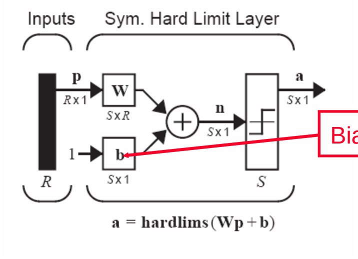
n|w Fachhochschule Nordwestschweiz
Hochschule für Technik

The Perceptron and its Learning Rule

Carlo U. Nicola, SGI FH Aargau
With extracts from publications of :
M. Minsky, MIT, Demuth, U. of Colorado,
D.J. C. MacKay, Cambridge University

Perceptron



Bias b is sometimes called θ

(i) Single layer ANN
(ii) It works with continuous or binary inputs
(iii) It stores pattern pairs (A_k, C_k) where: $A_k = (a_1^k, \ldots, a_n^k)$ and $C_k = (c_1^k, \ldots, c_n^k)$ are bipolar valued [-1, +1].
(iv) It applies the perceptron error-correction procedure, which always converges.
(v) A perceptron is a classifier.

Perceptron convergence procedure (1)

Step1: Initialize weights and thresholds: Set the w_{ij} and the b_j to small random values in the range [-1,+1].

Step2: Present new continuous valued input $p_0, p_1, \cdots, p_n$ along with the desired output vector $T = (t_0, \cdots, t_n)$.

Step3: Calculate actual output: $a_j = f(\Sigma w_{ij} p_i + b_j)$ where f(.) = hardlim(.) = sgn(.).

Step 4: When an error occurs adapt the weights with:
$w_{ij}^{new} = w_{ij}^{old} + \eta[t_j - a_j] \times p_i$ where: $0 < \eta \leq 1$ (learning rate)
and: $b^{new} = b^{old} + e$ where: $e = \eta[t_j - a_j]$

Step 5: Repeat by going to step 2 until no error.

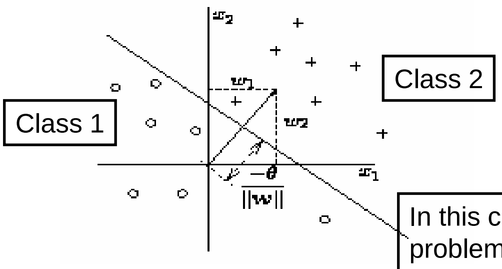
The important propriety of the perceptron rule

The algorithm outlined in the precedent slide, will always converge to weights that solve the desired classification problem, assuming that such weights do exist.
We will see, shortly, that such weights exist, as long as the classification problem is linearly separable.

Example

The discrimination line between two classes in a two input perceptron is determined by: $w_1x_1 + w_2x_2 + \theta = 0$.

We plot this line in a x_1, x_2 plane as: $x_2 = -(w_1/w_2)x_1 - \theta/w_2$



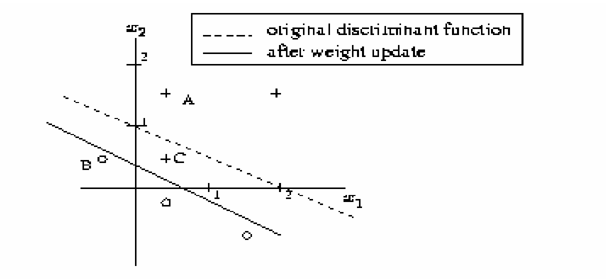
Class 1

Class 2

In this case the classification problem is neatly linear separable and the perceptron rule works very well.

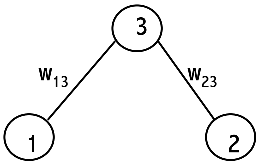
Example of perceptron learning rule

A perceptron is initialized with the following weights: $w_1 = 1$; $w_2 = 2$; $\theta = -2$. The first point A : **x**= (0.5, 1.5) and desired output t(**x**)= +1 is presented to the network. From step 3, it can be calculated that the network output is +1, so no weights are adjusted. The same is the case for point B, with values **x** = (-0.5, 0.5) and target value t(**x**)= -1. Point C: **x**= (0.5, 0.5) gives an output -1, while the expected target is t(**x**)= +1. According to the learning rule we change the weights as follows: $w_1 = 1.5$; $w_2 = 2.5$; $\theta = -1$. ($\eta = 0.5$; $p_C = 0.5$; $\rightarrow \Delta w_1 = 0.5$; $\Delta w_2 = 0.5$; $\Delta\theta = 1$). Now point C too, is correctly classified.



Failure of the perceptron ANN to solve the XOR problem

The XOR-problem (see table):



i_1	i_2	i_1 xor i_2
-1	-1	-1
1	-1	1
-1	1	1
1	1	-1

$$\text{Output}_i = \begin{cases} 1 & \text{if } \sum w_{ij}\, i_i > h_i \\ -1 & \text{if } \sum w_{ij}\, i_i \leq h_i \end{cases}$$

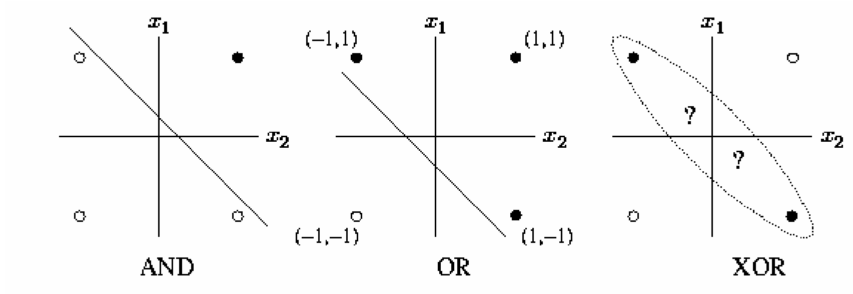
(i) $i_1 = 1, i_2 = 1: w_{13} + w_{23} \leq h_3$ (ii) $i_1 = -1, i_2 = -1 : -(w_{13} + w_{23}) \leq h_3$

(iii) $i_1 = 1, i_2 = -1: w_{13} - w_{23} > h_3$ (iv) $i_1 = -1 , i_2 = 1 : w_{23} - w_{13} > h_3$

(i), (ii), (iii) and (iv) lead to a contradiction!

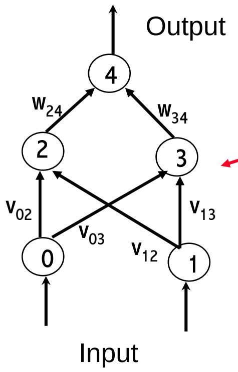
The xor-problem is not linearly separable

Linearly separable and not separable problems



XOR is not a linearly separable classification problem

Perceptrons net with hidden layer

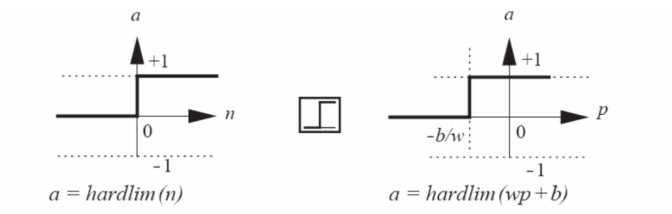


Feedforward hidden layer

Now this perceptrons network can solve the XOR-problem!

Show this.

Transfer function for perceptron



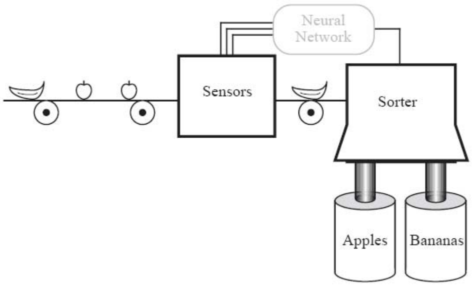
$a = hardlim(n)$

Hard Limit Transfer Function

$a = hardlim(wp + b)$

Single-Input *hardlim* Neuron

A more complex example



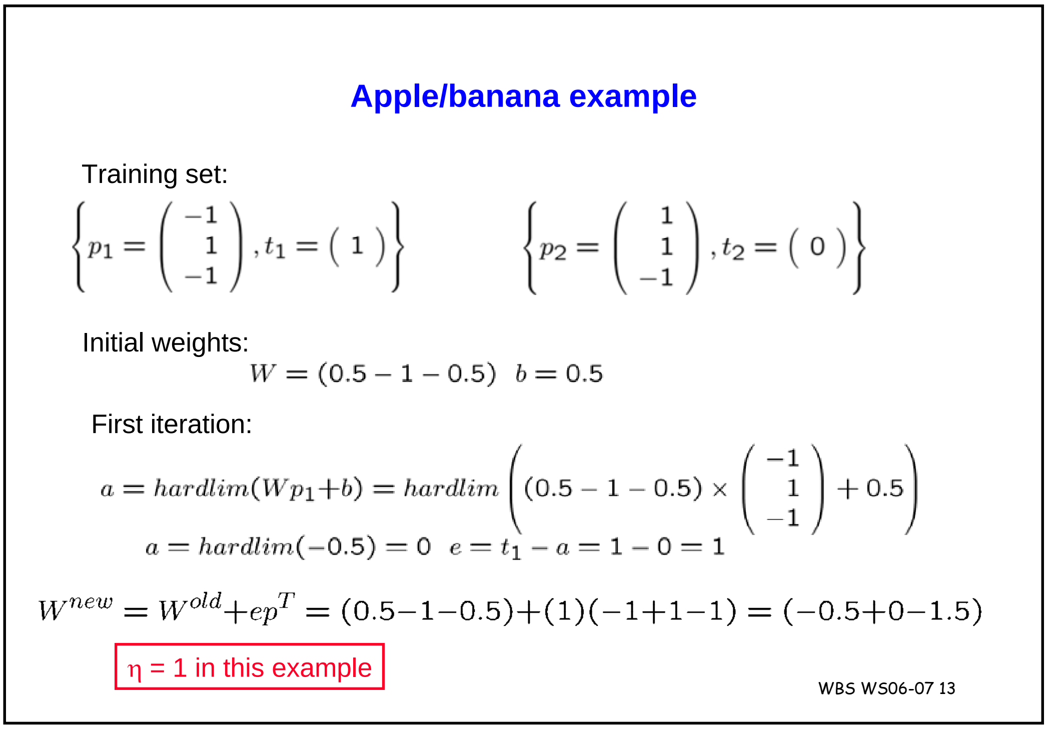
Prototype vectors

We take 3 parameters to differentiate between a banana and a apple: shape, texture, weight.

Shape: {1: round; -1: elliptical}
Texture: {1: smooth; -1: rough}
Weight: {1: > 100 g; -1: < 100 g}

We can now define a prototype banana and apple:

$$p_1 = \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix} \qquad p_2 = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}$$



Apple/banana example

Training set:

$$\left\{ p_1 = \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}, t_1 = \begin{pmatrix} 1 \end{pmatrix} \right\} \qquad \left\{ p_2 = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}, t_2 = \begin{pmatrix} 0 \end{pmatrix} \right\}$$

Initial weights:

$$W = (0.5 - 1 - 0.5) \quad b = 0.5$$

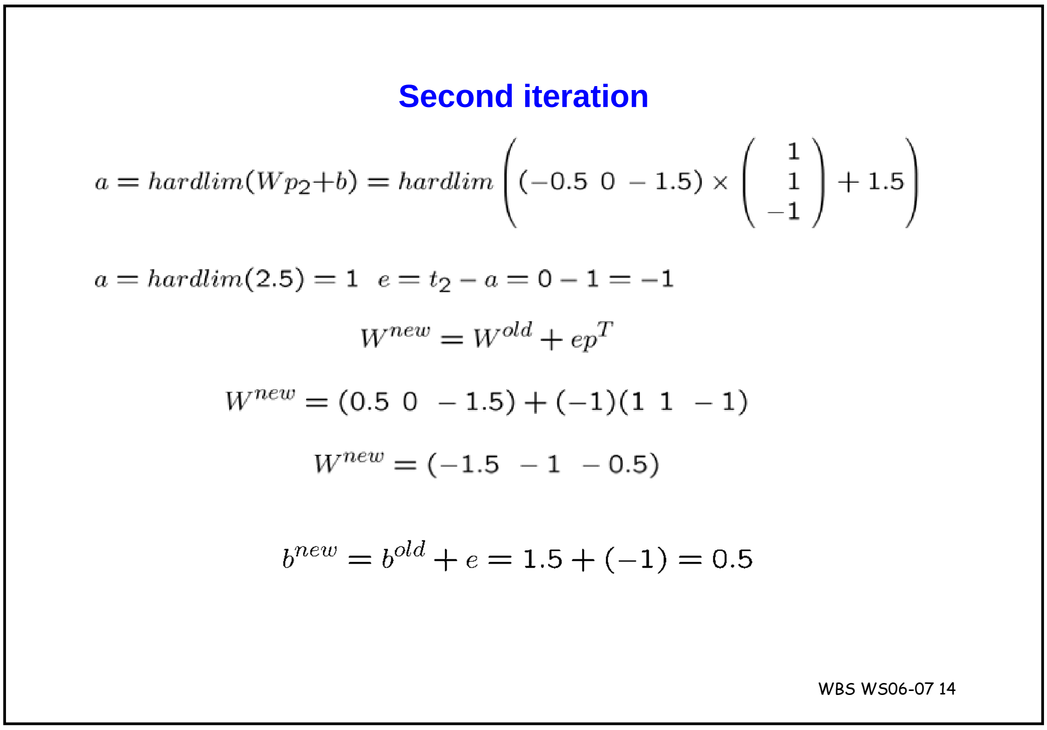
First iteration:

$$a = hardlim(Wp_1 + b) = hardlim\left((0.5 - 1 - 0.5) \times \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix} + 0.5 \right)$$

$$a = hardlim(-0.5) = 0 \quad e = t_1 - a = 1 - 0 = 1$$

$$W^{new} = W^{old} + ep^T = (0.5 - 1 - 0.5) + (1)(-1 + 1 - 1) = (-0.5 + 0 - 1.5)$$

η = 1 in this example



Second iteration

$$a = hardlim(Wp_2 + b) = hardlim\left((-0.5\ \ 0\ \ -1.5) \times \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} + 1.5 \right)$$

$$a = hardlim(2.5) = 1 \quad e = t_2 - a = 0 - 1 = -1$$

$$W^{new} = W^{old} + ep^T$$

$$W^{new} = (0.5\ \ 0\ \ -1.5) + (-1)(1\ \ 1\ \ -1)$$

$$W^{new} = (-1.5\ \ -1\ \ -0.5)$$

$$b^{new} = b^{old} + e = 1.5 + (-1) = 0.5$$

Checking the solution (test vectors)

$$a = hardlim(Wp_1{+}b) = hardlim\left((-1.5\ -1\ -0.5) \times \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix} + 0.5\right)$$

$$a = hardlim(1.5) = 1 = t_1$$

$$a = hardlim(Wp_2{+}b) = hardlim\left((-1.5\ -1\ -0.5) \times \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} + 0.5\right)$$

$$a = hardlim(-1.5) = 0 = t_2$$

Checking the solution (testing the network)

$$a = hardlim\left((-1.5\ -1\ -0.5) \times \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix} + 0.5\right)$$

$$a = hardlim(1) = 1(banana)$$

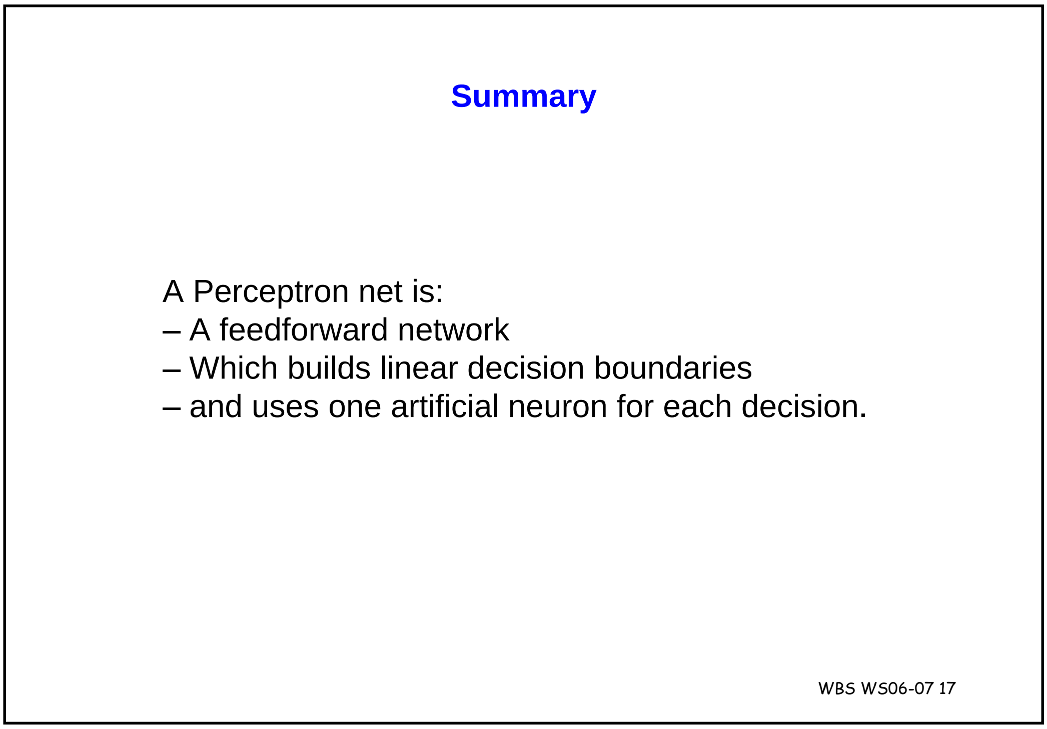
$$a = hardlim\left((-1.5\ -1\ -0.5) \times \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} + 0.5\right)$$

$$a = hardlim(-2) = -1(apple)$$

The net recovers the correct answer from noisy information:

$$a = hardlim\left((-1.5\ -1\ -0.5) \times \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} + 0.5\right)$$

$$a = hardlim(3) = 1(banana)$$



Summary

A Perceptron net is:
– A feedforward network
– Which builds linear decision boundaries
– and uses one artificial neuron for each decision.