition**: If no weights changed in step2, stop; else, continue.

Perceptron: Learning Algorithm

- Variables and parameters at iteration n of the learning algorithm:

x (n) = input vector

= $[+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$

w(n) = weight vector

= $[b(n), w_1(n), w_2(n), \ldots, w_m(n)]^T$

b(n) = bias

y(n) = actual response

d(n) = desired response

η = learning rate parameter

The fixed-increment learning algorithm

n=1;

initialize **w**(n) randomly;

while (there are misclassified training examples)

Select a misclassified augmented example (**x**(n),*d(n)*)

$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta d(n)\mathbf{x}(n)$;

n = n+1;

end-while;

η = learning rate parameter (real number)

WEIGHT CHANGES AND DELTA RULE

- Another learning method is called the **delta rule**, because of the way the Perceptron (We ll come to this shortly) checks its accuracy.
- However, other ways can also be used to training neural network model based on Perceptron.
- The difference between the perceptron's output and the correct output is assigned the Greek letter *delta*, and the *Weight i* for *Input i* is altered like this:

Change in *Weight i* = Current Value of input *i* × (Desired Output - Current Output)

Continue…

- This can be elegantly summed up to:

$$w_i = x_i \delta$$

- The new *Weight i* is found simply by adding the change for *Weight i* to the current value of *Weight i*.
- Wi(new) = Wi(old) + wi (change)

Perceptron Learning Algorithm based on Delta Rule

- Step0: Initialize weight and bias.
  - (For simplicity, set weight & bias to zero.)
  - Set learning rate α ($0 < \alpha \leq 1$).
    - (For simplicity, α can be set to 1.)
- Step1: While stopping condition is false,
  - do steps 2- 5.
- Step2: For each input set (training pair, input & target), do steps 3-4

Continue…

- Step3: Compute response of output unit:

$$y\_in = b + \sum_i x_i w_i;$$

$$y = \begin{cases} 1 & \text{if } y\_in \geq \theta \\ -1 & \text{if} < \theta \end{cases}$$

Continue…

- Step4: Update weights and bias if an error occurred for this pattern.

$$
\begin{aligned}
&if \quad y \neq t, \\
&\qquad w_i(new) = w_i(old) + \alpha(t - y)x_i \\
&\qquad b(new) = b(old) + \alpha\,(t - y) \\
&else \\
&\qquad w_i(new) = w_i(old) \\
&\qquad b(new) = b(old)
\end{aligned}
$$

Continue..

- Step5. **Test the stopping condition**: If no weights changed in step2, stop; else, continue.

Convergence first Then GD

Gradient Descent Training Rule