2. Perceptron

Prof. Renato Tinós

Depto. de Computação e Matemática (FFCLRP/USP)

2. Perceptron

2.1. Introdução

2.2. Funcionamento do perceptron

2.3. Aprendizado

2.4. Teorema da convergência

2.5. Alguns aspectos do problema de reconhecimento de padrões

2.3. Aprendizado

- **Suponha que as variáveis de entrada se originem de duas classes C_1 e C_2 linearmente separáveis**
- **O problema do aprendizado do *perceptron* (treinamento) consiste em encontrar um conjunto de pesos e bias que definem um hiperplano que separe linearmente os vetores de entrada das classes C_1 e C_2**
  - Isto é, um vetor de pesos **w** e um bias w_0 tal que

$$\begin{cases} \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 > 0 \quad , \text{ para todo } \mathbf{x} \text{ pertencente à classe } C_1 \\ \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 \leq 0 \quad , \text{ para todo } \mathbf{x} \text{ pertencente à classe } C_2 \end{cases} \tag{2.12}$$

2.3. Aprendizado

- **Para isso, um conjunto com padrões para o treinamento do *perceptron* é definido**
  - Conjunto de treinamento *T*
    - » Conjunto de pares entrada **x**(*n*) e saída desejada *d* (*n*)
- **O conjunto de treinamento *T* é composto por dois subconjuntos**
  - T_1 composto por vetores de entrada que pertencem à classe C_1
  - T_2 composto por pares entrada que pertencem à classe C_2

2.3. Aprendizado

- **Antes de apresentarmos a estratégia de treinamento, vamos definir o padrão de treinamento e o vetor de pesos na *n*-ésima iteração do algoritmo como**

$$\mathbf{x}(n) = \begin{bmatrix} +1 \\ x_1(n) \\ x_2(n) \\ \mathrm{M} \\ x_m(n) \end{bmatrix} \qquad \mathbf{w}(n) = \begin{bmatrix} w_0(n) \\ w_1(n) \\ w_2(n) \\ \mathrm{M} \\ w_m(n) \end{bmatrix} \tag{2.13}$$

  - Note que o bias agora é visto como um peso
  - A saída do *k*-ésimo neurônio será dada por

$$y_k(n) = \varphi(u_k(n)) = \varphi\left(\mathbf{w}(n)^{\mathrm{T}}\,\mathbf{x}(n)\right) \tag{2.14}$$

2.3. Aprendizado

- **Estratégia de aprendizado para duas classes**

1. Se o padrão de treinamento **x**(*n*) é corretamente classificado utilizando-se o vetor de pesos **w**(*n*) calculado na *n*-ésima iteração do algoritmo, então o vetor de pesos não é corrigido. Ou seja,

$$\begin{cases} \mathbf{w}(n+1) = \mathbf{w}(n) \quad , \text{ se } \mathbf{w}(n)^{\mathrm{T}}\mathbf{x}(n) > 0 \text{ e } \mathbf{x}(n) \text{ pertence à classe } C_1 \\ \mathbf{w}(n+1) = \mathbf{w}(n) \quad , \text{ se } \mathbf{w}(n)^{\mathrm{T}}\mathbf{x}(n) \leq 0 \text{ e } \mathbf{x}(n) \text{ pertence à classe } C_2 \end{cases} \tag{2.15}$$

2.3. Aprendizado

- **Estratégia de aprendizado**

2. Caso contrário, o vetor de pesos é atualizado de acordo com

$$\begin{cases} \mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n)\,\mathbf{x}(n) & \text{, se } \mathbf{w}(n)^{\mathrm{T}}\,\mathbf{x}(n) > 0 \text{ e } \mathbf{x}(n) \text{ pertence à classe } C_2 \\ \mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\,\mathbf{x}(n) & \text{, se } \mathbf{w}(n)^{\mathrm{T}}\,\mathbf{x}(n) \leq 0 \text{ e } \mathbf{x}(n) \text{ pertence à classe } C_1 \end{cases} \quad (2.16)$$

na qual $\eta(n)$ é a taxa de aprendizagem, que controla o ajuste aplicado ao vetor de pesos na iteração n .

Se $\eta(n) = \eta > 0$, na qual η é uma constante, temos uma regra de adaptação com incremento fixo para o perceptron.

2.3. Aprendizado

- **Usando-se a função sinal como ativação e considerando a taxa de aprendizagem fixa, podemos simplificar a regra de ajuste de pesos através de**

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta\,[d(n) - y(n)]\,\mathbf{x}(n) \tag{2.17}$$

na qual

$$y(n) = \varphi\left(\mathbf{w}(n)^{\mathrm{T}}\mathbf{x}(n)\right) = \begin{cases} +1 & \text{se } \mathbf{w}(n)^{\mathrm{T}}\mathbf{x}(n) > 0 \\ -1 & \text{se } \mathbf{w}(n)^{\mathrm{T}}\mathbf{x}(n) \leq 0 \end{cases}$$

$$d(n) = \begin{cases} +1 & \text{, se } \mathbf{x}(n) \text{ pertence à classe } C_1 \\ -1 & \text{, se } \mathbf{x}(n) \text{ pertence à classe } C_2 \end{cases} \tag{2.18}$$

2.3. Aprendizado

- **Regra de adaptação com incremento fixo**
  - Proposta por Rosenblatt em 1958
  - Guarda uma certa semelhança com a Regra Delta
    - » A Regra Delta utiliza o conceito do gradiente descendente do erro médio quadrático

2.3. Aprendizado

Algoritmo perceptron ($\mathbf{X_T}$, $\mathbf{d_T}$, η)

// inicialização dos pesos
$\mathbf{w} \leftarrow \mathbf{0}$
$n \leftarrow 1$

faça

// ativação do neurônio (saída)
$y(n) \leftarrow \varphi (\mathbf{w}^T \mathbf{x}(n))$

// ajuste de pesos
$\mathbf{w} \leftarrow \mathbf{w} + \eta [d(n) - y(n)] \mathbf{x}(n)$

$n \leftarrow n+1$

enquanto (erro na classificação não é aceitável)

2.3. Aprendizado

- **No algoritmo anterior,**
  - O conjunto de treinamento *T* tem *N* padrões. Ou seja

$$\mathbf{X_T} = \begin{bmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathrm{M} \\ \mathbf{x}_N^{\mathrm{T}} \end{bmatrix} \qquad \mathbf{d_T} = \begin{bmatrix} d_1 \\ \mathrm{M} \\ d_N \end{bmatrix} \tag{2.19}$$

  - O vetor de entradas **x**(*n*) e a saída desejada *d*(*n*) podem ser obtidas a partir de $\mathbf{X_T}$ e $\mathbf{d_T}$ através de

$$\mathbf{x}(n) = \mathbf{x}_t \quad , \quad d(n) = d_t$$

    na qual $t = \mathrm{mod}(n\text{-}1, N)+1$

  - O erro é dado por

$$e(n) = d(n) - y(n) \tag{2.20}$$

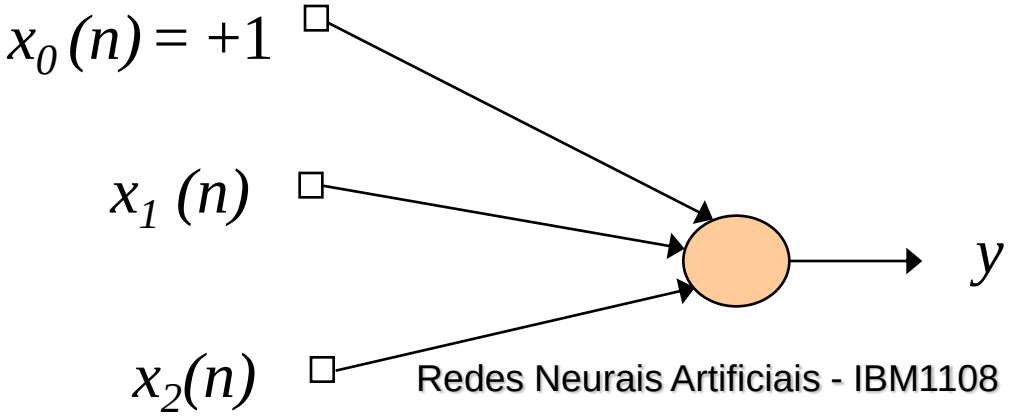
2.3. Aprendizado

- **Escolha da taxa de aprendizagem η**
  - Deve se restrita ao intervalo $0 < \eta \leq 1$
  - η baixa:
    - » adaptação mais lenta
    - » a mudança dos pesos é mais estável
  - η alta:
    - » adaptação mais rápida
    - » a mudança dos pesos é menos estável

2.3. Aprendizado

<u>Exemplo 2.</u> Treine o *perceptron* para o problema de classificação da função AND

Padrão	x_1	x_2	Classe	d
1	0	0	C_1	1
2	0	1	C_1	1
3	1	0	C_1	1
4 (N)	1	1	C_2	-1

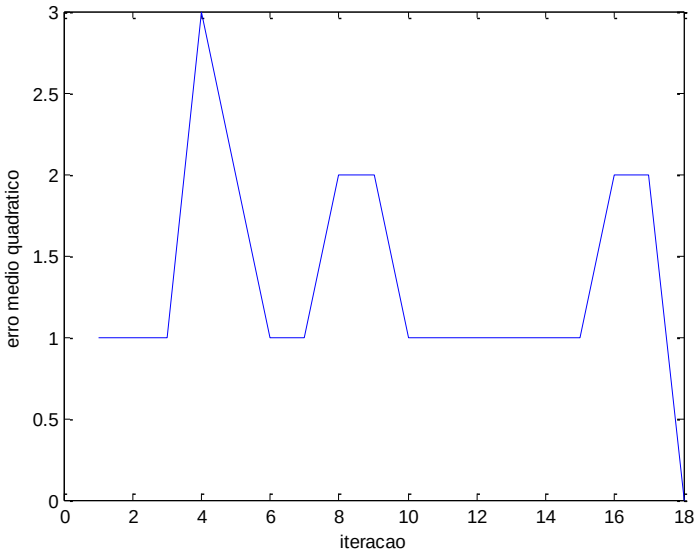


2.3. Aprendizado

Exemplo 2. função AND

- Taxa de aprendizagem

  » η = 0,01



Pesos

Columns 1 through 8

0	0.0200	0.0200	0.0200	0	0.0200	0.0400	0.0400
0	0	0	0	-0.0200	-0.0200	-0.0200	-0.0200
0	0	0	0	-0.0200	-0.0200	0	0

Columns 9 through 16

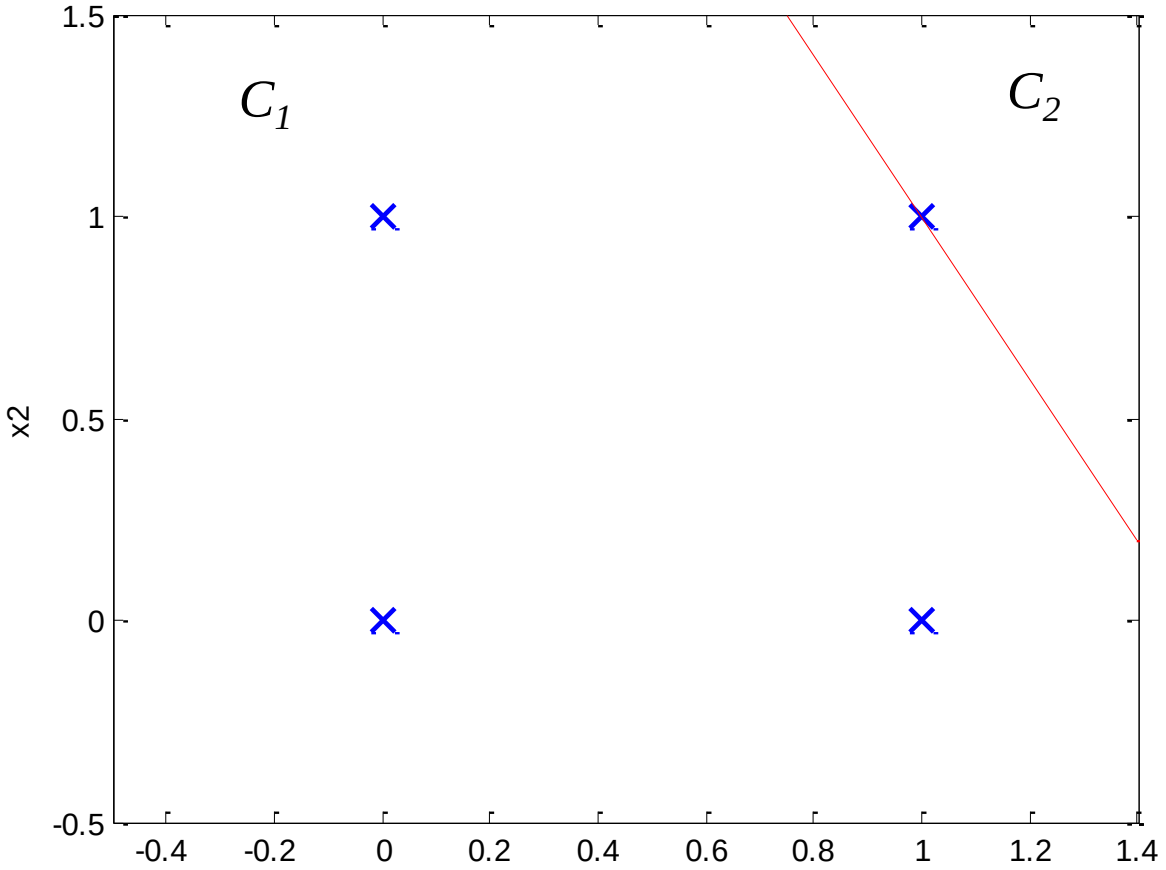
0.0200	0.0200	0.0400	0.0600	0.0400	0.0400	0.0400	0.0600
-0.0400	-0.0400	-0.0400	-0.0200	-0.0400	-0.0400	-0.0400	-0.0200
-0.0200	-0.0200	0	0	-0.0200	-0.0200	-0.0200	-0.0200

Columns 17 through 19

0.0400	0.0400	0.0600
-0.0400	-0.0400	-0.0400
-0.0400	-0.0400	-0.0200

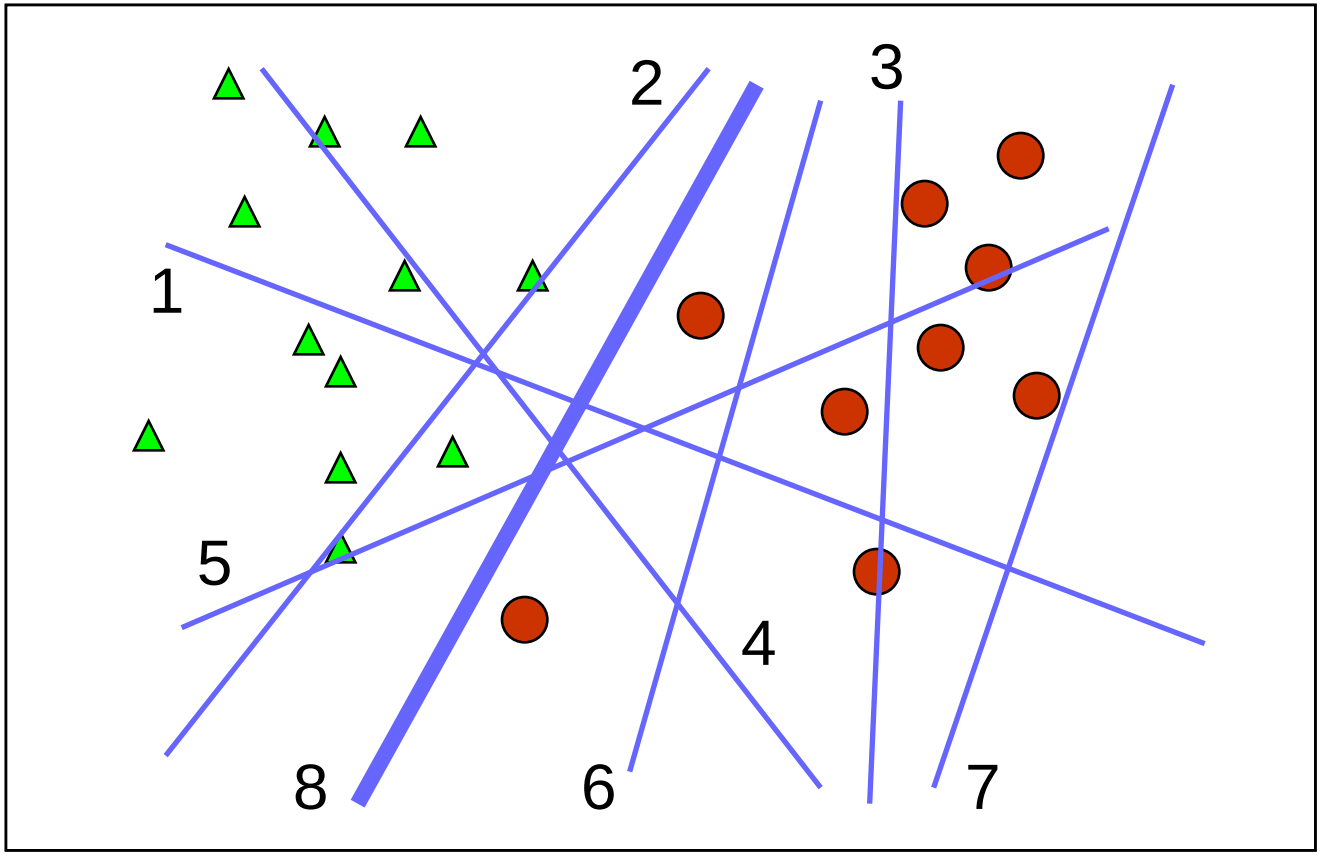
Exemplo 2. função AND

Iteração 18: Erro médio quadrático = 0



2.3. Aprendizado

Exemplo 2.3.



2.3. Aprendizado

<u>Exercício 2.1.</u> Treine o *perceptron* para o problema de classificação da função OR

Padrão	x_1	x_2	Classe	d
1	0	0	C_1	1
2	0	1	C_2	-1
3	1	0	C_2	-1
4(N)	1	1	C_2	-1

2.3. Aprendizado

- **A convergência da regra de adaptação com incremento fixo para o *perceptron* pode ser provada**
  - As provas de convergência são em geral apresentadas para η igual a 1
  - O valor de η não é importante, desde que seja positivo
    - » Valores de η diferentes de 1 meramente escalam os vetores de entrada sem afetar sua separabilidade

2.4. Teorema da Convergência

- **Teorema da Convergência para a regra de adaptação com incremento fixo**

Sejam os subconjuntos de vetores de treinamento T_1 e T_2 linearmente separáveis. Suponha que as entradas apresentadas ao perceptron se originem destes dois subconjuntos. O perceptron converge após n_o iterações, significando que

$$\boldsymbol{w}(n_o) = \boldsymbol{w}(n_o+1) = \boldsymbol{w}(n_o+2) = \ldots$$

é a solução para $n_o \leq n_{max}$.

Prova: ver [HAYKIN, 2001], Seção 3.1, p. 163-169.

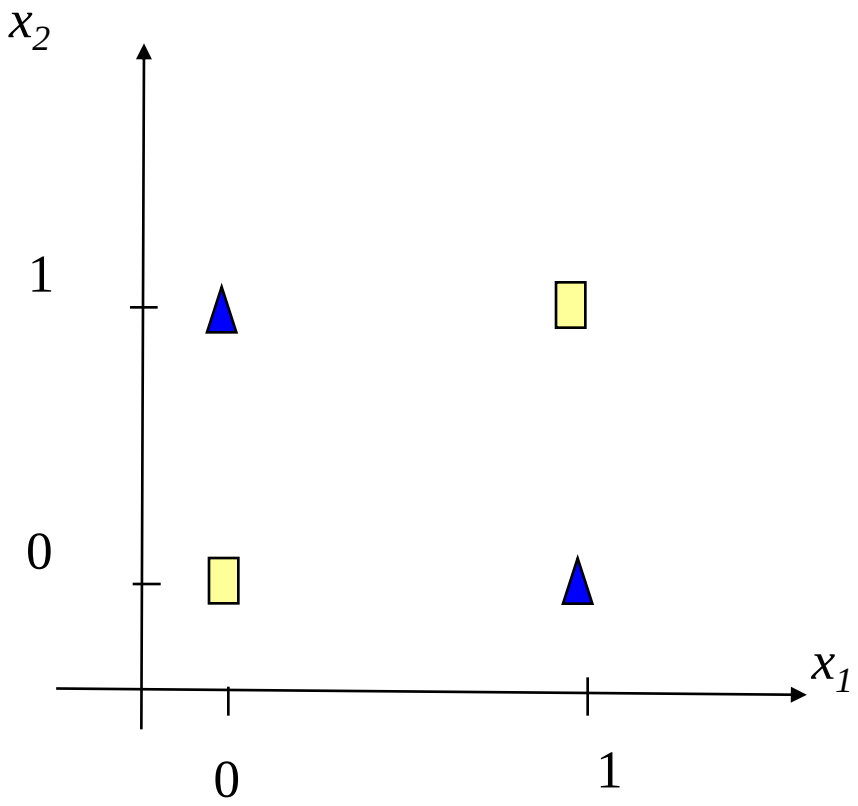
2.5. Alguns Aspectos do Problema de Reconhecimento de Padrões

Exemplo 2.3. Problema de classificação da função XOR

Padrão	x_1	x_2	Classe	d
1	0	0	C_2	-1
2	0	1	C_1	1
3	1	0	C_1	1
4 (N)	1	1	C_2	-1

2.5. Alguns Aspectos do Problema de Reconhecimento de Padrões

<u>Exemplo 2.3.</u> Problema de classificação da função XOR



2.5. Alguns Aspectos do Problema de Reconhecimento de Padrões

- **O *perceptron* não consegue resolver problemas de classificação que não são linearmente separáveis**
  - Exemplo: função XOR
  - Este problema advém do fato de cada *perceptron* gerar apenar um hiperplano (e conseqüentemente uma FD)
  - É possível gerar mais de um hiperplano se utilizarmos mais de um *perceptron* (neurônio) em uma única camada
    - » No entanto, ainda teríamos o problema de como associar as diferentes regiões produzidas pelas FD com as diferentes classes
  - Então, como seria possível resolver este problema?

Comentários

- **Referências**
  - **Haykin, S. S.. *Redes neurais: princípios e prática.* 2ª ed., Bookman, 2001.**
    - » Capítulo 3
  - **Príncipe, J. C.; Euliano, N. R. & Lefebvre, W. C. *Neural and Adaptive Systems: Fundamentals Through Simulations*. John Wiley & Sons, Inc. 2000**
    - » Seções 3.1 – 3.3
  - **Braga, A.P.; Carvalho, C.P.L.F. & Ludermir, T.B.. *Redes neurais artificiais: Teoria e Aplicações.* LTC, 2000.**
    - Capítulo 3