PMR5406 Redes Neurais e Lógica Fuzzy

Aula 3
Single Layer Percetron

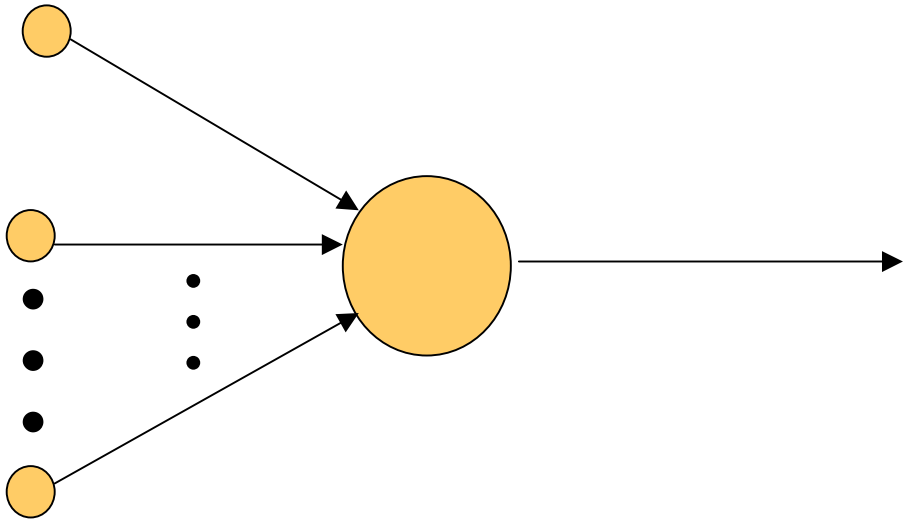
Baseado em:

Neural Networks, Simon Haykin, Prentice-Hall, 2nd edition

Slides do curso por Elena Marchiori, Vrije Unviersity

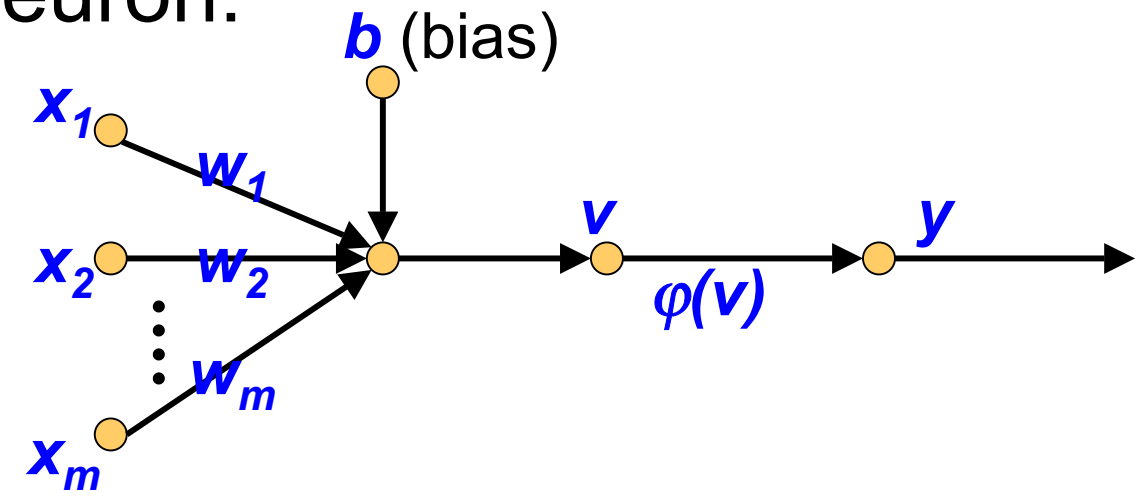
Architecture

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



Perceptron: Neuron Model

- Uses a non-linear (McCulloch-Pitts) model of neuron:



- φ is the *sign* function:

$$\varphi(v) = \begin{cases} +1 & \text{IF } v >= 0 \\ -1 & \text{IF } v < 0 \end{cases}$$

Is the function *sign*(v)

Perceptron: Applications

- The perceptron is used for classification: classify correctly a set of examples into one of the two classes C_1, C_2:

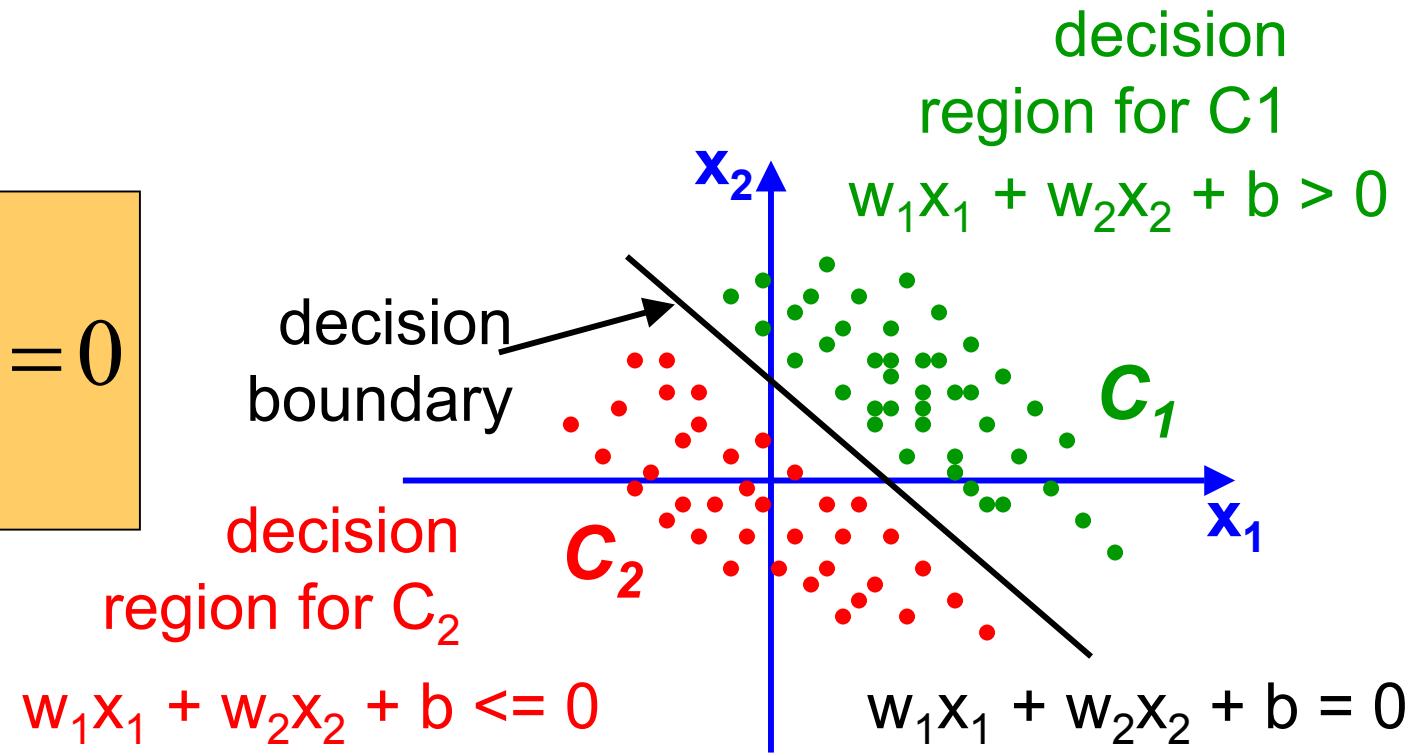
If the output of the perceptron is +1 then the input is assigned to class C_1

If the output is -1 then the input is assigned to C_2

Perceptron: Classification

- The equation below describes a hyperplane in the input space. This hyperplane is used to separate the two classes C1 and C2

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

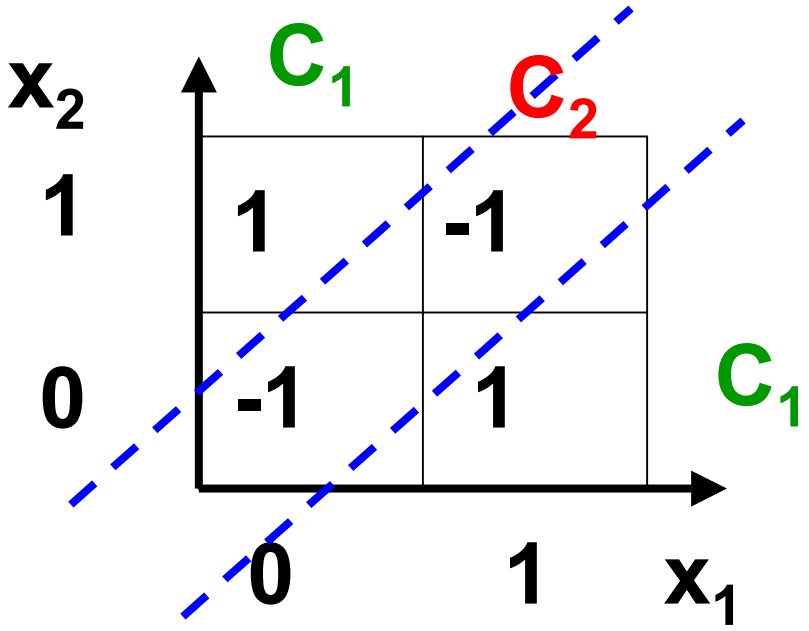


Perceptron: Limitations

- The perceptron can only model linearly separable functions.
- The perceptron can be used to model the following Boolean functions:
- AND
- OR
- COMPLEMENT
- But it cannot model the XOR. Why?

Perceptron: Limitations

- The XOR is not linear separable
- It is impossible to separate the classes C_1 and C_2 with only one line



Perceptron: Learning Algorithm

- Variables and parameters

$\mathbf{x}(n)$ = input vector

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

$\mathbf{w}(n)$ = 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

- Initialization: set $\mathbf{w}(0) = 0$
- Activation: activate perceptron by applying input example (vector $\mathbf{x}(n)$ and desired response $d(n)$)
- Compute actual response of perceptron:

$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

- Adapt weight vector: if $d(n)$ and $y(n)$ are different then

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n)-y(n)]\mathbf{x}(n)$$

$$\text{Where } d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \in C_1 \\ -1 & \text{if } \mathbf{x}(n) \in C_2 \end{cases}$$

- Continuation: increment time step n by 1 and go to Activation step

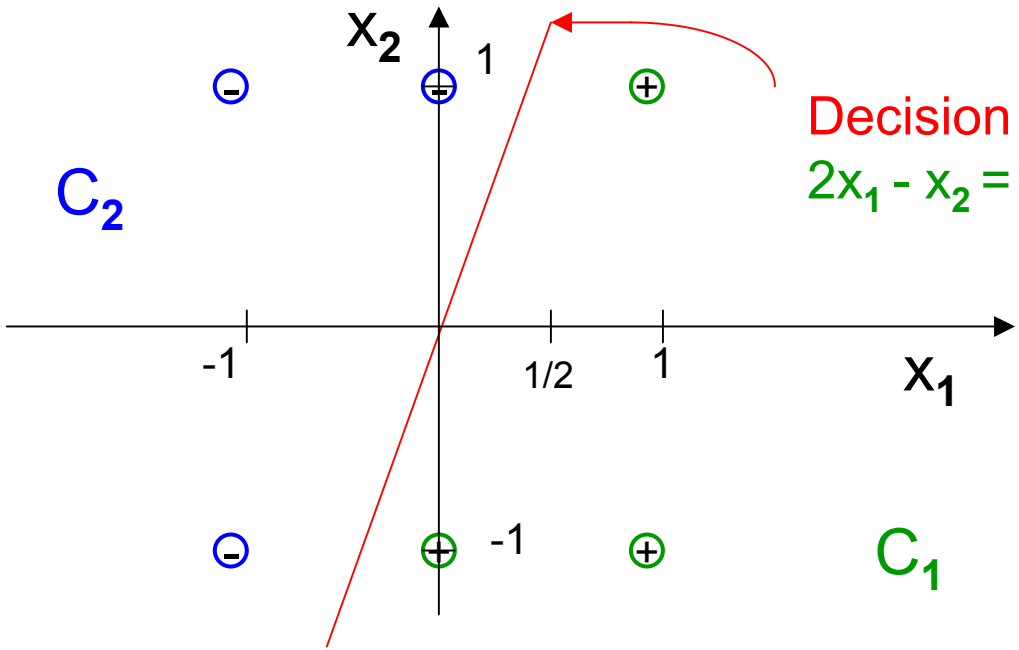
Example

Consider a training set $C_1 \cup C_2$, where:
$C_1 = \{(1,1), (1, -1), (0, -1)\}$ elements of class 1
$C_2 = \{(-1,-1), (-1,1), (0,1)\}$ elements of class -1

Use the perceptron learning algorithm to classify these examples.

- $\mathbf{w}(0) = [1, 0, 0]^T$ $\eta = 1$

Example



Convergence of the learning algorithm

Suppose datasets C_1, C_2 are linearly separable. The perceptron convergence algorithm converges after n_0 iterations, with $n_0 \leq n_{max}$ on training set $C_1 \cup C_2$.

Proof:

- suppose $\mathbf{x} \in C_1 \Rightarrow$ output = 1 and $\mathbf{x} \in C_2 \Rightarrow$ output = -1.
- For simplicity assume $\mathbf{w}(1) = 0$, $\eta = 1$.
- Suppose perceptron incorrectly classifies $\mathbf{x}(1) \ldots \mathbf{x}(n) \ldots \in \mathbf{C_1}$.

Then $\mathbf{w}^T(k)\,\mathbf{x}(k) \leq 0$.

$\Rightarrow$ Error correction rule:

$$\mathbf{w}(2) = \mathbf{w}(1) + \mathbf{x}(1)$$
$$\mathbf{w}(3) = \mathbf{w}(2) + \mathbf{x}(2)$$
$$\vdots \qquad \vdots \qquad \vdots$$
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n).$$

$\Rightarrow \mathbf{w}(n+1) = \mathbf{x}(1) + \ldots + \mathbf{x}(n)$

Convergence theorem (proof)

- Let $\mathbf{w_0}$ be such that $\mathbf{w_0}^T \mathbf{x}(n) > 0 \quad \forall \mathbf{x}(n) \in C_1$.

w_0 exists because C_1 and C_2 are linearly separable.

- Let $\alpha = \min w_0^T x(n) \mid x(n) \in C_1$.

- Then $\mathbf{w_0}^T w(n+1) = \mathbf{w_0}^T \mathbf{x}(1) + \ldots + \mathbf{w_0}^T \mathbf{x}(n) \geq n\alpha$

- Cauchy-Schwarz inequality:

$$\|\mathbf{w_0}\|^2 \; \|\mathbf{w}(n+1)\|^2 \geq [\mathbf{w_0}^T \mathbf{w}(n+1)]^2$$

$$\|\mathbf{w}(n+1)\|^2 \geq \frac{n^2 \alpha^2}{\|w_0\|^2} \qquad \textbf{(A)}$$

Convergence theorem (proof)

- Now we consider another route:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k)$$

$$\|\mathbf{w}(k+1)\|^2 = \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 + 2\,\mathbf{w}^T(k)\mathbf{x}(k)$$

$\|\cdot\|$: euclidean norm

$\mathbf{w}^T(k)\mathbf{x}(k) \le 0$ because $\mathbf{x}(k)$ is misclassified

$$\Rightarrow \|\mathbf{w}(k+1)\|^2 \le \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 \quad k=1,..,n$$

$=0$

$$\|\mathbf{w}(2)\|^2 \le \|\mathbf{w}(1)\|^2 + \|\mathbf{x}(1)\|^2$$

$$\|\mathbf{w}(3)\|^2 \le \|\mathbf{w}(2)\|^2 + \|\mathbf{x}(2)\|^2$$

$$\vdots$$

$$\Rightarrow \|\mathbf{w}(n+1)\|^2 \le \sum_{k=1}^{n} \|x(k)\|^2$$

convergence theorem (proof)

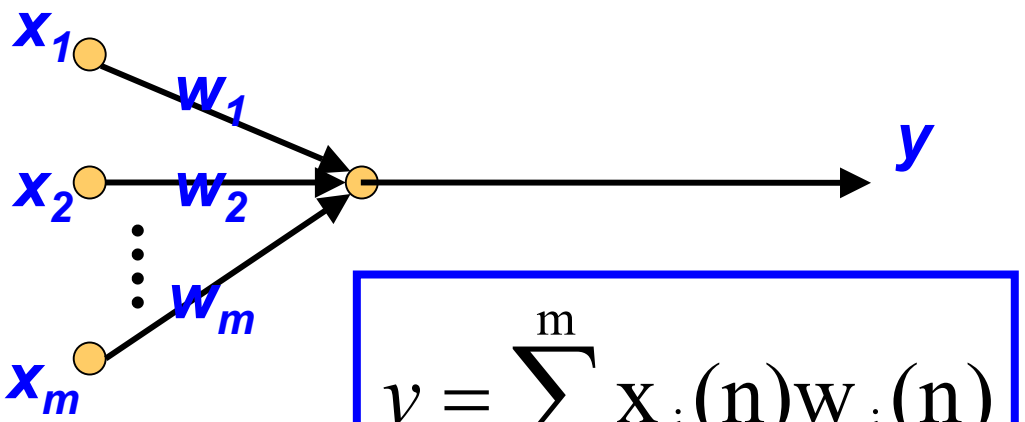
- Let $\beta = \max \|\mathbf{x}(n)\|^2 \quad \mathbf{x}(n) \in C_1$
- $\|\mathbf{w}(n+1)\|^2 \leq \mathbf{n}\,\beta$ **(B)**
- For sufficiently large values of k:
  (B) becomes in conflict with **(A).**
  Then n cannot be greater than n_{max} such that **(A)** and **(B)** are both satisfied with the equality sign.

$$\frac{n_{max}^2 \alpha^2}{\| w_0 \|^2} = n_{max}\beta \Rightarrow \boxed{n_{max} = \frac{\| w_0 \|^2}{\alpha^2}\beta}$$

- Perceptron convergence algorithm terminates in at most
  $n_{max} = \frac{\beta \|\mathbf{w_0}\|^2}{\alpha^2}$ iterations.

Adaline: Adaptive Linear Element

- The output **y** is a linear combination o **x**



$$y = \sum_{j=0}^{m} x_j(n) w_j(n)$$

Adaline: Adaptive Linear Element

- Adaline: uses a linear neuron model and the Least-Mean-Square (LMS) learning algorithm

  The idea: try to minimize the square error, which is a function of the weights

$$E(\mathrm{w}(\mathrm{n})) = \frac{1}{2}\mathrm{e}^2(\mathrm{n})$$

$$\mathrm{e}(\mathrm{n}) = \mathrm{d}(\mathrm{n}) - \sum_{\mathrm{j}=0}^{\mathrm{m}} \mathrm{x}_{\mathrm{j}}(\mathrm{n})\mathrm{w}_{\mathrm{j}}(\mathrm{n})$$

- We can find the minimum of the error function *E* by means of the Steepest descent method

Steepest Descent Method

- start with an arbitrary point
- find a direction in which E is decreasing most rapidly

$$-(\text{gradient of } E(\mathrm{w})) = -\left[\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_m}\right]$$

- make a small step in that direction

$$\mathrm{w}(\mathrm{n}+1) = \mathrm{w}(\mathrm{n}) - \eta(\text{gradientof } \mathrm{E}(\mathrm{n}))$$

*L*east-*M*ean-*S*quare algorithm (Widrow-Hoff algorithm)

- Approximation of gradient(E)

$$\frac{\partial E(\mathrm{w(n)})}{\partial \mathrm{w(n)}} = \mathrm{e(n)}\frac{\partial \mathrm{e(n)}}{\partial \mathrm{w(n)}}$$
$$= \mathrm{e(n)}[-\mathrm{x(n)}^{\mathrm{T}}]$$

- Update rule for the weights becomes:

$$\mathrm{w(n+1)} = \mathrm{w(n)} + \eta \mathrm{x(n)e(n)}$$

Summary of LMS algorithm

Training sample: input signal vector **x(n)**

desired response **d(n)**

User selected parameter η ***>0***

Initialization **set ŵ(1) = 0**

Computation **for n = 1, 2, … compute**

$$e(n) = d(n) - \hat{w}^T(n)x(n)$$

$$\hat{w}(n+1) = \hat{w}(n) + \eta\, x(n)e(n)$$