

PMR5406 Redes Neurais e Lógica Fuzzy

Aula 3 Single Layer Percetron

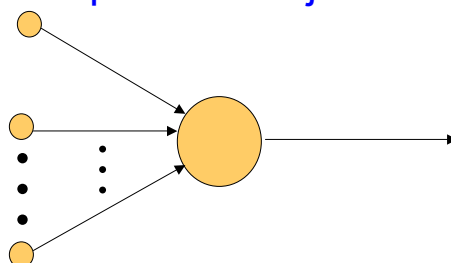
Baseado em:

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

Slides do curso por Elena Marchiori, Vrije University

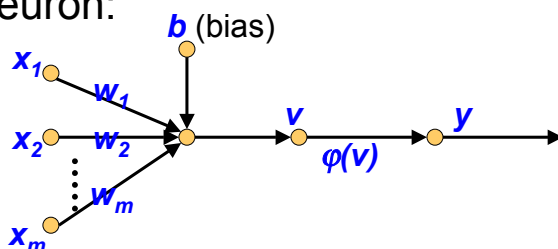
Architecture

- 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:

$$\phi(v) = \begin{cases} +1 & \text{IF } v \geq 0 \\ -1 & \text{IF } v < 0 \end{cases} \quad \text{Is the function } \textit{sign}(v)$$

Perceptron: Applications

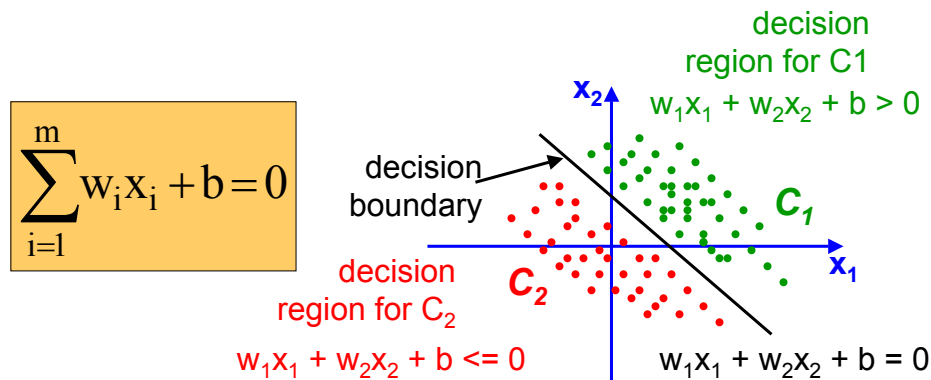
- The perceptron is used for classification:
classify correctly a set of examples into
one of the two classes C_1, C_2 :

*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



PMR5406 Redes Neurais e
Lógica Fuzzy

Single Layer Perceptron

5

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?

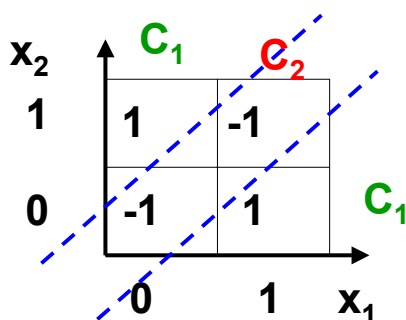
PMR5406 Redes Neurais e
Lógica Fuzzy

Single Layer Perceptron

6

Perceptron: Limitations

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



PMR5406 Redes Neurais e
Lógica Fuzzy

Single Layer Perceptron

7

Perceptron: Learning Algorithm

- Variables and parameters

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

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

$\mathbf{w}(n)$ = weight vector

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

$b(n)$ = bias

$y(n)$ = actual response

$d(n)$ = desired response

η = learning rate parameter

PMR5406 Redes Neurais e
Lógica Fuzzy

Single Layer Perceptron

8

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

PMR5406 Redes Neurais e
Lógica Fuzzy

Single Layer Perceptron

9

Example

Consider a training set $C_1 \cup C_2$, where:

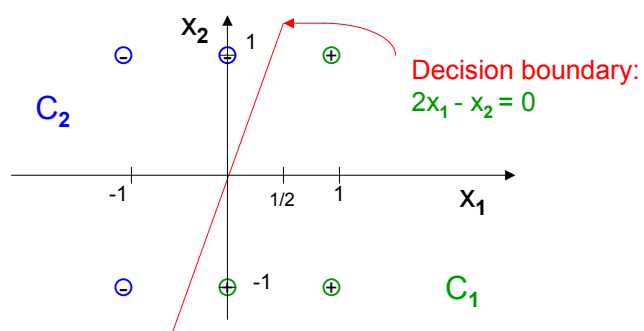
$C_1 = \{(1,1), (1, -1), (0, -1)\}$ elements of class 1

$C_2 = \{(-1,-1), (-1,1), (0,1)\}$ elements of class -1

Use the perceptron learning algorithm to classify these examples.

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

Example



PMR5406 Redes Neurais e
Lógica Fuzzy

Single Layer Perceptron

11

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 $\mathbf{x} \in C_1 \Rightarrow \text{output} = 1$ and $\mathbf{x} \in C_2 \Rightarrow \text{output} = -1$.
 - For simplicity assume $\mathbf{w}(1) = 0$, $\eta = 1$.
 - Suppose perceptron incorrectly classifies $\mathbf{x}(1) \dots \mathbf{x}(n) \dots \in C_1$.
- Then $\mathbf{w}^T(k) \mathbf{x}(k) \leq 0$.

$\Rightarrow$ Error correction rule:

$$\left. \begin{array}{lcl} \mathbf{w}(2) & = & \mathbf{w}(1) + \mathbf{x}(1) \\ \mathbf{w}(3) & = & \mathbf{w}(2) + \mathbf{x}(2) \\ \vdots & & \vdots \\ \mathbf{w}(n+1) & = & \mathbf{w}(n) + \mathbf{x}(n). \end{array} \right\} \Rightarrow \mathbf{w}(n+1) = \mathbf{x}(1) + \dots + \mathbf{x}(n)$$

Convergence theorem (proof)

- Let $\mathbf{w}_0$ be such that $\mathbf{w}_0^T \mathbf{x}(n) > 0 \quad \forall \mathbf{x}(n) \in C_1$.
 $\mathbf{w}_0$ exists because C_1 and C_2 are linearly separable.
- Let $\alpha = \min \mathbf{w}_0^T \mathbf{x}(n) \mid \mathbf{x}(n) \in C_1$.
- Then $\mathbf{w}_0^T \mathbf{w}(n+1) = \mathbf{w}_0^T \mathbf{x}(1) + \dots + \mathbf{w}_0^T \mathbf{x}(n) \geq n\alpha$
- Cauchy-Schwarz inequality:

$$\|\mathbf{w}_0\|^2 \|\mathbf{w}(n+1)\|^2 \geq [\mathbf{w}_0^T \mathbf{w}(n+1)]^2$$

$$\|\mathbf{w}(n+1)\|^2 \geq \frac{n^2 \alpha^2}{\|\mathbf{w}_0\|^2} \quad (\text{A})$$

Convergence theorem (proof)

- Now we consider another route:

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

$$\|\mathbf{w}(k+1)\|^2 = \underbrace{\|\mathbf{w}(k)\|^2}_{\text{euclidean norm}} + \underbrace{\|\mathbf{x}(k)\|^2 + 2\mathbf{w}^T(k)\mathbf{x}(k)}_{\leq 0 \text{ because } \mathbf{x}(k) \text{ is misclassified}}$$

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

$$\begin{aligned} &\downarrow = 0 \\ \|\mathbf{w}(2)\|^2 &\leq \|\mathbf{w}(1)\|^2 + \|\mathbf{x}(1)\|^2 \\ \|\mathbf{w}(3)\|^2 &\leq \|\mathbf{w}(2)\|^2 + \|\mathbf{x}(2)\|^2 \\ &\vdots \\ \Rightarrow \|\mathbf{w}(n+1)\|^2 &\leq \sum_{k=1}^n \|\mathbf{x}(k)\|^2 \end{aligned}$$

convergence theorem (proof)

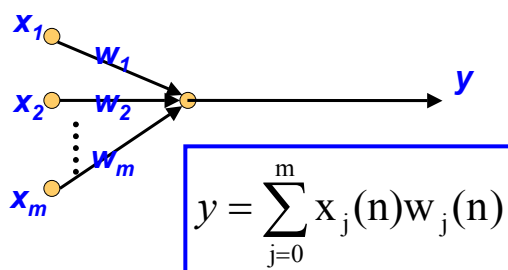
- Let $\beta = \max ||\mathbf{x}(n)||^2 \quad \mathbf{x}(n) \in C_1$
- $||\mathbf{w}(n+1)||^2 \leq n \beta$ (B)
- For sufficiently large values of n :
(B) becomes in conflict with (A).
 Then n cannot be greater than $n_{\max}$ such that (A) and (B) are both satisfied with the equality sign.

$$\frac{n_{\max}^2 \alpha^2}{||\mathbf{w}_0||^2} = n_{\max} \beta \Rightarrow n_{\max} = \frac{||\mathbf{w}_0||^2}{\alpha^2} \beta$$

- 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 of $\mathbf{x}$



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

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

$$e(n) = d(n) - \sum_{j=0}^m x_j(n) w_j(n)$$

- 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

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

- make a small step in that direction

$$w(n+1) = w(n) - \eta(\text{gradient of } E(n))$$

Least-Mean-Square algorithm (Widrow-Hoff algorithm)

- Approximation of gradient(E)

$$\begin{aligned}\frac{\partial E(w(n))}{\partial w(n)} &= e(n) \frac{\partial e(n)}{\partial w(n)} \\ &= e(n)[-x(n)^T]\end{aligned}$$

- Update rule for the weights becomes:

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

Summary of LMS algorithm

Training sample: input signal vector $x(n)$
desired response $d(n)$

User selected parameter $\eta > 0$

Initialization **set $\hat{w}(1) = 0$**

Computation **for $n = 1, 2, \dots$ compute**
 $e(n) = d(n) - \hat{w}^T(n)x(n)$
 $\hat{w}(n+1) = \hat{w}(n) + \eta x(n)e(n)$