Sistemas Inteligentes

SVM NÃO LINEAR

SVM para Classificação de Padrões

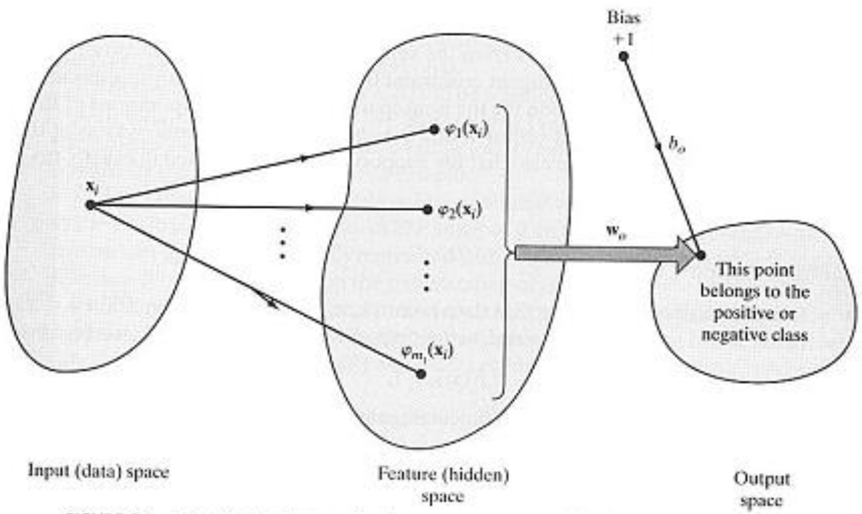


FIGURE 6.4 Illustrating the two mappings in a support vector machine for pattern classification: (i) nonlinear mapping from the input space to the feature space; (ii) linear mapping from the feature space to the output space.

Em alguns casos, classificadores lineares não são efetivos

Solução: Mapear dados em um espaço de maior dimensão (através de funções não-lineares) e realizar a classificação linear no espaço transformado (características)

- Mapear dados

$$\mathbf{x} \rightarrow \Phi(\mathbf{x})$$

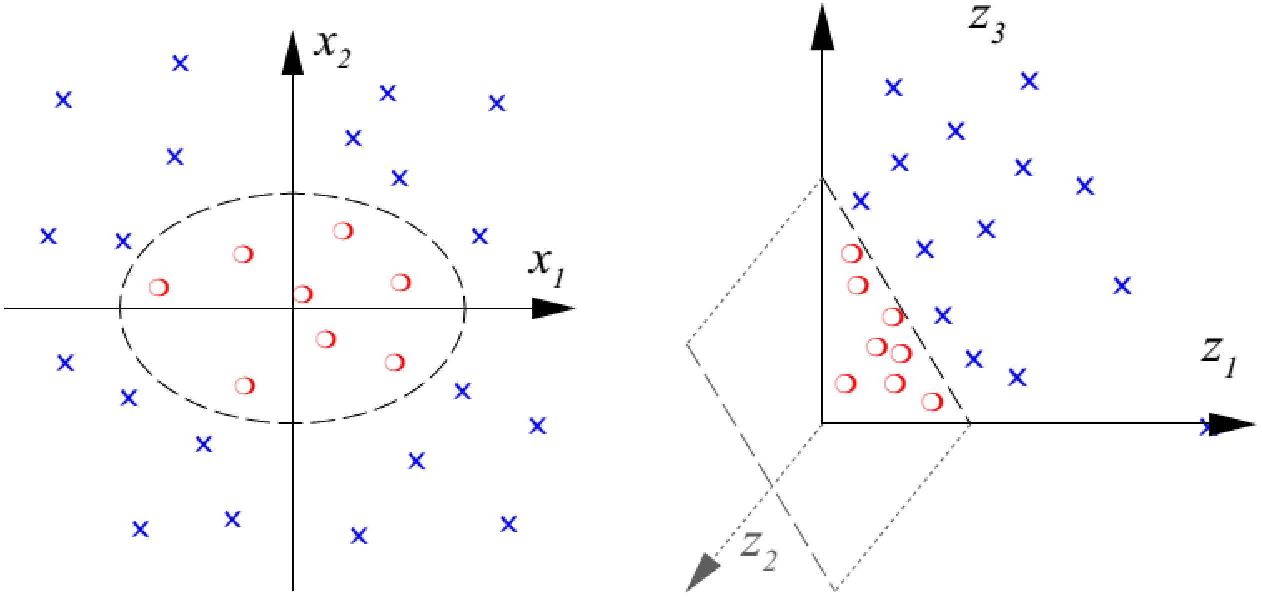
- Realizar classificação

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\Phi(\mathbf{x}) + b$$

SVM: Mapeamento Polinomial

$$\Phi : R^2 \rightarrow R^3$$

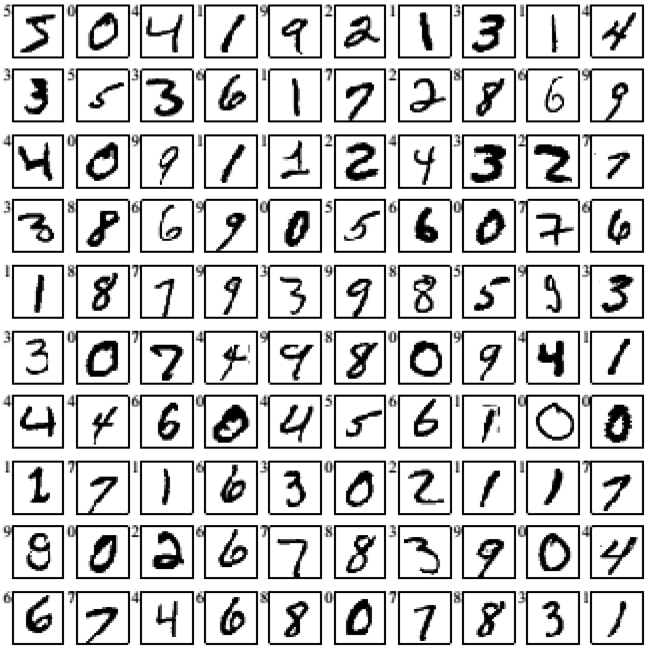
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(}2)x_1x_2, x_2^2)$$



SVM – Exemplo

Banco de dados MNIST (escrita manuscrita)

- 60.000 exemplos de treinamento
- 10.000 exemplos para teste (28x28)
- SVM Linear: erro de 8.5%
- SVM Polinomial: erro de 1%



Problema Primal: Otimização com restrições

Conforme visto anteriormente, obter o conjunto de parâmetros ótimos da SVM linear depende da solução de um problema de otimização definido por

$$\min \left\{ \Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} \right\}$$

Sujeito à

$$d_i(\mathbf{w}_o^T\mathbf{x}_i + b_0) \geq 1$$

para $i = 1,2, \ldots, N$

Problema Dual

O problema de otimização original possui um problema dual, que leva à mesma solução do problema original.

Dado o conjunto de treinamento $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$, encontre os multiplicadores $\{\alpha_i\}_{i=1}^{N}$ que **maximizam** a seguinte função objetivo:

$$Q(\alpha) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j d_i d_j \mathbf{x}_i^T\mathbf{x}_j + \sum_{i=1}^{N}\alpha_i$$

sujeito à

$$\sum_{i=1}^{N}\alpha_i d_i = 0 \quad \text{e} \quad \alpha_i \geq 0, \text{ para todo } i = 1,2,\dots,N$$

O problema é colocado inteiramente em termos dos dados de treinamento e dos produtos internos $\mathbf{x}_i^T\mathbf{x}_j$.

Solução do Problema de Otimização

Uma vez que os multiplicadores $\alpha_{o,i}$ são obtidos, podemos calcular

$$\mathbf{w}_o = \sum_{i=1}^{N} \alpha_{o,i} d_i \mathbf{x}_i$$

e quando $\mathbf{x}_i$ é um vetor suporte

$$b_o = d^{(s)} - \mathbf{w}_o^T \mathbf{x}^{(s)}$$

Note que não seria necessário o cálculo explícito do vetor $\mathbf{w}_o$, já que

$$\mathbf{w}_o^T \mathbf{x} = \sum_{i=1}^{N} \alpha_{o,i} d_i \mathbf{x}_i^T \mathbf{x}$$

SVM não linear

Nesse caso, as entradas $\mathbf{x}$ são mapeadas por meio de uma função não linear $\varphi(\mathbf{x})$

Considerando o vetor de pesos $\mathbf{w}$ (que inclui o bias b), a superfície de decisão no espaço de características (considere $\varphi_0(\mathbf{x}) = 1$):

$$\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) = 0$$

Seguindo os mesmos passos do SVM linear, obtemos

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i d_i \boldsymbol{\varphi}(\mathbf{x}_i)$$

Inner-Product Kernel

Combinando os dois resultados, obtemos a superfície de decisão como sendo

$$\sum_{i=1}^{N} \alpha_i d_i \, \boldsymbol{\varphi}^T(\mathbf{x}_i)\boldsymbol{\varphi}(\mathbf{x}) = 0$$

Note que a superfície depende do produto interno $\boldsymbol{\varphi}^T(\mathbf{x}_i)\boldsymbol{\varphi}(\mathbf{x})$ de vetores no espaço de característica.

O cálculo do produto interno pode ser simplificado por meio de um *Inner-Product Kernel* $K(\mathbf{x}, \mathbf{x}_i)$:

$$K(x, x_i) = \boldsymbol{\varphi}^T(\mathbf{x})\boldsymbol{\varphi}(\mathbf{x}_i) = \sum_{j=0}^{m_1} \varphi_j(\mathbf{x})\varphi_j(\mathbf{x}_i)$$

onde m_1 é a dimensão do espaço de características.

Inner-Product Kernel

Dessa forma, o hiperplano ótimo será dado por

$$\sum_{i=1}^{N} \alpha_i d_i \, K(\mathbf{x}, \mathbf{x}_i) = 0$$

É possível utilizar um “Truque” para evitar o cálculo explícito de $\boldsymbol{\varphi}(\cdot)$ → *Kernel Trick*

Exemplos de Funções Kernel

Linear: $K(\mathbf{x}, \mathbf{x}_i) = \mathbf{x}^T \mathbf{x}_i$

Polinomial: $K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^T \mathbf{x}_i + 1)^p$

RBF: $K(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}\right)$

Two-Layer Perceptron : $K(\mathbf{x}, \mathbf{x}_i) = \tanh(\beta_0 \mathbf{x}^T \mathbf{x}_i + \beta_1)$

Exemplos de Funções Kernel

$$K(\mathbf{x}, \mathbf{x}_i) = (1 + \mathbf{x}^T \mathbf{x}_i)^2$$

with $\mathbf{x} = [x_1, x_2]^T, \mathbf{x}_i = [x_{i1}, x_{i2}]^T$,

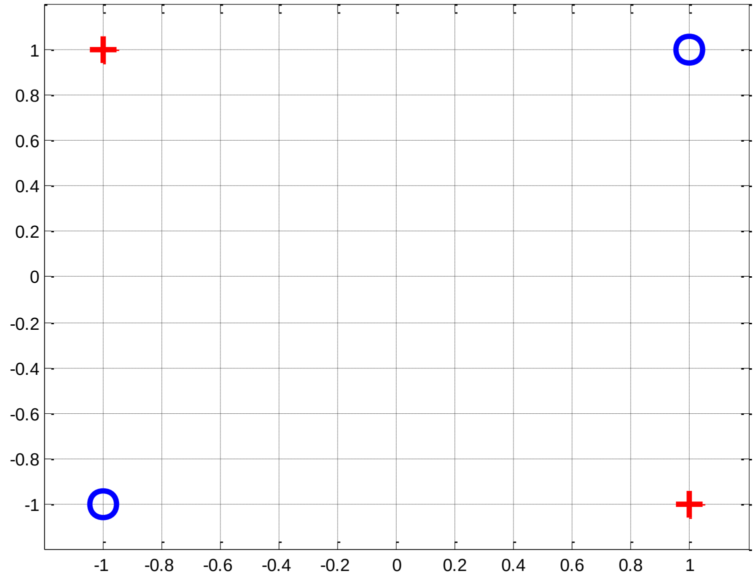
$$\begin{aligned}
K(\mathbf{x}, \mathbf{x}_i) &= 1 + x_1^2 x_{i1}^2 + 2x_1 x_2 x_{i1} x_{i2} \\
&\quad + x_2^2 x_{i2}^2 + 2x_1 x_{i1} + 2x_2 x_{i2} \\
&= [1, x_1^2, \sqrt{2}x_1 x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2] \\
&\quad [1, x_{i1}^2, \sqrt{2}x_{i1} x_{i2}, x_{i2}^2, \sqrt{2}x_{i1}, \sqrt{2}x_{i2}]^T \\
&= \boldsymbol{\varphi}(\mathbf{x})^T \boldsymbol{\varphi}(\mathbf{x}_i),
\end{aligned}$$

where $\boldsymbol{\varphi}(\mathbf{x}) = [1, x_1^2, \sqrt{2}x_1 x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2]^T$.

Exemplo: Problema XOR

Considere o problema de classificação

x_1	x_2	y_d
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	-1



Exemplo: Problema XOR

Considere o kernel

$$K(\mathbf{x}, \mathbf{x}_i) = (1 + \mathbf{x}^T \mathbf{x}_i)^2$$

Conforme vimos anteriormente

$$\mathbf{\Phi}(\mathbf{x}) = \left[1, x_1^2, \sqrt{2}x_1 x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2\right]^T$$

E, equivalentemente,

$$\mathbf{\Phi}(\mathbf{x}_i) = \left[1, x_{i1}^2, \sqrt{2}x_{i1} x_{i2}, x_{i2}^2, \sqrt{2}x_{i1}, \sqrt{2}x_{i2}\right]^T$$

Exemplo: Problema XOR

Dessa forma,

$$Q(\alpha) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2}(9\alpha_1^2 - 2\alpha_1\alpha_2 - 2\alpha_1\alpha_3 + 2\alpha_1\alpha_4 + 9\alpha_2^2 + 2\alpha_2\alpha_3 - 2\alpha_2\alpha_4 + 9\alpha_3^2 - 2\alpha_3\alpha_4 + 9\alpha_4^2)$$

Exemplo: Problema XOR

A otimização em relação aos parâmetros α leva a

$$
\begin{gathered}
9\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4 = 1 \\
-1\alpha_1 + 9\alpha_2 + \alpha_3 - \alpha_4 = 1 \\
-\alpha_1 + \alpha_2 + 9\alpha_3 - \alpha_4 = 1 \\
\alpha_1 - \alpha_2 - \alpha_3 + 9\alpha_4 = 1
\end{gathered}
$$

Ou seja

$$\alpha_{o,1} = \alpha_{o,2} = \alpha_{o,3} = \alpha_{o,4} = \frac{1}{8}$$

i.e., todos vetores são vetores suporte.

Exemplo: Problema XOR

Assim

$$Q_o(\boldsymbol{\alpha}) = \frac{1}{4} = \frac{1}{2}\|\mathbf{w}_o\|^2 \rightarrow \|\mathbf{w}_o\| = \frac{1}{\sqrt{2}}$$

$$\mathbf{w}_o = \sum_{i=1}^{N} \alpha_{o,i} d_i \Phi(\mathbf{x}_i) \rightarrow \mathbf{w}_o = \frac{1}{8}[-\Phi(\mathbf{x}_1) + \Phi(\mathbf{x}_2) + \Phi(\mathbf{x}_3) - \Phi(\mathbf{x}_4)]$$

$$= \begin{bmatrix} 0 \\ 0 \\ -1/\sqrt{2} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\boldsymbol{\Phi}(\mathbf{x}_i) = \left[1, x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2, \sqrt{2}x_{i1}, \sqrt{2}x_{i2}\right]^T$$

$$b_o = 1 - \mathbf{w}_o^T \Phi(\mathbf{x}_i) \rightarrow b_o = 0$$

Exemplo: Problema XOR

O hiperplano ótimo é dado então por

$$\mathbf{w}_o^T \mathbf{\Phi}(\mathbf{x}) = 0$$

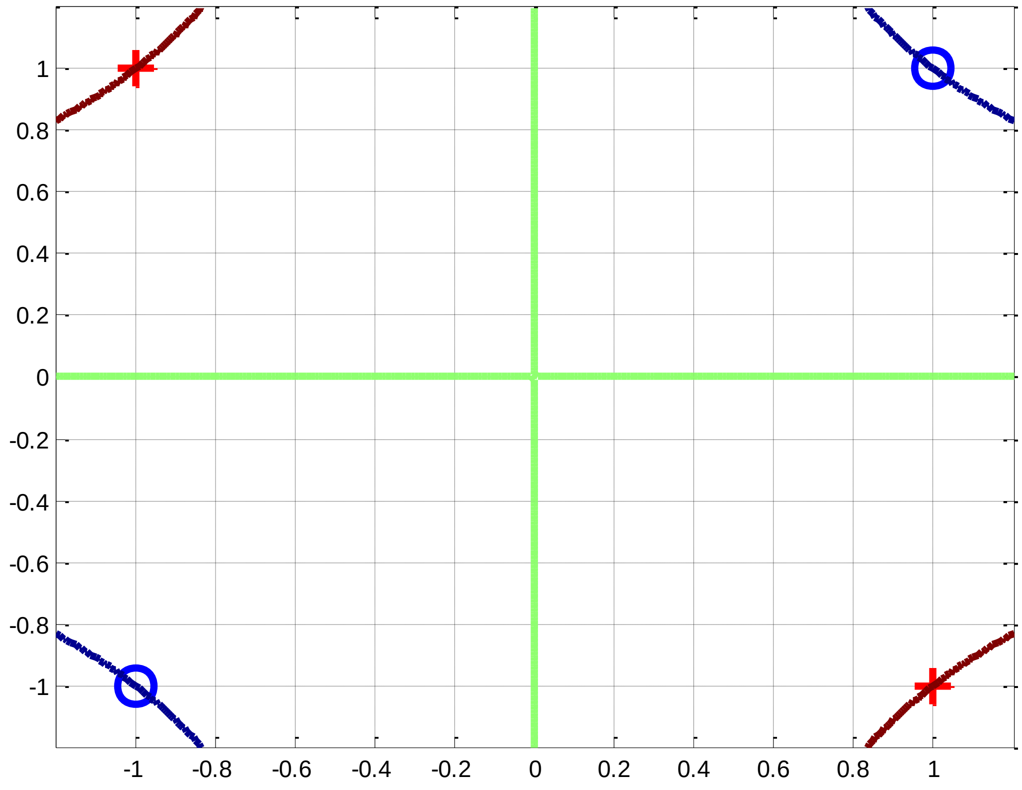
$$\downarrow$$

$$\begin{bmatrix} 0 \; 0 \; -\frac{1}{\sqrt{2}} \; 0 \; 0 \; 0 \end{bmatrix} \begin{bmatrix} 1 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \end{bmatrix} = 0$$

$$\downarrow$$

$$-x_1x_2 = 0$$

Exemplo: Problema XOR



Sistemas Inteligentes

REDES NEURAIS ARTIFICIAIS

Redes Neurais Artificiais

De maneira geral, uma Rede Neural Artificial (RNA) é um paradigma de computação tendo como inspiração redes neuronais biológicas (cérebro humano)

Uma Rede Neural Artificial é um *paradigma de computação*

- A estrutura de processamento da informação consiste de um grande número de elementos processadores (neurônios) interconectados e trabalhando de maneira a solucionar problemas específicos

Inspiração Biológica

Animais são capazes de reagir a mudanças em seu meio ambiente, adaptando seu sistema nervoso para modificar seu comportamento.

O sistema nervoso é composto por unidades de processamento simples, os neurônios – alguns bilhões de neurônios interconectados. Cada neurônio pode estar conectado a outros por meio de sinapses.

Quando um desses neurônios "dispara"

- Pulso é "transmitido" aos neurônios interconectados, e pode ocasionar o disparo das demais células.

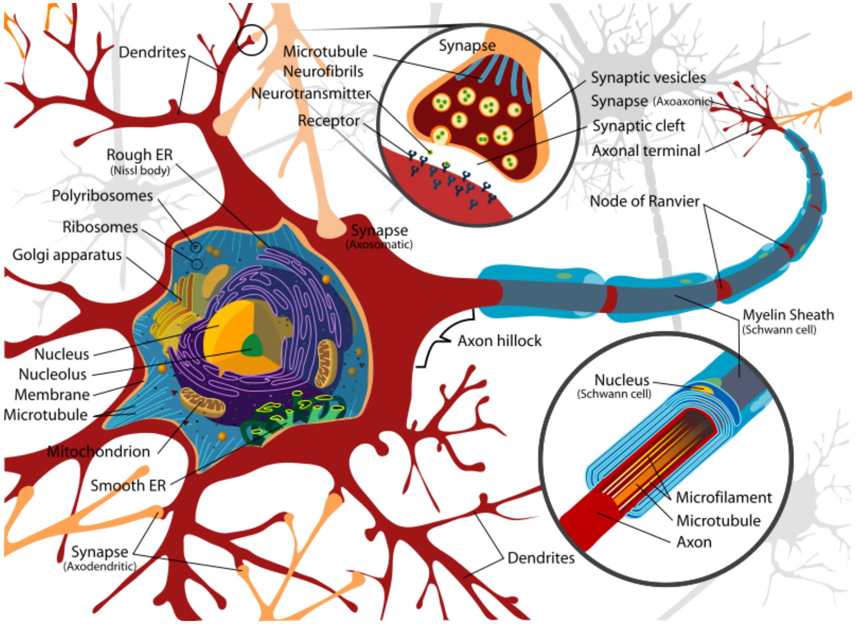
Neurônio Biológico e as Sinapses

Acredita-se ser impossível que o código genético de um indivíduo seja capaz de conduzir todo o processo de organização topológica do cérebro. Apenas aspectos gerais dos circuitos envolvidos devem estar codificados geneticamente.

Há um expressivo aumento na densidade de conexões sinápticas da vida embrionária até a idade de 2 anos.

Esse processo de ampliação e redução de sinapses, contudo, não é homogêneo, pois nas regiões sensório-motoras este processo ocorre mais cedo, enquanto que ele é retardado em áreas associadas aos processos cognitivos.

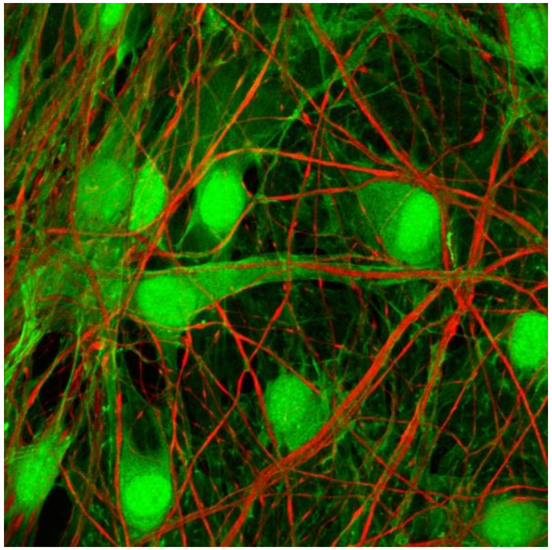
A redução de sinapses é dramática: o número de sinapses ao término da puberdade pode chegar a 50% do número existente com a idade de 2 anos. Há uma perda de até 100.000 sinapses por segundo na adolescência.

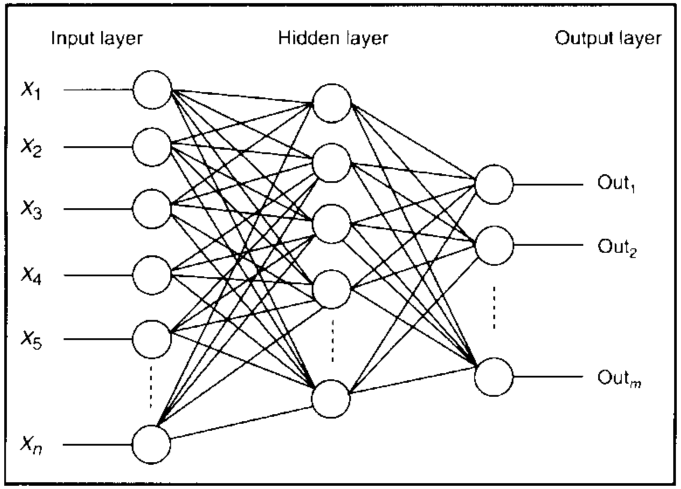


Aprendizado

A modificação das sinapses está ligada à ideia de treinamento. Em outras palavras, a rede neural deve aprender a partir de exemplos

- Reconhecemos um cachorro mesmo sem ter visto todas as possíveis raças existentes → existe, portanto, um processo de generalização além do conjunto de dados utilizados para treinamento.





Redes Neurais Artificiais (RNA)

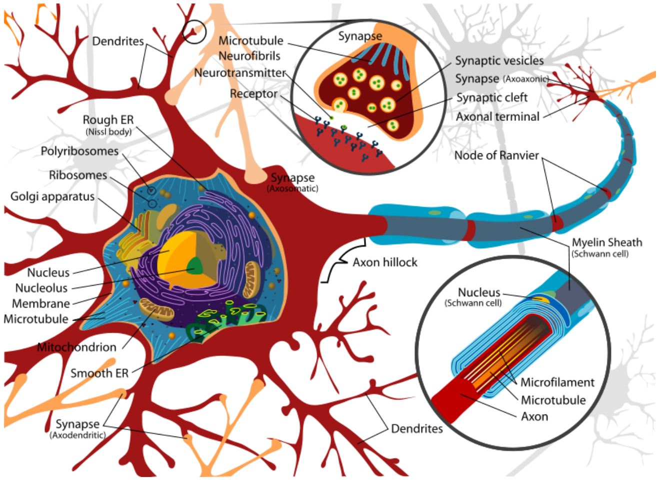
Diversas características em comum com o sistema nervoso:

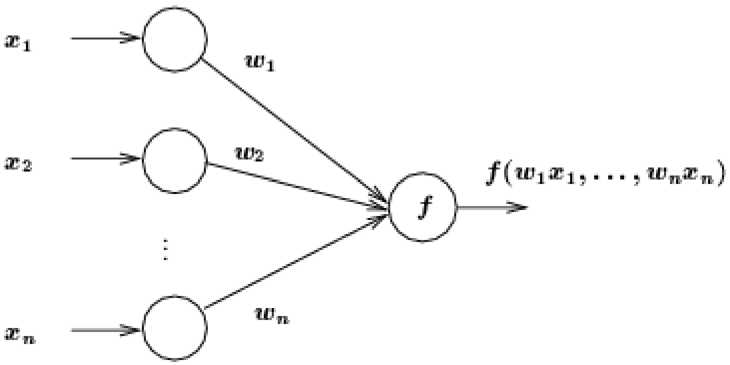
- O processamento básico de informação ocorre em diversas unidades simples denominadas de *neurônios artificiais* ou simplesmente *neurônios* (ou *nós*);
- Os neurônios estão interconectados, o que dá origem a redes de neurônios ou redes neurais;
- A informação (sinais) é transmitida entre neurônios através de conexões ou sinapses;
- A eficiência de uma sinapse, representada por um *peso* associado, corresponde à informação armazenada pelo neurônio e, portanto, pela rede neural;
- O conhecimento é adquirido do ambiente através de um processo de *aprendizagem* que é, basicamente, responsável por adaptar os pesos das conexões aos estímulos recebidos do ambiente.

Modelos de Neurônios

- Cada nó possui uma ou mais entradas, e uma saída
- Valores de entrada e saída podem ser
  - Binários {0, 1}
  - Bipolares {-1, 1}
  - Contínuos
- Todas entradas são sincronizadas – permanecem ativas até que a saída seja produzida
- Pesos são atribuídos às conexões
- f (net) representa a função de ativação

$$net = \sum_{i=1}^{n} w_i x_i$$





Arquiteturas de Redes Neurais

Redes Totalmente interconectadas

Redes em Camadas

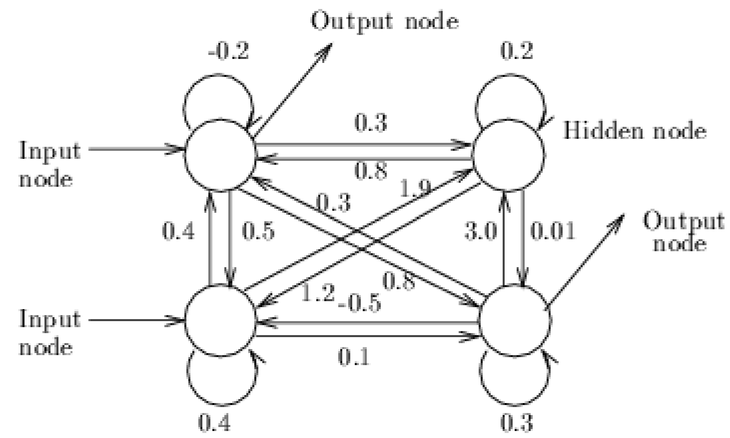
Redes *Feedforward*

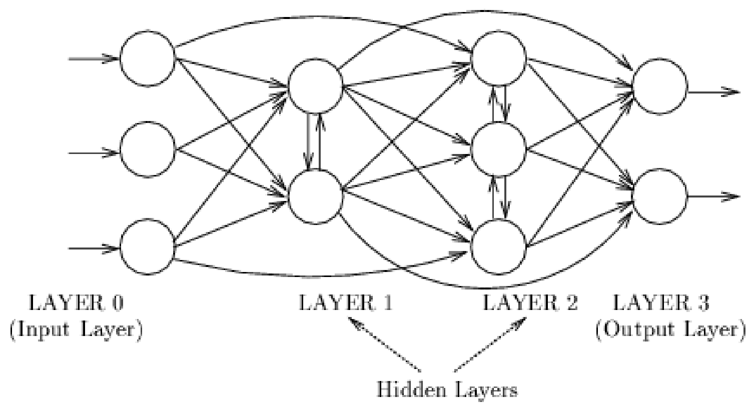
Redes Recorrentes

- Comportamento dinâmico

Redes Modulares

Outras arquiteturas





Aprendizagem

Em nosso caso, aprender a solução equivale a ajustar adequadamente os parâmetros da rede

- De maneira supervisionada
  - Backpropagation
  - Aprendizado por reforço
- De maneira não-supervisionada
  - Agrupamento de dados (Clustering)
  - Redução de dimensionalidade → compressão
    - Principal Component Analysis

Redes Neurais Artificiais - Histórico

Desenvolvimento das redes neurais artificiais data do início do anos 1940 – "Idade da Ilusão"

- McCulloch & Putts (1943)
  - Proposta do modelo do neurônio artificial
- Wiener (1948)
  - Cibernética
- Hebb (1949)
  - Princípio de Aprendizagem em redes neurais
- Widrow & Hoff (1960)
  - Algoritmo para treinamento de redes neurais
- Rosenblatt (1958)
  - Perceptron
- Minsky and Papert (1969)
  - Críticas duras às Redes Neurais, mostrando limitações dos modelos propostos

Redes Neurais Artificiais - Histórico

1969-1984:“Idade das Trevas”

- Poucas publicações envolvendo redes neurais artificiais

1982-hoje: “Renascimento”

- Hopfield (1982)
  - Redes de Hopfield – circuitos de memórias
- Rumelhart , Hinton & Williams(1986)
  - Algoritmo de treinamento – Back Propagation

Por que utilizar Redes Neurais Artificiais?

Aprendizagem Adaptativa

- Habilidade de aprender como realizar determinada tarefa a partir de dados disponíveis para treinamento ou uma experiência inicial

Auto-Organização

- Uma RNA pode criar sua própria organização ou representação da informação que ela recebe durante o período de treinamento.

Processamento Paralelo

- O processamento em uma RNA é feito em paralelo, uma tendência que têm chamado muito a atenção de pesquisadores e desenvolvedores tendo em vista os avanços alcançados com a computação paralela.

Tolerância à falha por meio de codificação redundante da informação

- A destruição de partes da rede pode piorar a performance, mas algumas capacidades ainda podem ser mantidas.

Aplicações de Redes Neurais (algumas dentre várias)

Reconhecimento de Padrões/Classificação

- Identificação de usuário
- Reconhecimento de fala

Associação de dados

- Não apenas identificar caracteres em um texto escaneado, mas também identificar quando o scanner não está operando corretamente

Predição

- Bolsa de valores
- Previsão do tempo

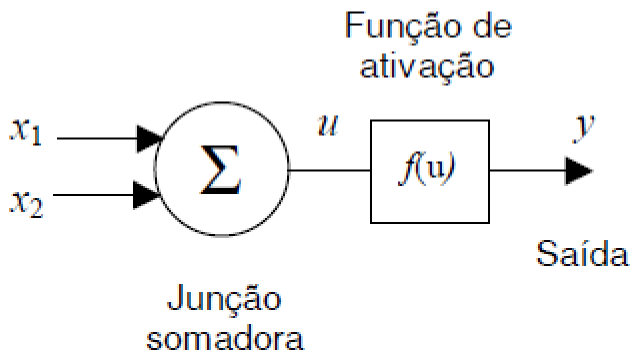
Sistemas Inteligentes

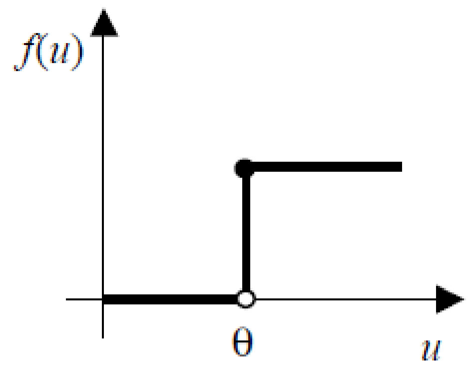
REDES NEURAIS ARTIFICIAIS – PERCEPTRON

Neurônios Artificiais- McCulloch e Pitts

McCulloch e Pitts (1943)

- Exemplo de operação: funções lógicas OR e AND.





Neurônios Artificiais - "Integrate and Fire"

Modelos clássicos em neurociência computacional.

Modelos de tempo contínuo

$$\tau_m \frac{du(t)}{dt} = u_{res} - u(t) + R_m i(t)$$

onde

τ_m é a constante de tempo da membrana

u_{res} é o potencial de repouso

$i(t)$ é a soma das correntes geradas pelos neurônios pré-sinápticos

R_m é a resistência do neurônio à corrente

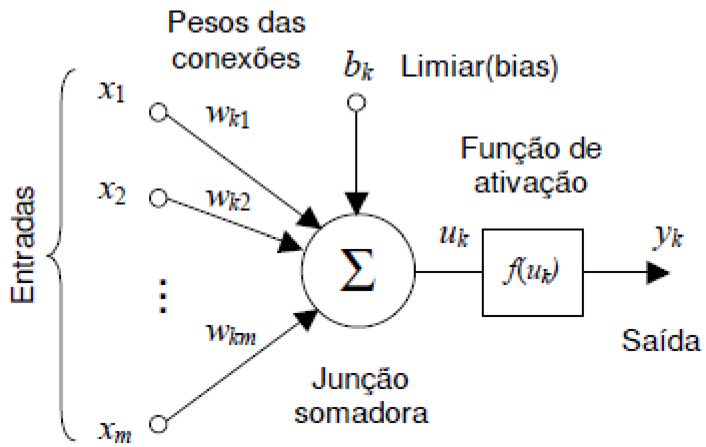
Neurônios Artificiais – Modelo Genérico

Modelo Geral é composto por:

- as sinapses, caracterizadas pelos seus pesos associados;
- a junção somadora; e
- a função de ativação

A função de ativação tem dois propósitos:

- limitar a saída do neurônio
- introduzir não-linearidade no modelo



Neurônios Artificiais – Modelo Genérico

O limiar b_k (ou θ_k) tem o papel de aumentar ou diminuir a influência do valor da entrada líquida para a ativação do neurônio k.

Para o neurônio de McCulloch e Pitts

$$y = \begin{cases} 0, se\ u < \theta \\ 1, se\ u \geq \theta \end{cases} \text{onde } u = \sum_i w_i x_i$$

ou ainda

$$y = \begin{cases} 0, se\ u < 0 \\ 1, se\ u \geq 0 \end{cases} \text{onde } u = \sum_i w_i x_i - \theta$$

Neurônios Artificiais – Modelo Genérico

A saída do neurônio *k* pode ser descrita por:

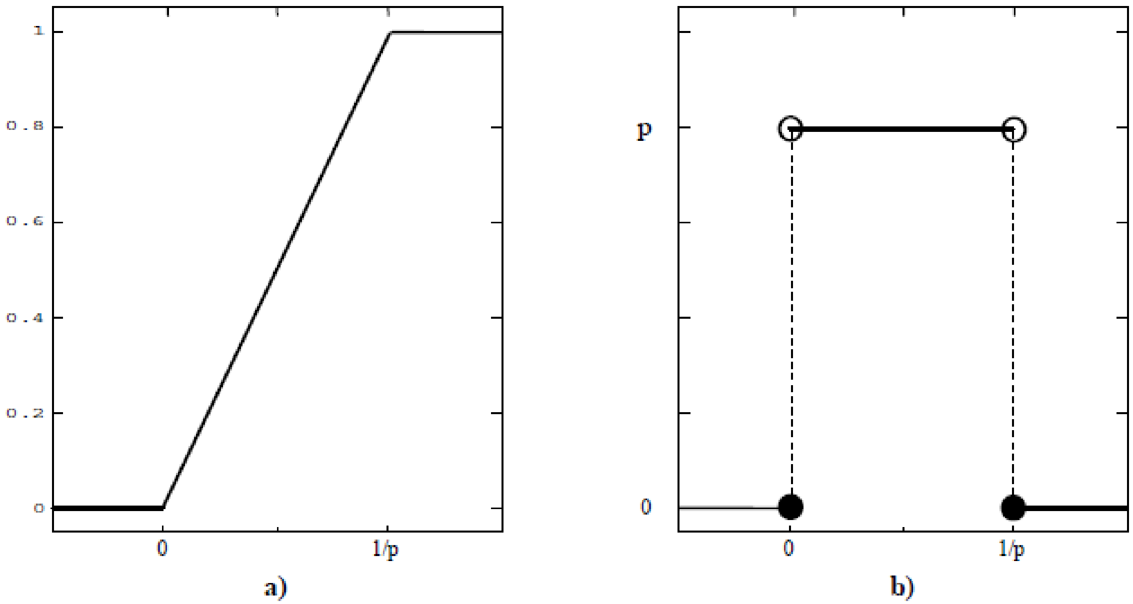
$$y_k = f(u_k) = f\left(\sum_{j=1}^{m} w_{kj} x_j + b_k\right)$$

É possível simplificar a notação acima de forma a incluir o bias simplesmente definindo um sinal de entrada de valor $x_0 = 1$ (ou -1) com peso associado $w_{k0} = b_k$

Função de Ativação

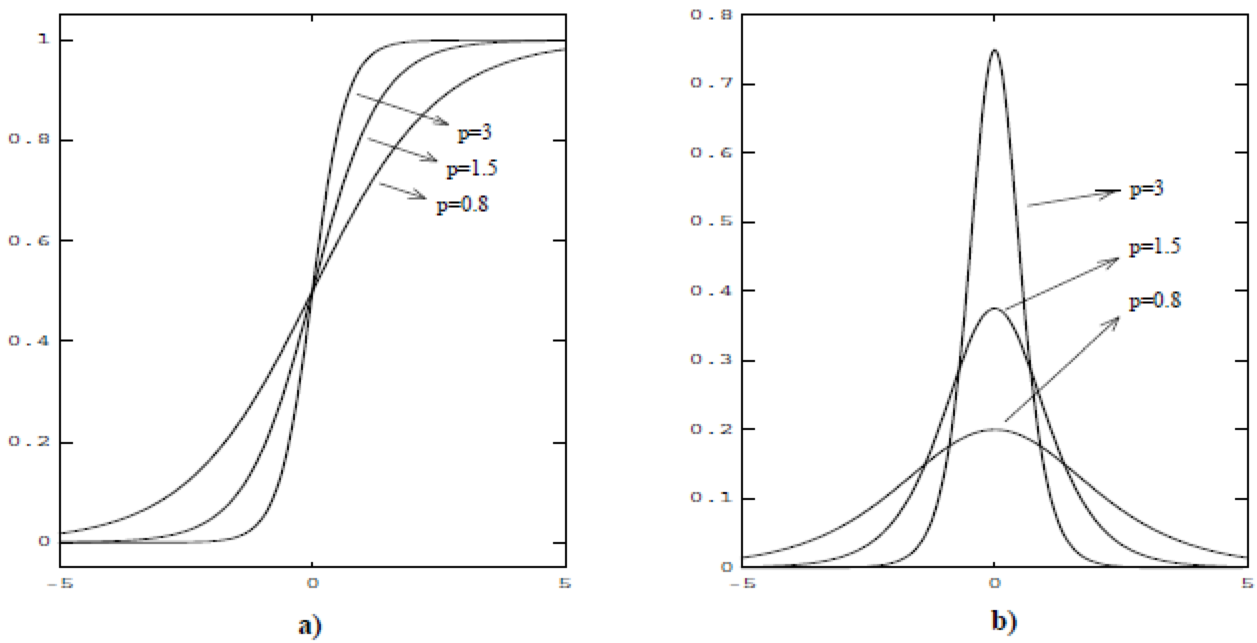
$$f(\mathbf{u}_k) = \begin{cases} 1 & \text{se} \quad p\mathbf{u}_k \geq 1 \\ p\mathbf{u}_k & \text{se} \quad 0 < p\mathbf{u}_k < 1 \\ 0 & \text{se} \quad p\mathbf{u}_k \leq 0 \end{cases}$$

com p constante e positivo.



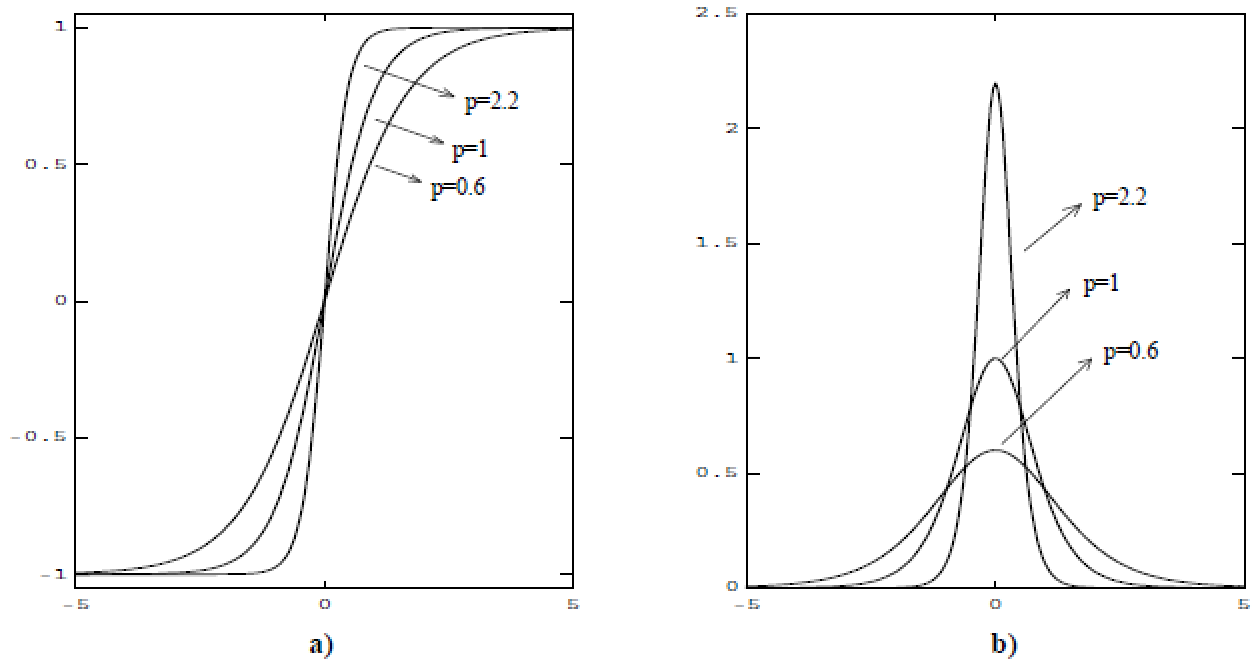
Função semi-linear (a) e sua derivada em relação à entrada interna (b)

Função de Ativação



Função logística (a) e sua derivada em relação à entrada interna (b)

Função de Ativação



Função tangente hiperbólica (a) e sua derivada em relação à entrada interna (b)

Armazenamento da informação - Aprendizado

Nos casos mais simples, o conhecimento é armazenado nos pesos das conexões entre neurônios.

A representação de conhecimento é feita de forma que o conhecimento necessariamente influencie a forma de processamento da rede, ou seja, o seu comportamento de entrada-saída.

Se o conhecimento está armazenado nos pesos das conexões, então o processo de aprendizagem corresponde a identificar um conjunto apropriado de pesos de forma que a rede se comporte como desejado.

Algoritmo de Aprendizado - Perceptron

Considerando apenas um neurônio com n entradas e 1 saída, temos

$$y = \begin{cases} 0, se\ u < 0 \\ 1, se\ u \geq 0 \end{cases} \text{ onde } u = \sum_i w_i x_i - \theta$$

Usando notação matricial, temos

$$u = \mathbf{w}^T \mathbf{x}$$

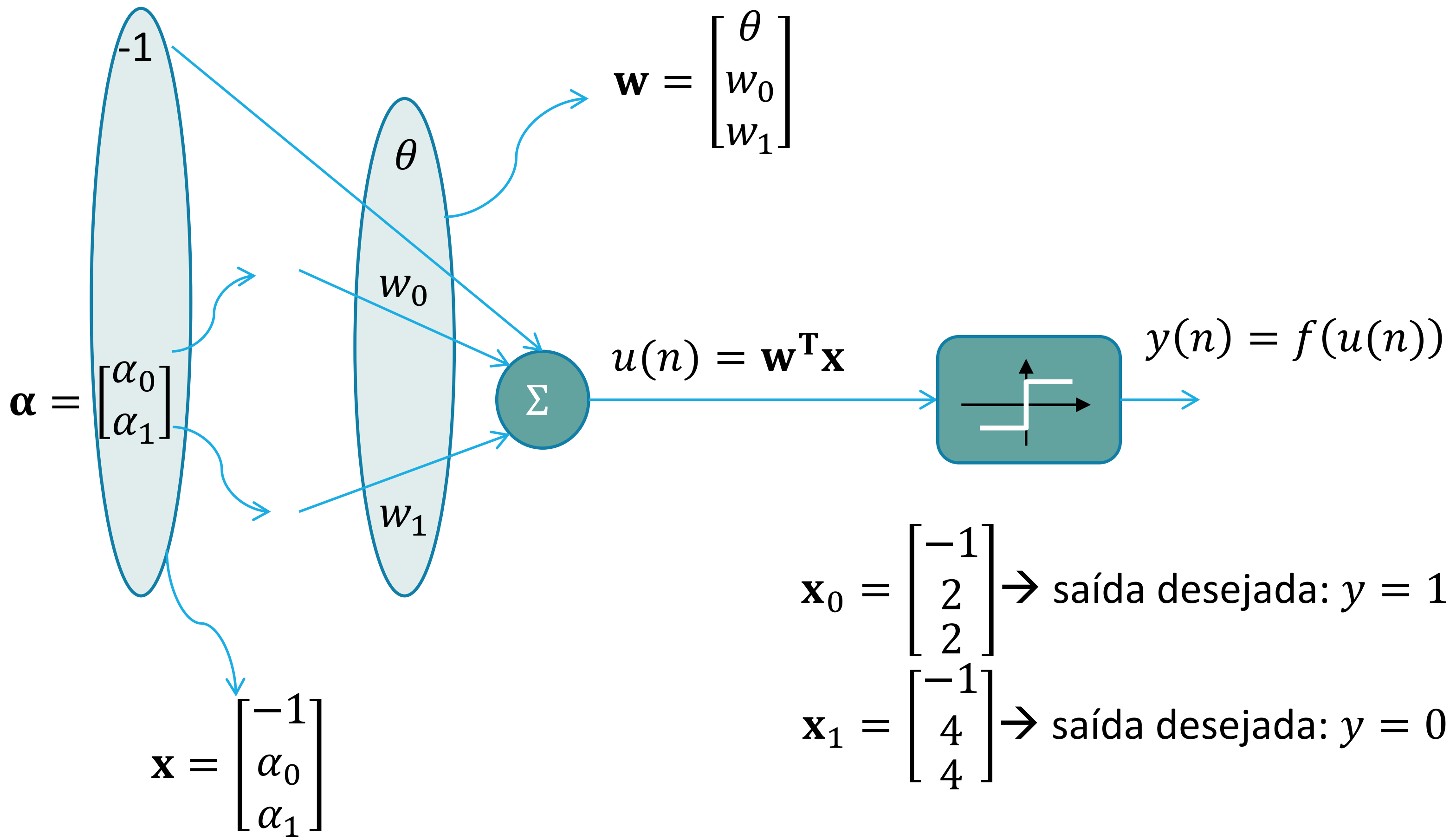
onde

$$\mathbf{w} = \left[\theta, w_1, w_{2,} \cdots, w_n\right]^T \text{ e } \mathbf{x} = [-1, x_1, x_2, \cdots, x_n]^T$$

e, portanto,

$$y = \begin{cases} 0, se\ \mathbf{w}^T \mathbf{x} < 0 \\ 1, se\ \mathbf{w}^T \mathbf{x} \geq 0 \end{cases}$$

Algoritmo de Aprendizado - Perceptron



$\mathbf{x}_0 = \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}$ → saída desejada: $y = 1$

$\mathbf{x}_1 = \begin{bmatrix} -1 \\ 4 \\ 4 \end{bmatrix}$ → saída desejada: $y = 0$

Algoritmo de Aprendizado - Perceptron

Considere um par de treinamento $\{\mathbf{x}, y_d\}$ (vetor de entrada e saída desejada)

O erro é definido como

$$e = y_d - y$$

Como $y_d, y \in \{0,1\}$, então o erro pode assumir 4 possíveis valores :

y (saída atual)	y_d (saída desejada)	e (erro)
0	0	0
0	1	1
1	0	-1
1	1	0

Algoritmo de Aprendizado - Perceptron

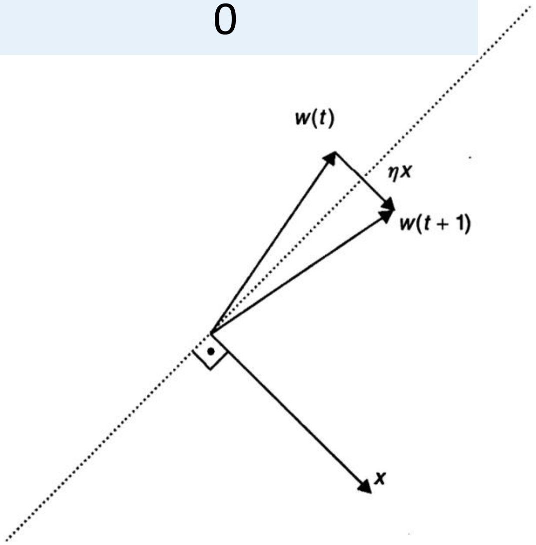
y (saída atual)	y_d (saída desejada)	e (erro)
0	0	0
0	1	1
1	0	-1
1	1	0

Suponha que $y = 0$ e $y_d = 1$

Nesse caso, temos $e = y_d - y = 1$ e

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} < 0$$

Ou seja, ângulo α entre vetores é maior do que 90 graus ($\cos(\alpha) < 0$)



Algoritmo de Aprendizado - Perceptron

Regra de adaptação

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

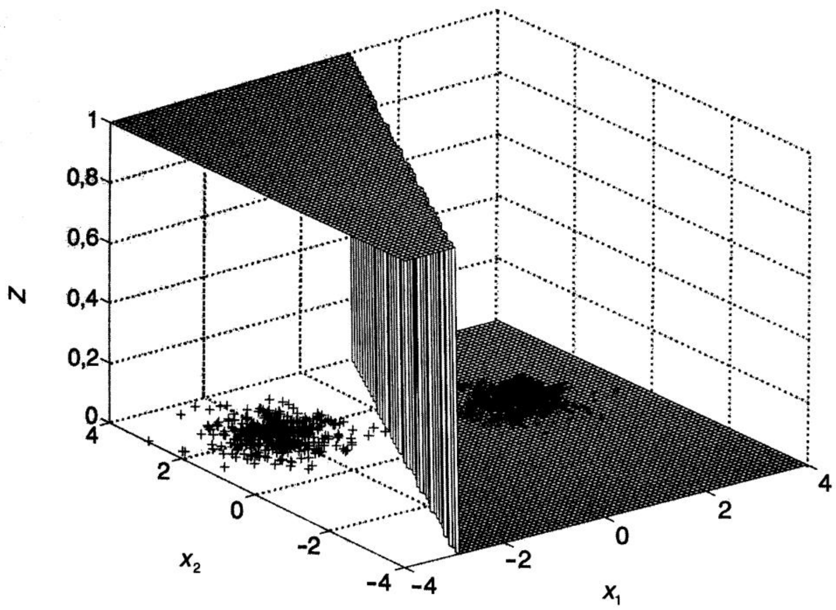
onde $e = y_d - y$

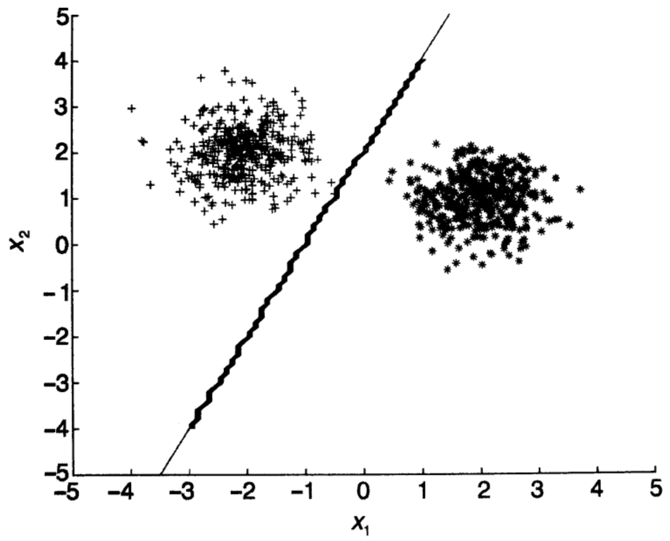
Nota:

- Algoritmo converge em tempo finito caso as classes sejam linearmente separáveis
- Em geral, pesos iniciais são definidos com valores amostrados com distribuição uniforme no intervalo [-a,a] (a próximo de zero)

Exemplo de Aplicação

Separar conjuntos de vetores $\mathbf{x} = [x_1, x_2]$





Exemplo de Aplicação

Problema: Classificação de Padrões

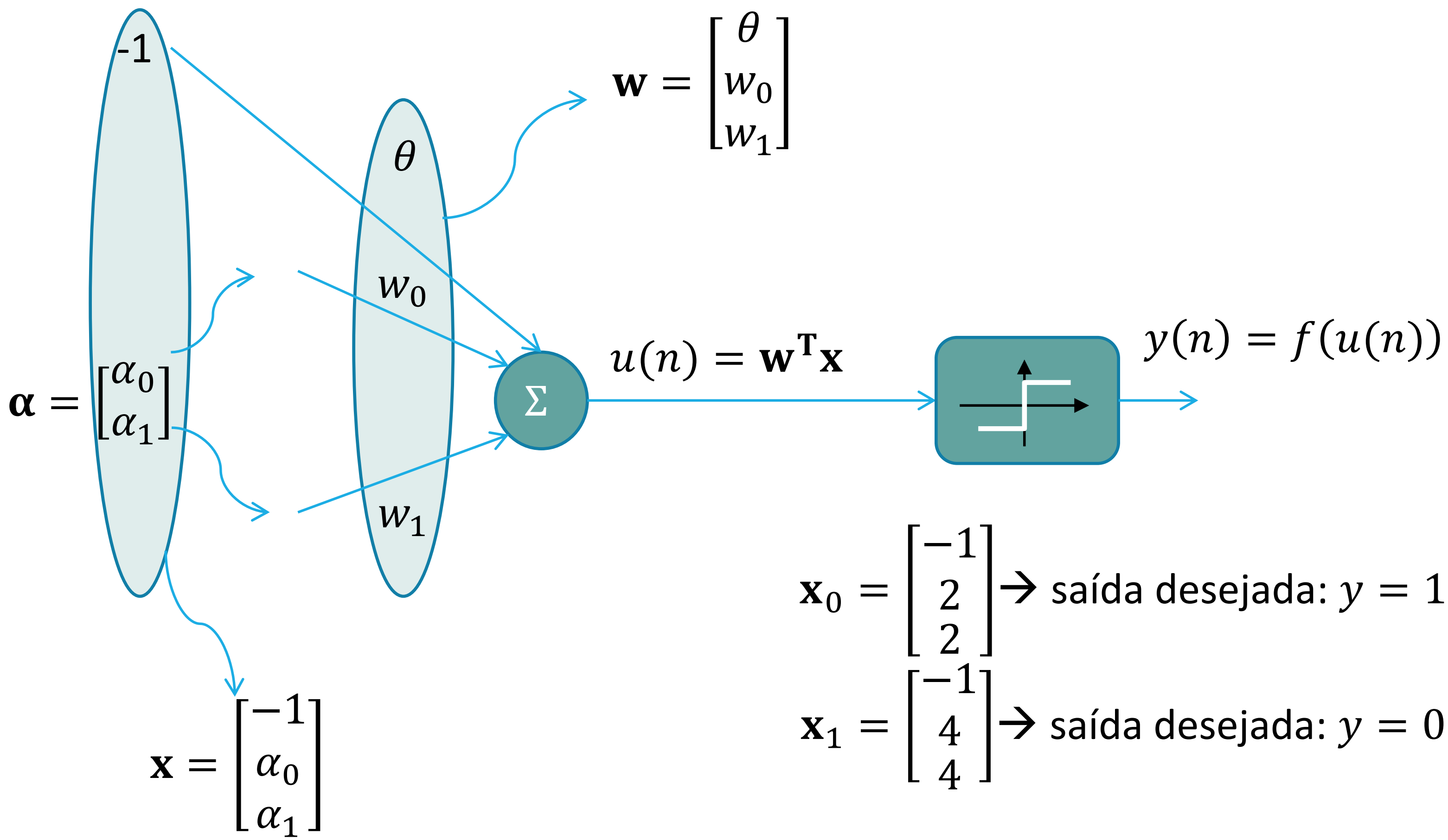
Conjunto de treinamento

- Padrão $\mathbf{x}_0$ corresponde à classe 1;
- Padrão $\mathbf{x}_1$ corresponde à classe 0;

Perceptron deve “aprender” a diferenciar os dois padrões → se $\mathbf{x}_0$ é apresentado em sua entrada, a saída do perceptron deve ser 1 etc.

Consideraremos o passo de adaptação $\eta = 0.1$

Exemplo de Aplicação



$\mathbf{x}_0 = \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}$ → saída desejada: $y = 1$

$\mathbf{x}_1 = \begin{bmatrix} -1 \\ 4 \\ 4 \end{bmatrix}$ → saída desejada: $y = 0$

Exemplo de Aplicação

Chute Inicial (inicialização aleatória)

$$\mathbf{w}(0) = \begin{bmatrix} -0.5441 \\ 0.5562 \\ -0.4074 \end{bmatrix}$$

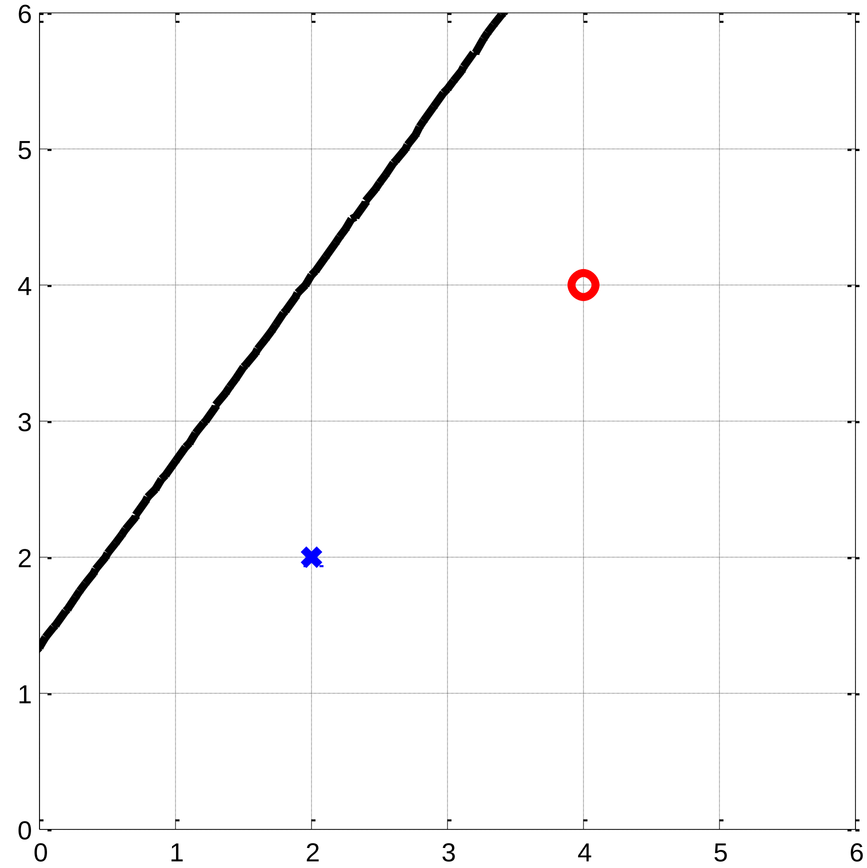
$$u(0) = \mathbf{w}(0)^{\mathbf{T}}\mathbf{x_0} = 0.8417$$

Portanto

$$y(0) = 1$$

e como a saída desejada é 1

$$e = y(0) - y_d = 0$$



Exemplo de Aplicação

1ª Iteração

$$\mathbf{w}(1) = \mathbf{w}(0) + \eta e \mathbf{x_0}$$
$$= \begin{bmatrix} -0.5441 \\ 0.5562 \\ -0.4074 \end{bmatrix} + 0.1 \times 0 \times \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$
$$= \begin{bmatrix} -0.5441 \\ 0.5562 \\ -0.4074 \end{bmatrix}$$

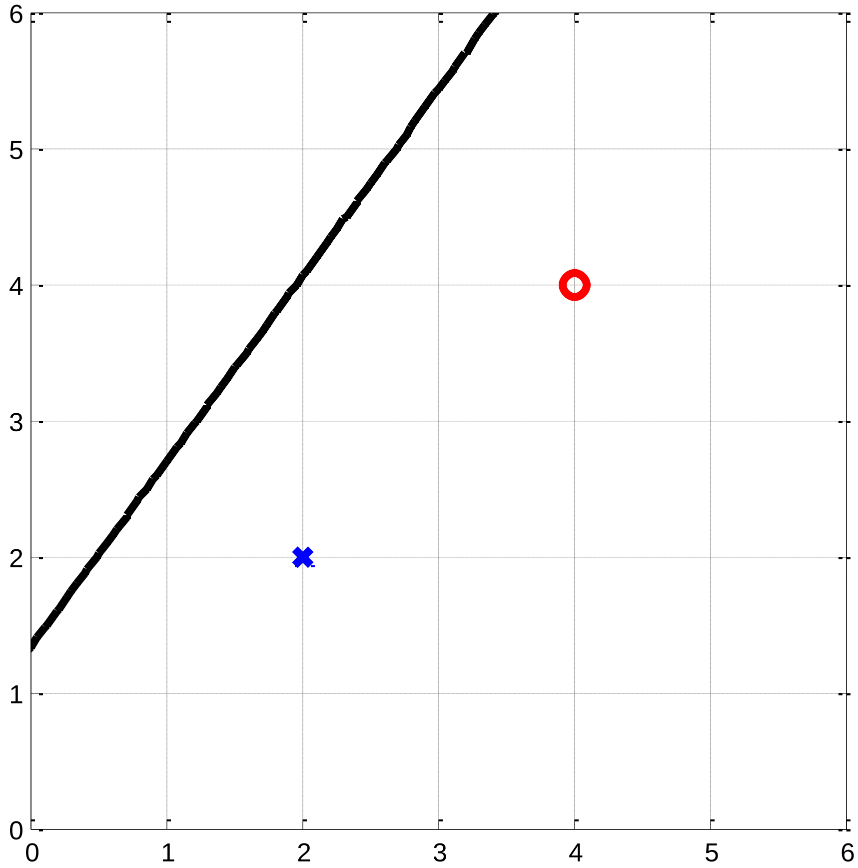
$$u(1) = \mathbf{w}(1)^{\mathbf{T}} \mathbf{x_1} = 1.1393$$

Portanto

$$y(0) = 1$$

e como a saída desejada é 0

$$e = y_d - y(0) = -1$$



Exemplo de Aplicação

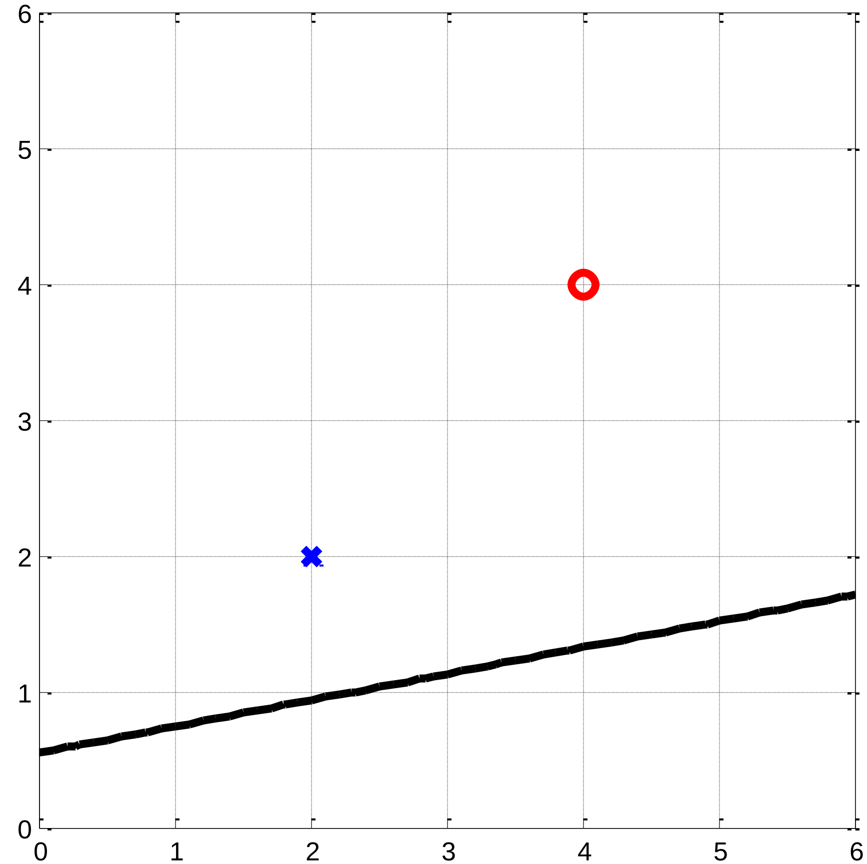
2ª Iteração

$$\mathbf{w}(2) = \mathbf{w}(1) + \eta e \mathbf{x}_1$$

$$= \begin{bmatrix} -0.5441 \\ 0.5562 \\ -0.4074 \end{bmatrix} + 0.1 \times -1 \times \begin{bmatrix} -1 \\ 4 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} -0.4441 \\ 0.1562 \\ -0.8074 \end{bmatrix}$$

Como foram utilizados todos os dados para treinamento, diz-se que completou-se um **época** (epoch) de treinamento.



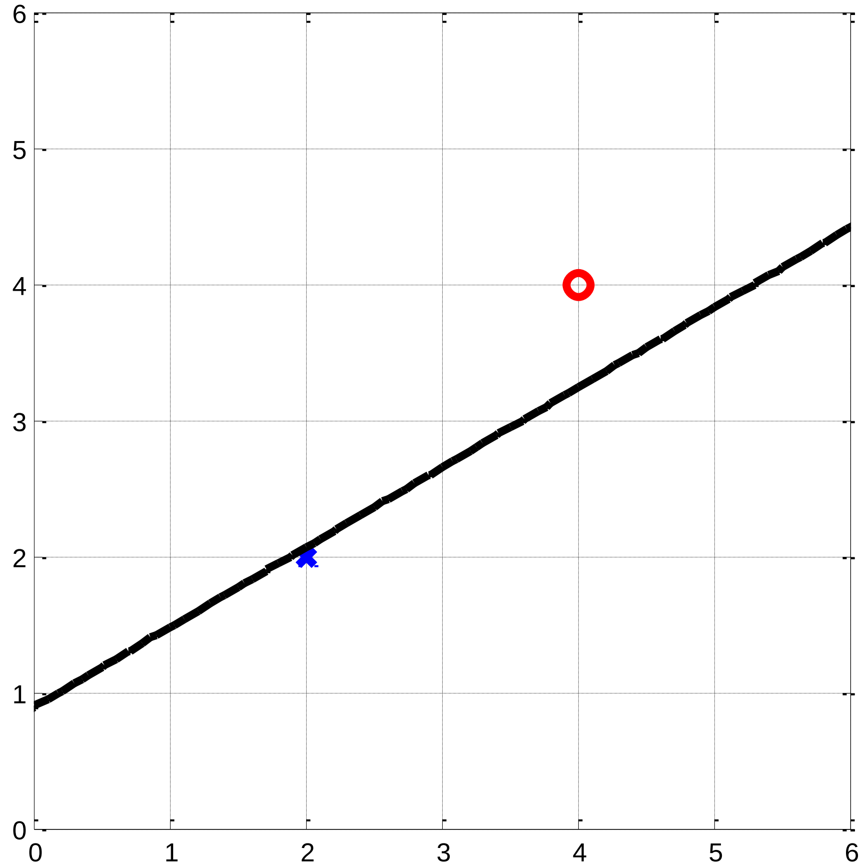
Exemplo de Aplicação

3ª Iteração (2ª Época)

$$\mathbf{w}(3) = \mathbf{w}(2) + \eta e \mathbf{x}_1$$

$$= \begin{bmatrix} -0.4441 \\ 0.1562 \\ -0.8074 \end{bmatrix} + 0.1 \times 1 \times \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} -0.5441 \\ 0.3562 \\ -0.6074 \end{bmatrix}$$



Como foram utilizados todos os dados para treinamento, diz-se que completou-se um **época** (epoch) de treinamento.

Exemplo de Aplicação

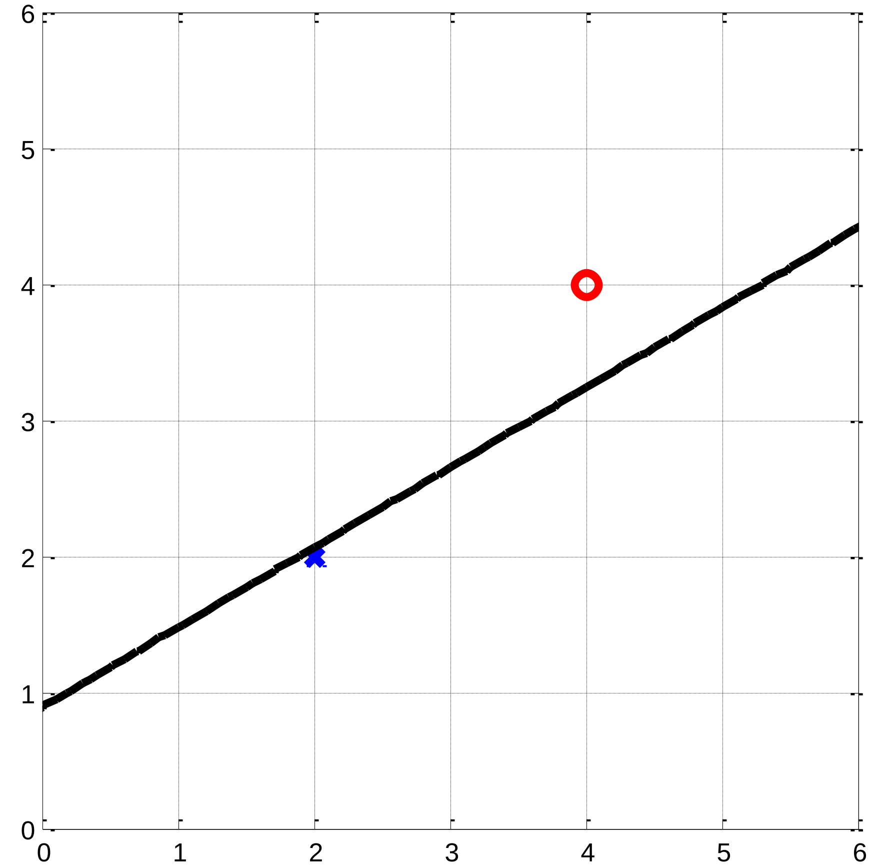
4ª Iteração (2ª Época)

$$\mathbf{w}(4) = \mathbf{w}(3) + \eta e \mathbf{x}_1$$

$$= \begin{bmatrix} -0.5441 \\ 0.3562 \\ -0.6074 \end{bmatrix} + 0.1 \times 0 \times \begin{bmatrix} -1 \\ 4 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} -0.5441 \\ 0.3562 \\ -0.6074 \end{bmatrix}$$

Não há mudança nos pesos pois a solução já separa as duas classes.



Algoritmo de Treinamento do Perceptron

Escolher um valor arbitrário de η

Inicializar o vetor de pesos $\mathbf{w}$

Aplique a regra de atualização

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

onde $e = y_d - y$, para todos os dados de treinamento (pares de padrão de entrada $\mathbf{x}_i$ e saída desejada y_d^i)

Repita o passo anterior até que $e = 0$ para todos os padrões (ou até que um certo critério de para seja atendido)

Sistemas Inteligentes

REDE NEURAL ADAPTIVE LINEAR (ADALINE)

Adaline – Adaptive Linear

Modelo utiliza uma função de ativação linear

$$f(u) = u$$

onde $u = \sum_{i=0}^{n} w_i x_i$. Portanto

$$y = \mathbf{w}^T \mathbf{x}$$

Assim, a Adaline provê um mapeamento linear (afim) genérico.

Aprendizado – Função Custo

O aprendizado/treinamento é baseado em uma função custo, em geral utilizando o erro quadrático

$$J = \frac{1}{2}\sum_{i=1}^{n}(e_i)^2 = \frac{1}{2}\sum_{i=1}^{n}\left(y_d^i - y^i\right)^2$$

A função custo apresenta apenas um mínimo global (equação de uma parábola)

Regra Delta - Aprendizado

Para obter os parâmetros ótimos da rede podemos utilizar um procedimento iterativo, a exemplo da regra de adaptação do perceptron

Para isso, é necessário obter o gradiente da função custo (aproximando, do erro instantâneo e_i) em relação aos parâmetros

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial y} \frac{\partial y}{\partial w_i}$$

Regra Delta - Aprendizado

$$\frac{\partial e^2}{\partial w_i} = 2e\frac{\partial e}{\partial w_i} = 2e\frac{\partial e}{\partial y}\frac{\partial y}{\partial w_i} =$$

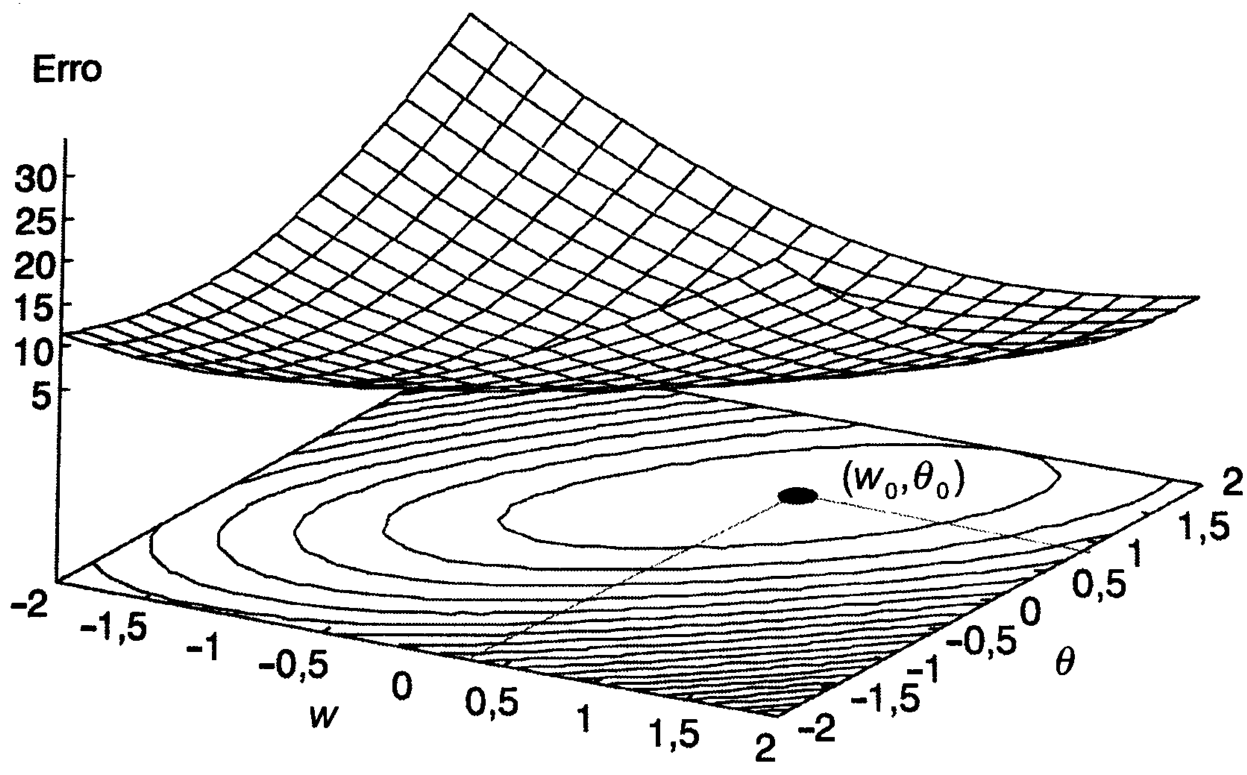
$$= 2e\frac{\partial (y_d - y)}{\partial y}\frac{\partial (w_0 x_0 + w_1 x_1 + \cdots + w_n x_n)}{\partial w_i}$$

$$= -2x_i e$$

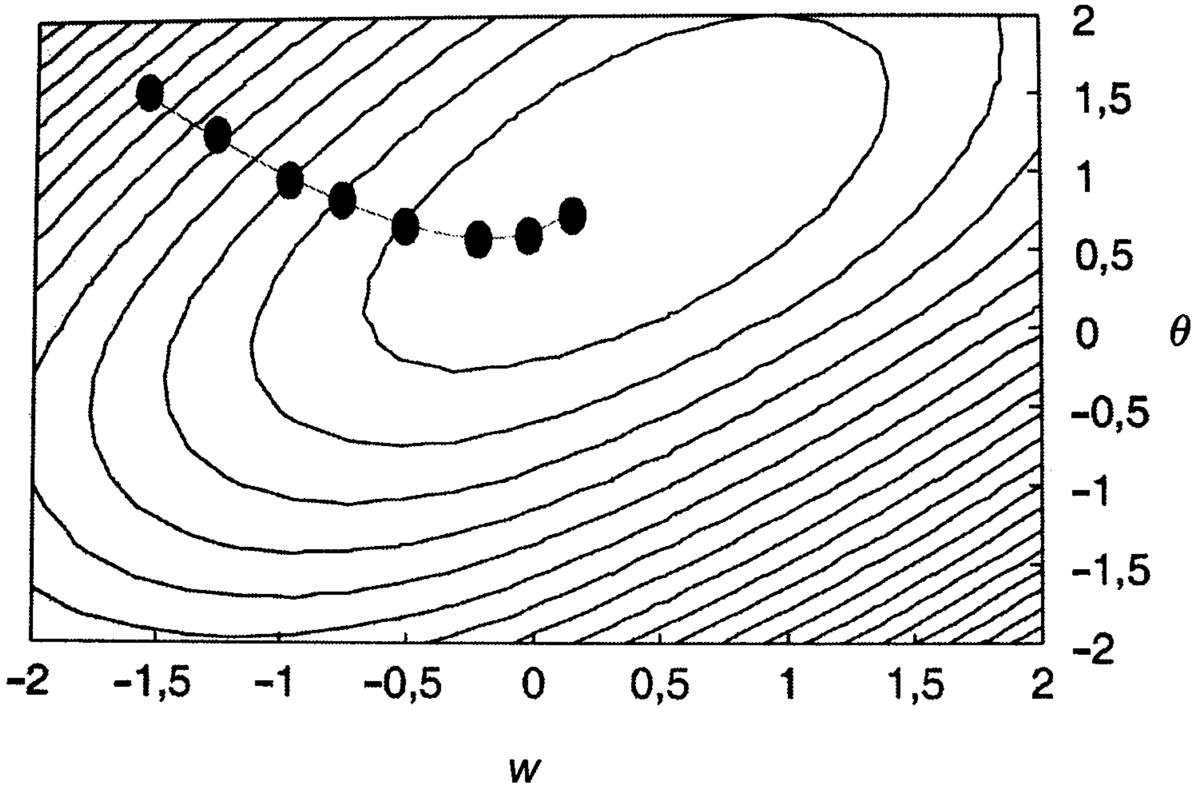
Como o gradiente aponta na direção de maior crescimento, devemos adaptar os coeficientes considerando

$$\Delta w_i = \eta e x_i \;\rightarrow\; \mathbf{w}(n+1) = \mathbf{w}(n) + \eta e\,\mathbf{x}\,(n)$$

Superfície de erro – Erro Quadrático



Evolução dos parâmetros



Multi-Layer Perceptron (MLP)

O perceptron de múltiplas camadas (*multi-layer perceptron*) → generalização do perceptron simples.

Otmização dos parâmetros da rede → algoritmo de treinamento mais sofisticado capaz de definir de forma automática os pesos.

O algoritmo de treinamento → generalização da regra delta da Adaline.

Introdução

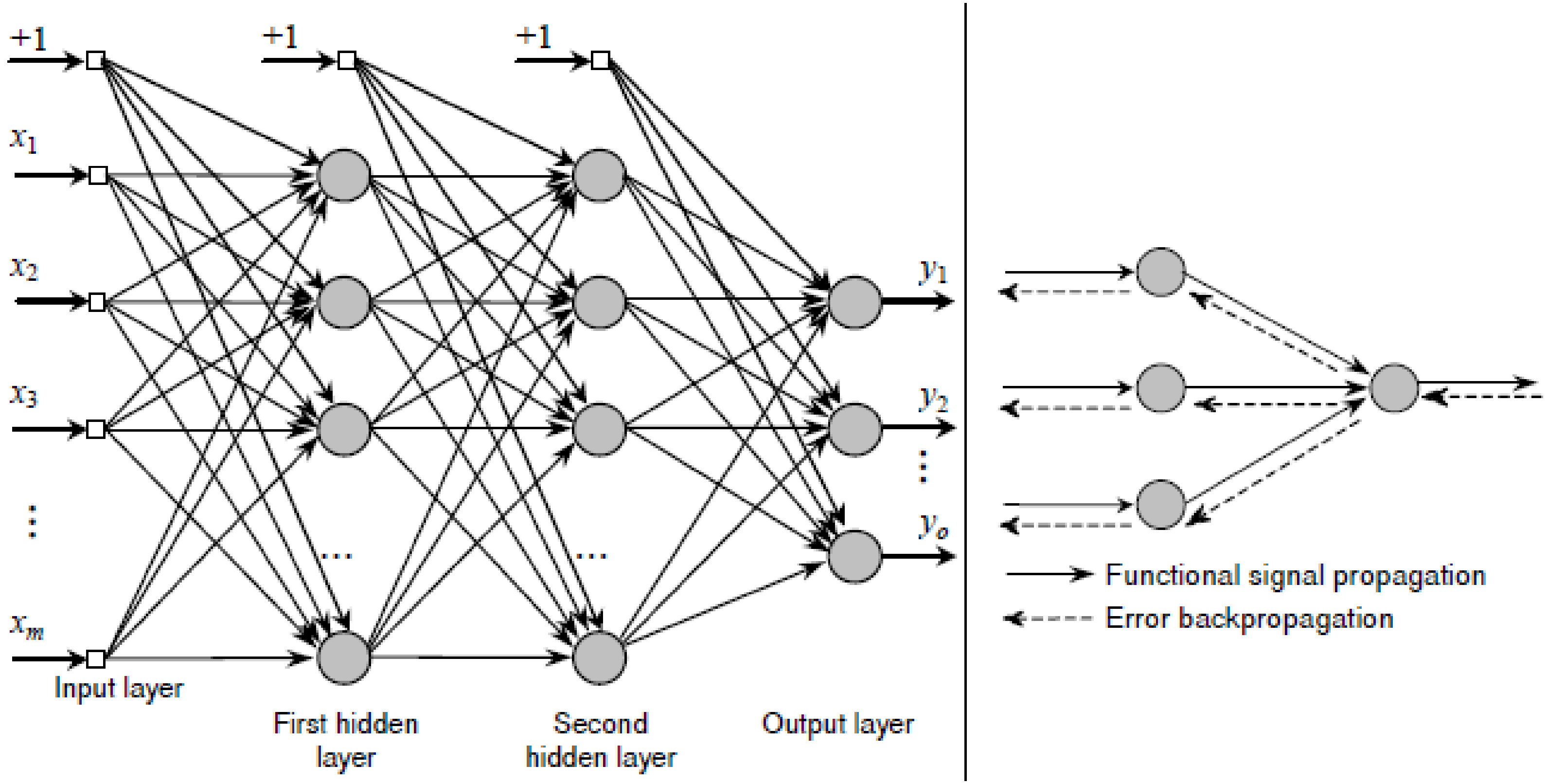
Algoritmo de treinamento: *backpropagation*.

- *Propagação positiva do sinal funcional*: durante este processo todos os pesos da rede são mantidos fixos; e
- *Retropropagação do erro*: durante este processo os pesos da rede são ajustados tendo por base uma medida de erro por base.

Uma rede MLP típica possui três características principais:

- Os neurônios das camadas intermediárias (e, eventualmente, os da camada de saída) possuem uma função de ativação não-linear do tipo sigmoidal (e.g. função logística ou tangente hiperbólica).
- A rede possui uma ou mais camadas intermediárias.
- A rede possui um alto grau de conectividade.

Estrutura da MLP



Algoritmo Backpropagation

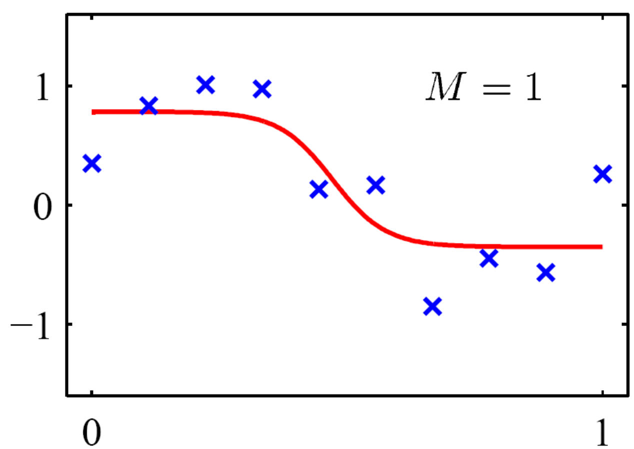
Pesos da RNA são ajustados de maneira a minimizar o erro entre a saída atual da rede e a resposta desejada

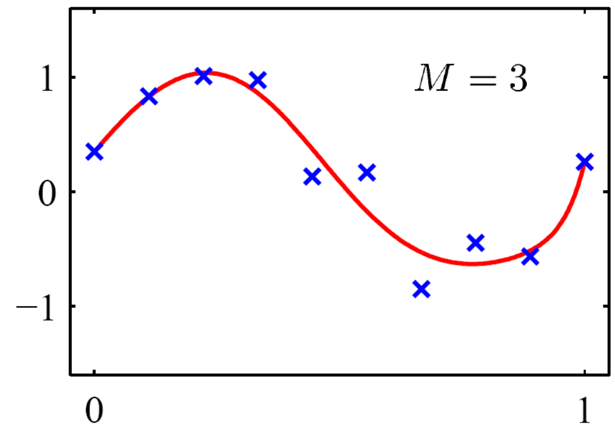
Para isso, serão utilizadas informações sobre o gradiente do erro em relação aos pesos e limiares da rede → gradiente informa a direção de máxima variação do erro.

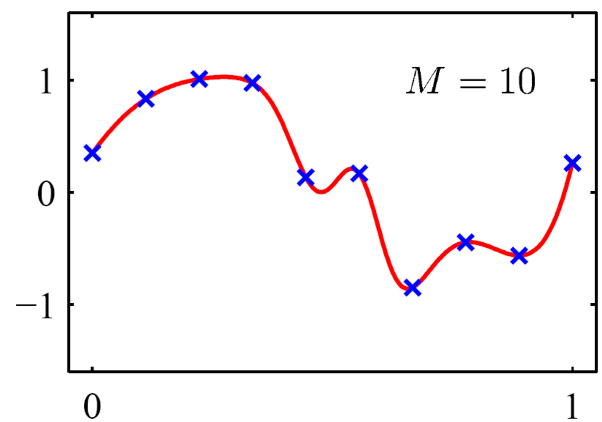
Entretanto, o erro é calculado diretamente apenas para a camada de saída. Como determinar a influência do erro nas camadas intermediárias da rede?

Complexidade da função vs Número de neurônios

M = número de neurônios







O efeito do mínimo local

Convergência do algoritmo depende da inicialização

Erro de validação