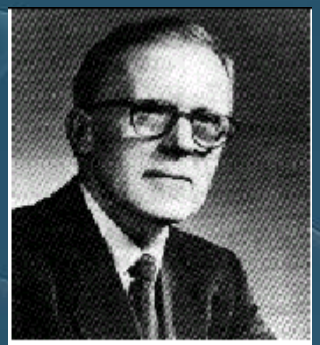
Redes Neurais Artificiais : Visão Geral e o Perceptron

Prof. Dr. Ernane José Xavier Costa – LAFAC- FZEA- USP

Redes Neurais

Histórico


Donald Hebb

Donald Hebb (1949) – Progressos na neuroanatomia e neurofisiologia. Propõe uma regra de aprendizado, marco inicial no treinamento de RNA

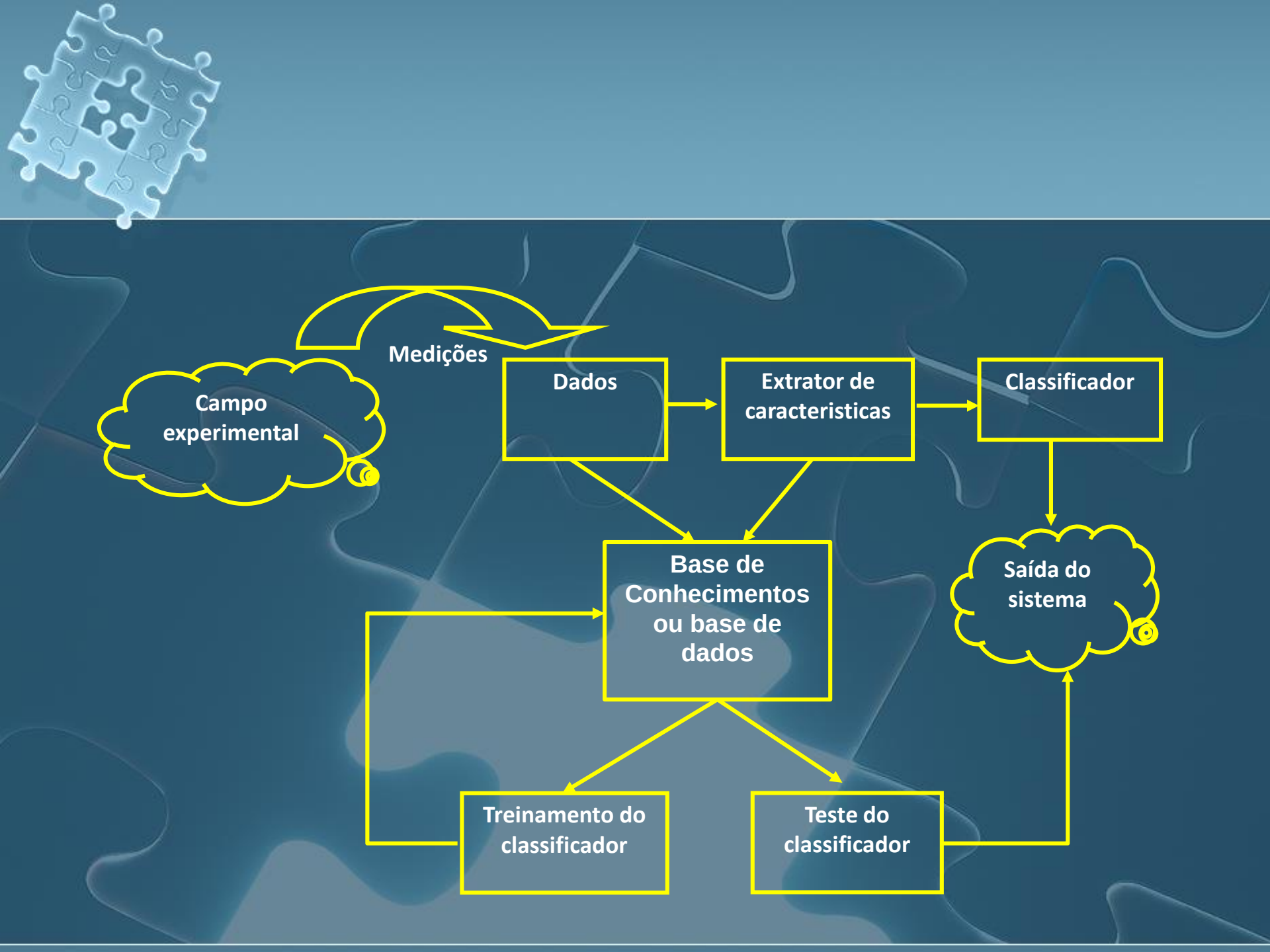
Trabalho pioneiro:

MacCulloch-Pitts: propuseram um modelo para o neurônio dizendo que qualquer função lógica finita pode ser implementada pela associação de neurônios artificiais

Representação do conhecimento

Conhecimento: refere-se à informação armazenada ou à modelos utilizados por uma pessoa ou máquina para interpretar, prever e responder apropriadamente ao mundo exterior.

Uma boa solução depende de uma bo representação do conhecimento


Medições
Campo experimental
Dados
Extrator de caracteristicas
Classificador
Base de Conhecimentos ou base de dados
Saída do sistema
Treinamento do classificador
Teste do classificador

Representação do conhecimento

Principais caracteristicas da representação do conhecimento:

1) Que informação é realmente tornada explicita?

2) Como a informação é codificada fisicamente para uso subsequente?

Representação do conhecimento

Existem dois tipos de conhecimentos:

1) Conhecimento a priori baseado nos fatos e estados do mundo

2) Medidas e observaçoes do mundo realizadas por sensores .

Destas observações pode se retirar exemplos para treinar uma RNA

Representação do conhecimento

Estes exemplos podem ser Rotulados ou Não Rotulados

Rotulados

- Cada exemplo de entrada possui uma saida desejada,
- Neste caso deve-se supervisionar o treinamento para verificar a similaridades entre a saida obtida e a desejada.

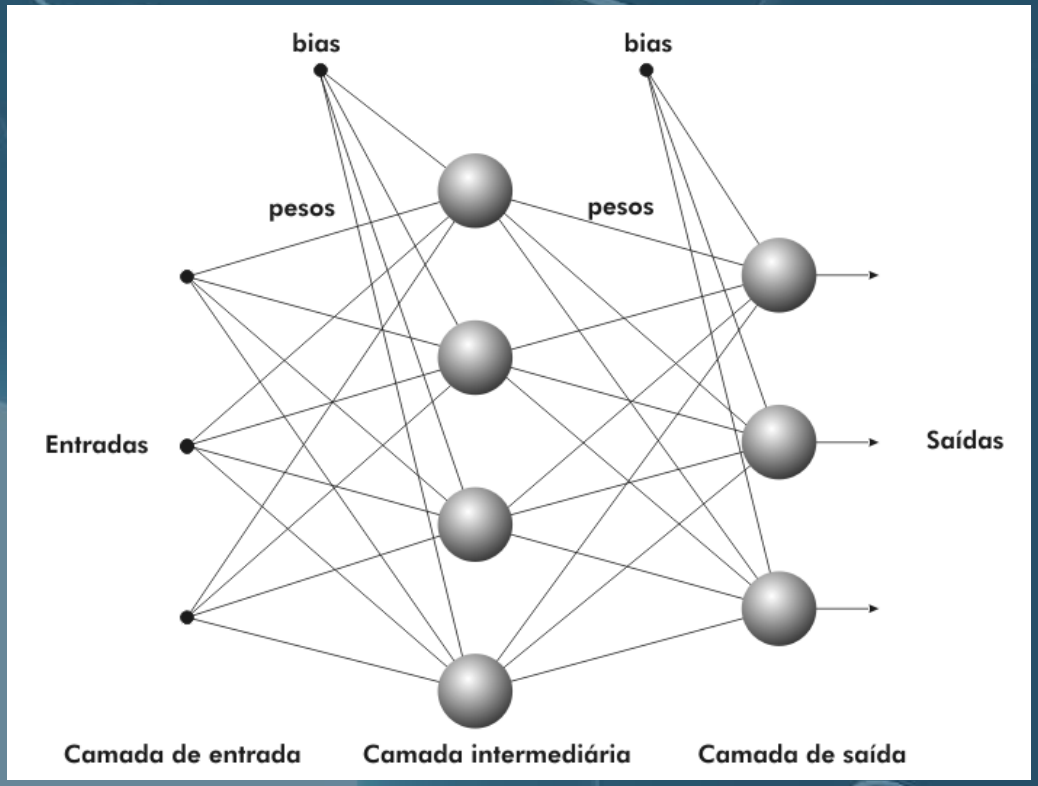
Não rotulados

- Os exemplos possuem caracteristicas comuns que podem ser encontradas pela rede sem supervisão

Redes Neurais

A Rede Artificial

- Disposição em camadas
- Comunicação entre neurônios entre camadas adjacentes
- São aproximadores universais de funções multivariáveis



Redes Neurais

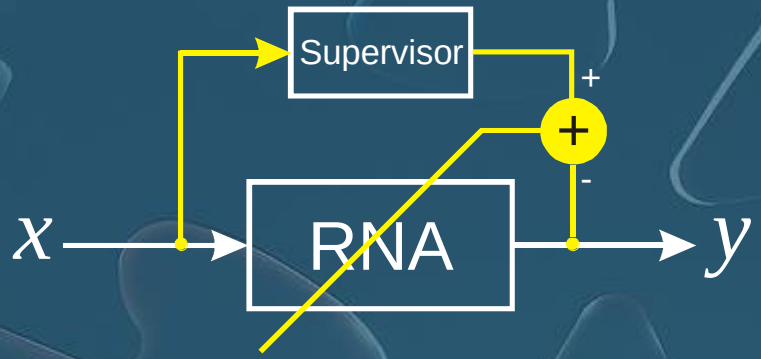
Aprendizado

- Aprendizado por exemplos.

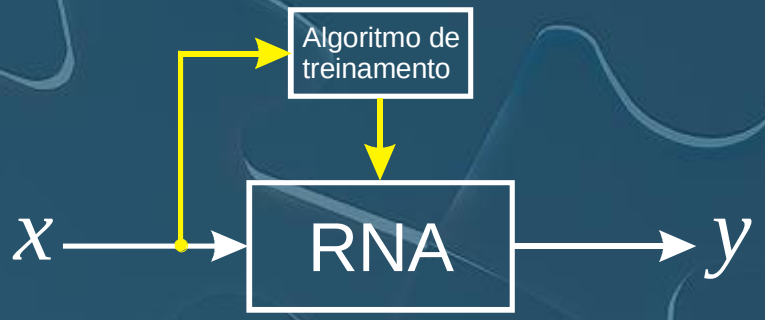
- Para um conjunto de dados, o algoritmo de aprendizado promove adaptação dos parâmetros da rede (pesos e bias) de maneira que, em um número finito de iterações , haja convergência para uma solução.

Redes Neurais

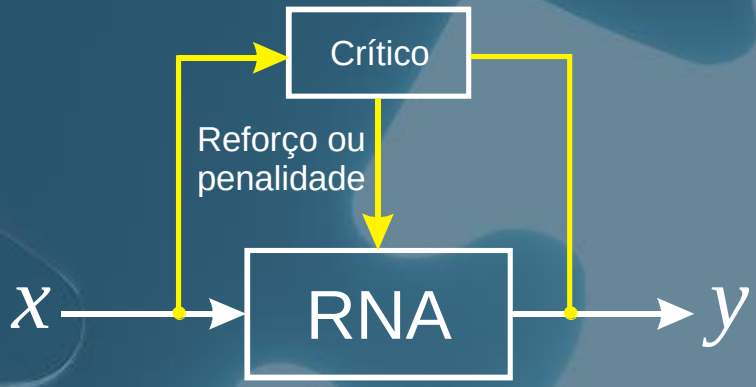
Aprendizado



Aprendizado supervisionado



Aprendizado não-supervisionado



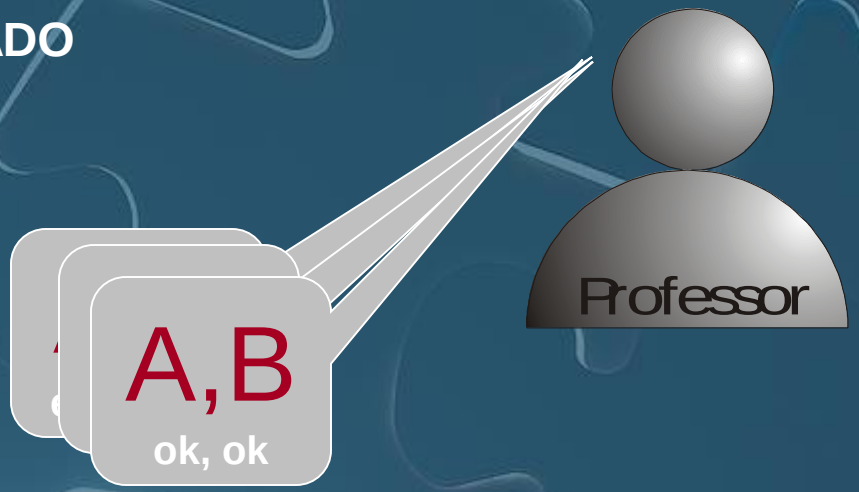
Aprendizado por reforço

Redes Neurais

Aprendizado

TREINAMENTO SUPERVISIONADO

- conjunto de treinamento



PADRÕES

A
B

REDE NEURAL

1ª ITERAÇÃO	2ª ITERAÇÃO	3ª ITERAÇÃO
~~B~~	A	A
~~B~~	~~B~~	B

Redes Neurais

Aplicações

As RNAs são capazes de resolver:

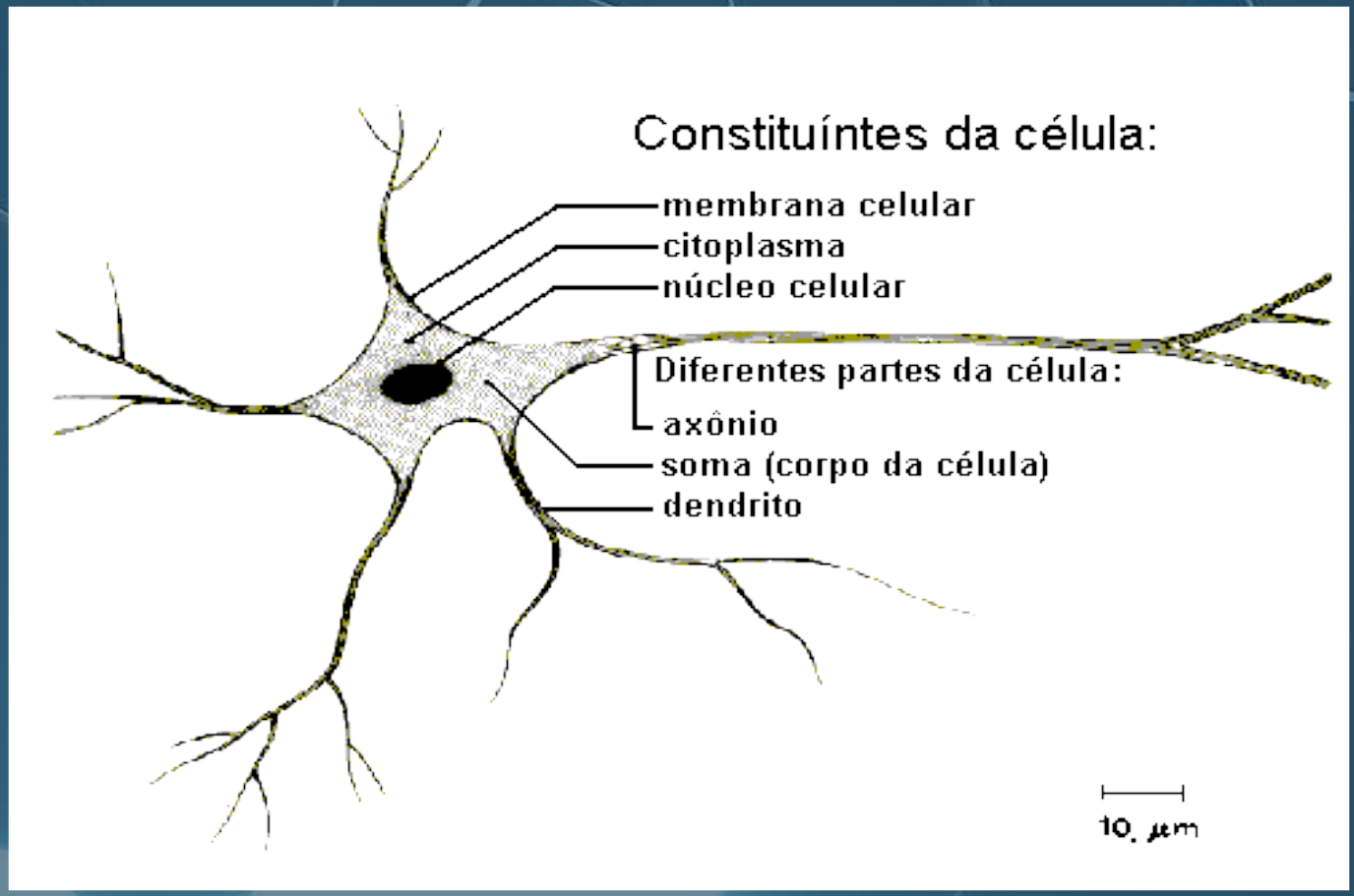
- Problemas de aproximação
- Predição
- Classificação
- Categorização
- Otimização

Redes Neurais

O Neurônio

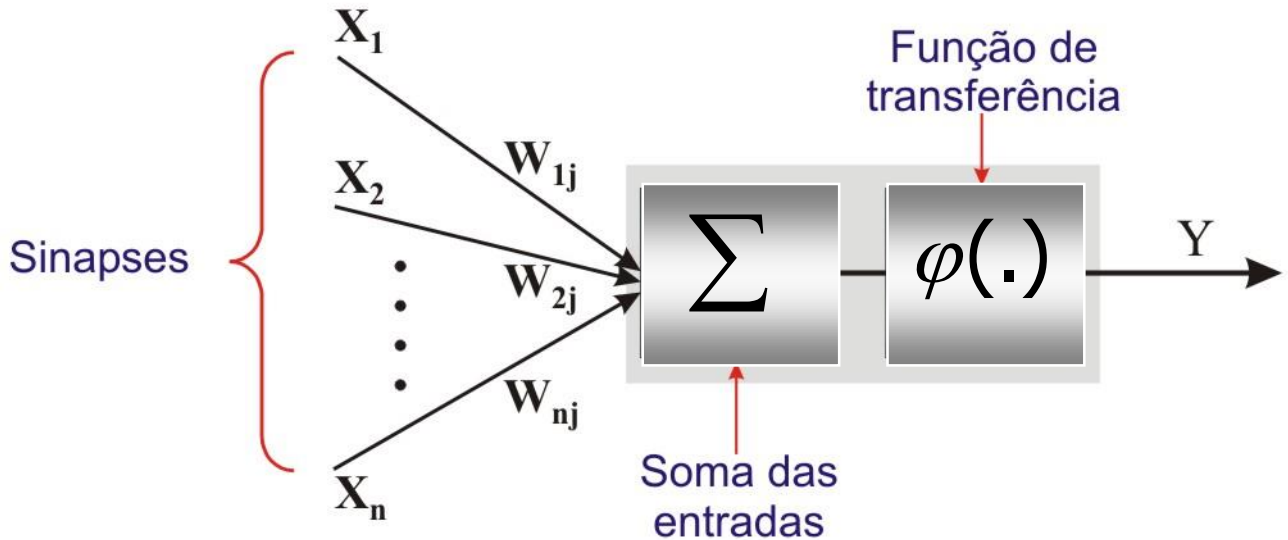
PARTES

- dendritos
- corpo celular
- axônio
- sinapse



Redes Neurais

O Neurônio artificial

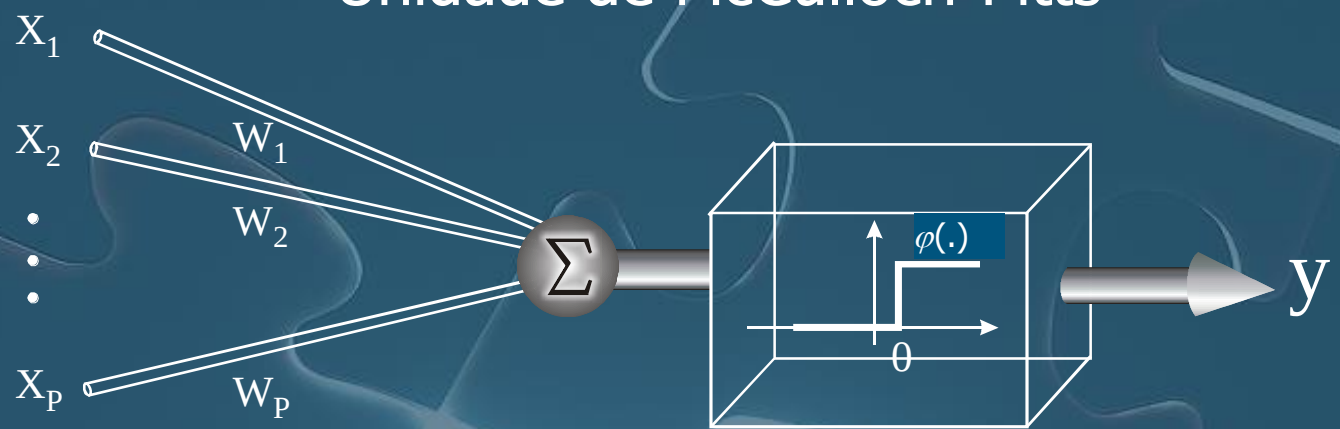


$$Y = \varphi\left(\sum_{i=1}^{n} W_{ij} X_j\right)$$

Redes Neurais

Histórico

Unidade de McCulloch-Pitts

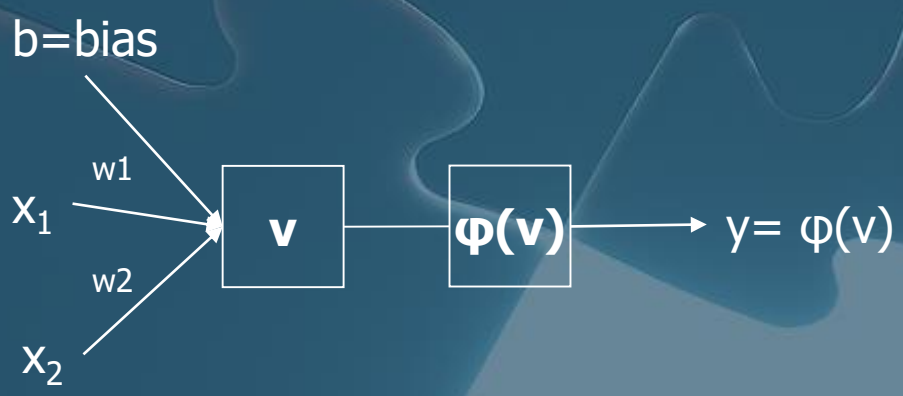


- Sinais são apresentados à entrada.
- Cada sinal é multiplicado por um número (peso).
- É feita a soma ponderada que produz um nível de atividade.
- Se este nível de atividade exceder um limite, a unidade produz uma determinada resposta de saída.

Redes Neurais

O Neurônio artificial

Exemplo:



Para: $v = \gamma + b$

$$\varphi(v) = \begin{cases} 0 & v < 0 \\ 1 & v \geq 0 \end{cases}$$

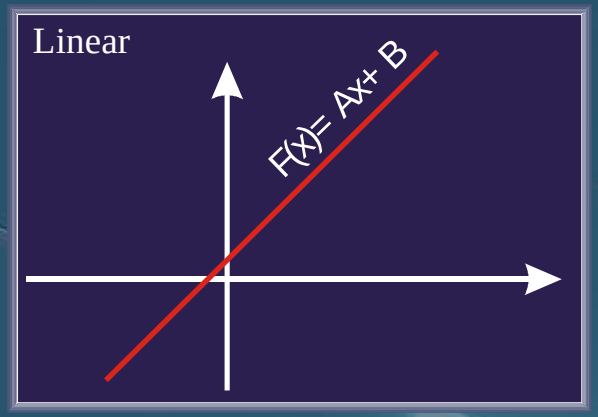
$$\varphi(v) = \begin{cases} 0 & \gamma + b < 0 \\ 1 & \gamma + b \geq 0 \end{cases}$$

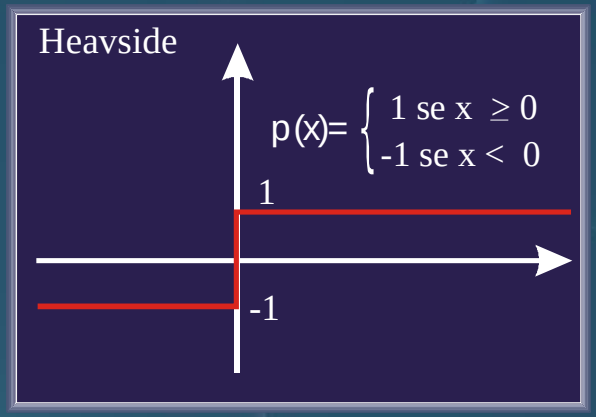
$$\varphi(v) = \begin{cases} 0 & \gamma < -b \\ 1 & \gamma \geq -b \end{cases}$$

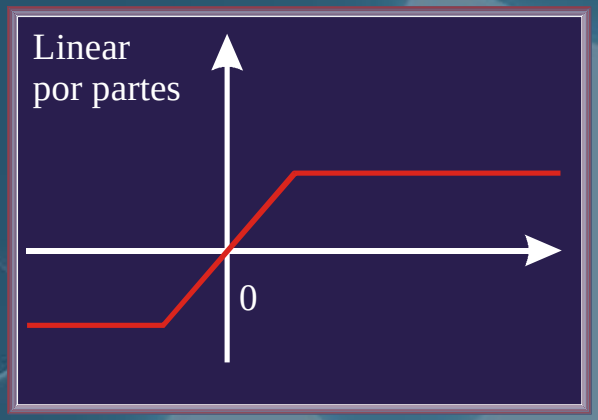
Redes Neurais

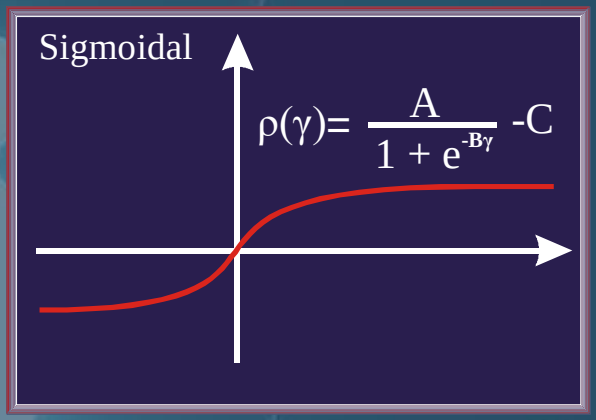
O Neurônio artificial

Funções de ativação









Redes Neurais

O Neurônio artificial

Na ativação tipo sigmoidal onde a função é:

$$\varphi(\gamma) = \frac{A}{1 + e^{-B\gamma}} - C$$

As saídas são limitadas a: $\gamma = \sum_{i=1}^{n} \omega_{ij} y_j$

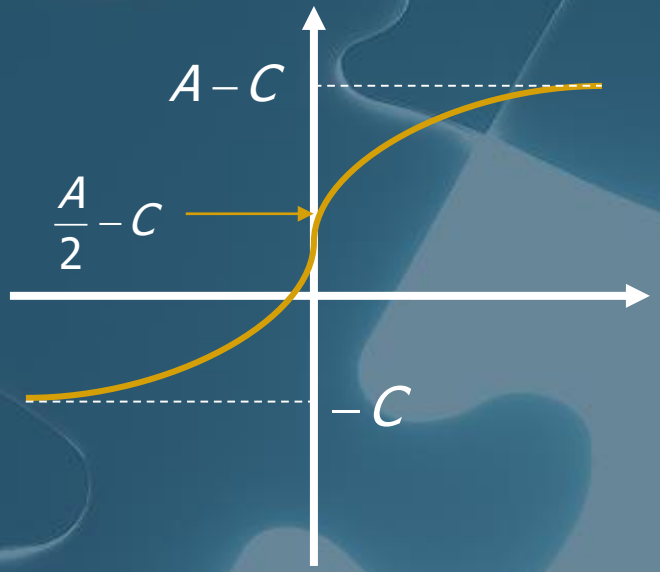
Como é a saída do neurônio com esta função?

$$\begin{cases} \varphi(\gamma = 0) = \frac{A}{2} - C \\ \varphi(\gamma \to \infty) = A - C \leftarrow \text{Limite superior} \\ \varphi(\gamma \to -\infty) = -C \leftarrow \text{Limite inferior} \end{cases}$$

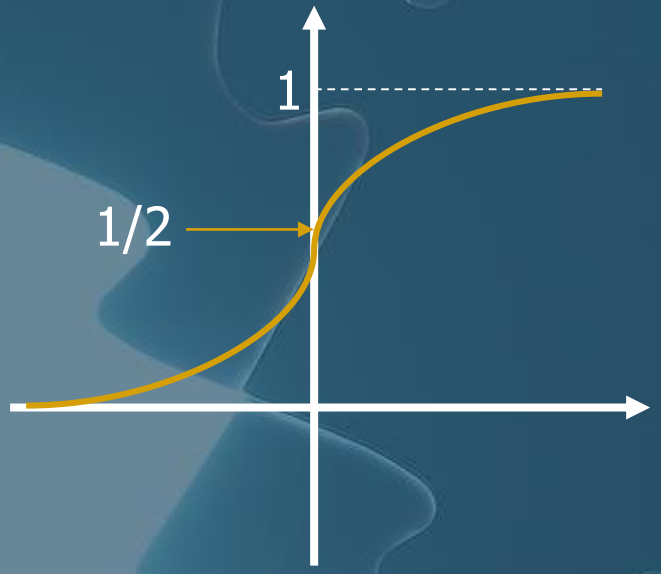
Redes Neurais

O Neurônio artificial

A e C controlam o limiar de ativação
B controla a forma da ativação



Se A=B=1 e C=0, temos:



Redes Neurais

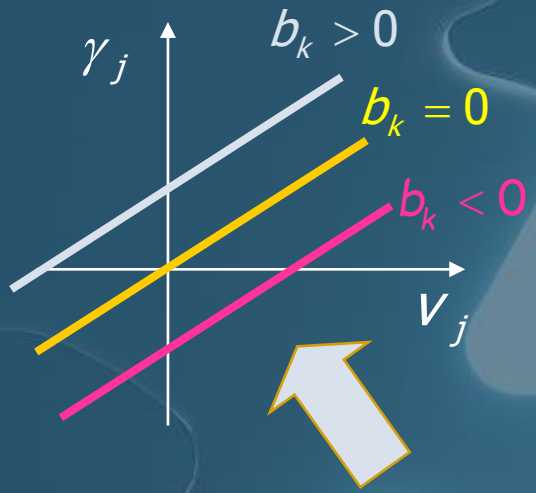
O Neurônio artificial

O que é bias?

Bias é uma transformação afim da função

$$v_j = \sum_{i=0}^{n} \omega_{ji} x_i + b_j \quad \Rightarrow \quad v_j = \gamma_j + b_j$$

$$\gamma_j$$



$$\varphi(v) \rightarrow \varphi(\gamma_j + b_j) \rightarrow \varphi(\sum_{i=1}^{n} \omega_{ij} x_i + b_j)$$

b_k desloca a reta!!!!!!

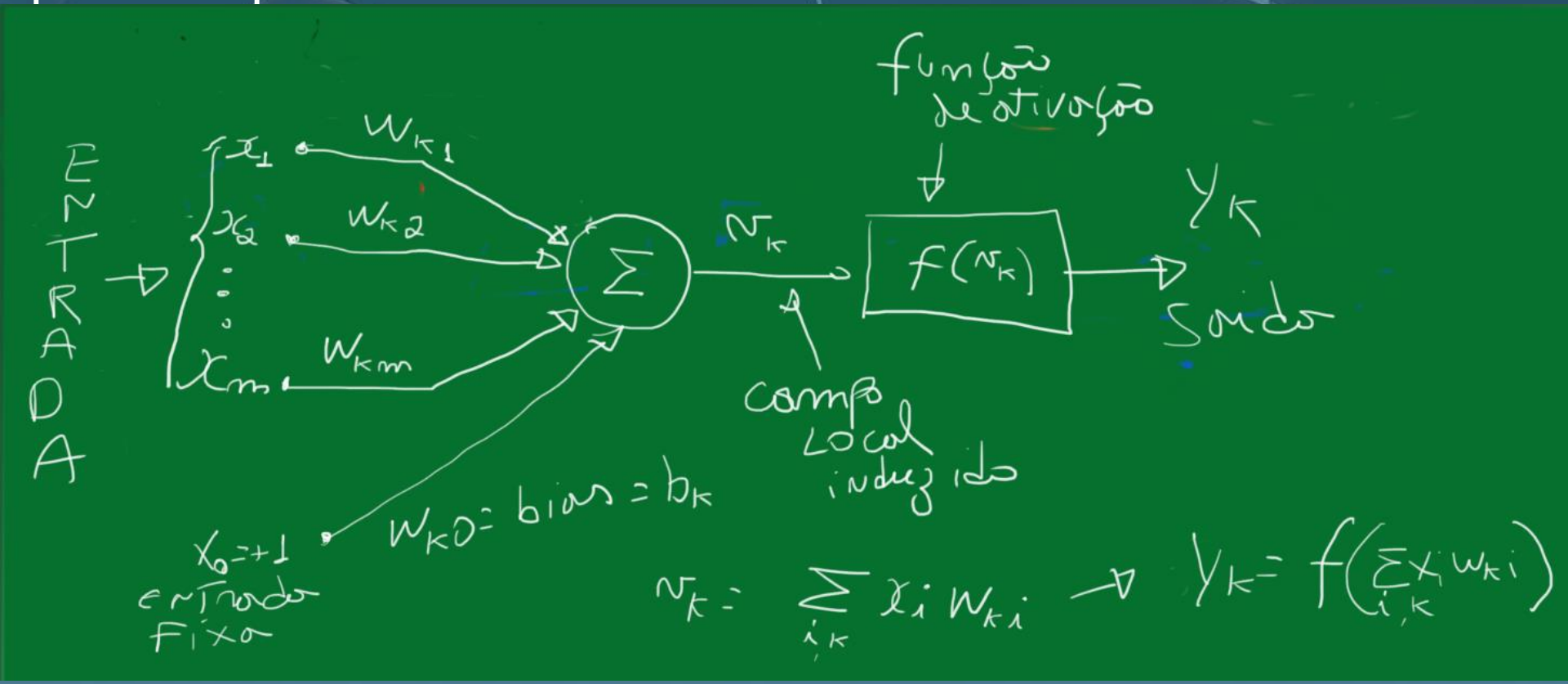
Perceptron

- Basicamente o perceptron consiste de uma única camada de neurônios com com pesos sinápticos e bias ajustáveis.

- Se os padrões de entrada forem linearmente separáveis, o algoritmo de treinamento possui convergência garantida, i.é, tem capacidade para encontrar um conjunto de pesos que classifica corretamente os dados.

Perceptron

- Os neurônios do perceptron são similares ao de McCulloch-Pitts, por terem a função de ativação do tipo degrau, mas possuem pesos associados e bias.



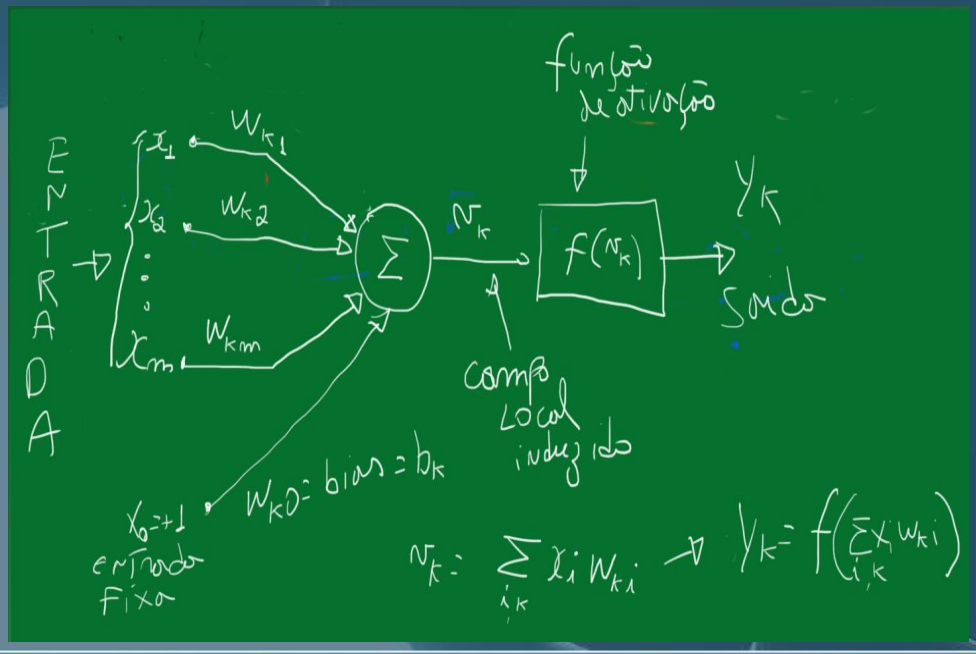
Algoritmo de Treinamento do Perceptron Classico

- para cada padrão de treinamento xi, a saída da rede yi é calculada.
- determina-se o erro ei entre a saída desejada para esse padrão di e a saída da rede yi, (ei = di – yi).

O vetor de pesos e o bias são atualizados de acordo com as regras:

W_i (novo) = W_i (velho) + $\alpha T x_i$)
b(novo) = b(velho) + αT

T = saída desejada
A = taxa de aprendizagem



Algoritmo de treinamento Classico

Para classificação padrões de entrada como pertencentes ou não a uma dada classe, seja o conjunto de treinamento formado por N amostras $\{\mathbf{x}_1,d_1\}$, $\{\mathbf{x}_2,d_2\}$, ..., $\{\mathbf{x}_N,d_N\}$,

onde $\mathbf{x}_j$ é o vetor de entradas e d_j a saída desejada, que em notação vetorial tem-se $\{\mathbf{X},\mathbf{d}\}$, onde:

$$\mathbf{X} \in \Re^{mxN} \qquad \mathbf{d} \in \Re^{1xN}$$

Perceptron Classico

Passo 0 – Inicializa pesos e bias com zero

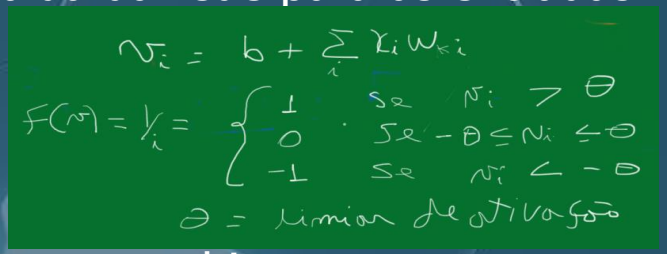
Inicialize a taxa de aprendizagem $< 0 \alpha \leq 1$

Passo 1 – Enquanto condição de parada for falsa execute os passos 2 à 6

Passo 2 - Para cada par de treinamento s:T execute os passos 3 à 5

passo 3 – Faça xi = Si ou seja entre com os dados na rede

passo 4 - calcule a saída da rede para as entradas xi como:



passo 5 – Atualize os pesos e bias se :

Y for diferente de T

senão:

Não mude os valores dos pesos.

Passo 6 – teste a condição de parada: Se nenhum peso mudar dentro do passo 2 pare:

senão continue do passo 1

Perceptron Modificado

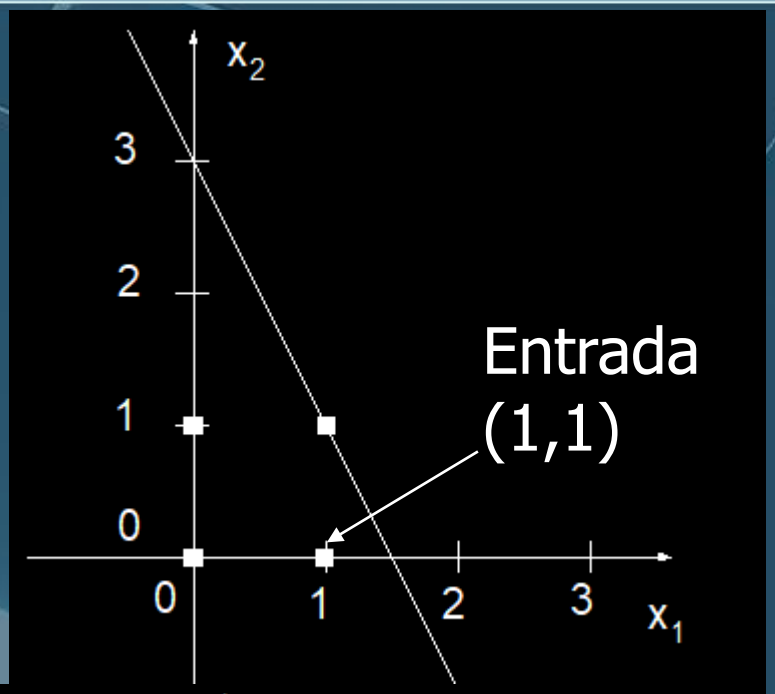
Function [**w**] = perceptron (max_it, E, α, **X**,**d**)
inicializar **w** // para simplicidade, com zeros
inicializar b // para simplicidade, com zero
t $\leftarrow$ 1 // controle da iteração ou época
acumula_erro = 0
while t < max_it & E > 0 do
for i from 1 to N do // para cada padrão de entrada
$y_i \leftarrow f(\mathbf{w}\,\mathbf{x}_i + b)$ // determinar a saída
$e_i \leftarrow d_i - y_i$ // determinar o erro
$\mathbf{w} \leftarrow \mathbf{w} + \alpha\, e_i\, \mathbf{x}_i$ // atualizar o vetor peso
$b \leftarrow b + \alpha\, e_i$ // atualizar o bias
end for
acumula_erro $\leftarrow$ acumula_erro + e_i //quantidade de erros
t $\leftarrow$ t + 1
end while
end procedure

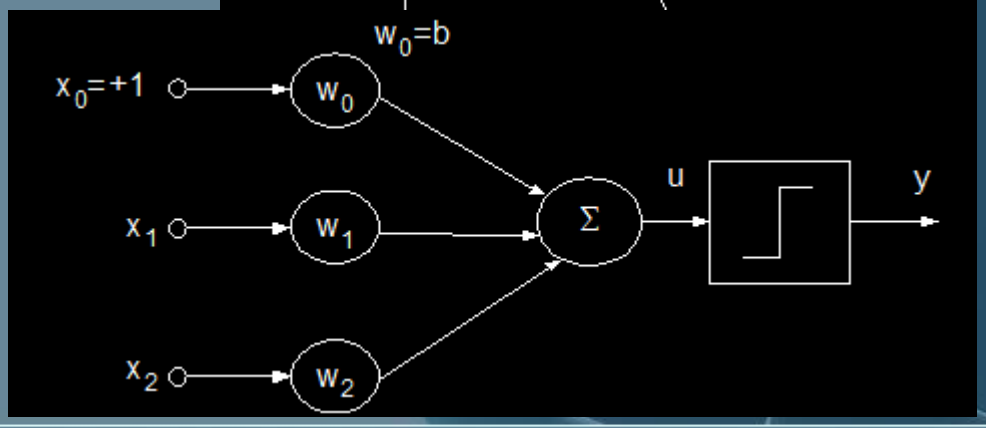
Exemplo: função AND

Temos como vetores de entrada e saída desejada:

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \qquad \mathbf{d} = [0 \quad 0 \quad 0 \quad 1]$$

Iniciando os pesos e o limiar em zero **w** = [0 0], $b = 0$ e $\alpha = 1$, tem-se $w_1 = 2$, $w_2 = 1$, $b = -3$ e a equação da reta $2x_1 + 1x_2 - 3 = 0$.





Algoritmo de treinamento para perceptron de múltiplos neurônios

Nesse caso, para cada vetor de entradas tem-se um vetor de saída

$\mathbf{y}_i = f(\mathbf{W}\mathbf{x}_i + \mathbf{b});\ \mathbf{W} \in \Re^{oxm};\ \mathbf{x}_i \in \Re^{mx1}, i = 1,..., N;\ \mathbf{y}_i \in \Re^{ox1}, \mathbf{b} \in \Re^{ox1}\ e\ \mathbf{D} \in \Re^{oxN}$

Procedure [**W**] = perceptron (max_it, α, **X**,**D**)
  inicializar **W** // para simplicidade, com zeros
  inicializar **b** // para simplicidade, com zero
  t $\leftarrow$ 1
  while t < max_it do
    for i from 1 to N do // para cada padrão de entrada
      $\mathbf{y}_i \leftarrow f(\mathbf{W}\,\mathbf{x}_i + \mathbf{b})$ // determinar a saída
      $\mathbf{e}_i \leftarrow \mathbf{d}_i - \mathbf{y}_i$ // determinar o erro
      $\mathbf{W} \leftarrow \mathbf{W} + \alpha\,\mathbf{e}_i\,\mathbf{x}_i^T$ // atualizar a matriz peso
      $\mathbf{b} \leftarrow \mathbf{b} + \alpha\,\mathbf{e}_i$ // atualizar o vetor bias
    end for
    t $\leftarrow$ t + 1
  end while
end procedure

Treinamento por Correção de Erros

Introdução

- Perceptron simples de uma camada (1958)

  1º modelo de RNA que envolveu o conceito de aprendizado

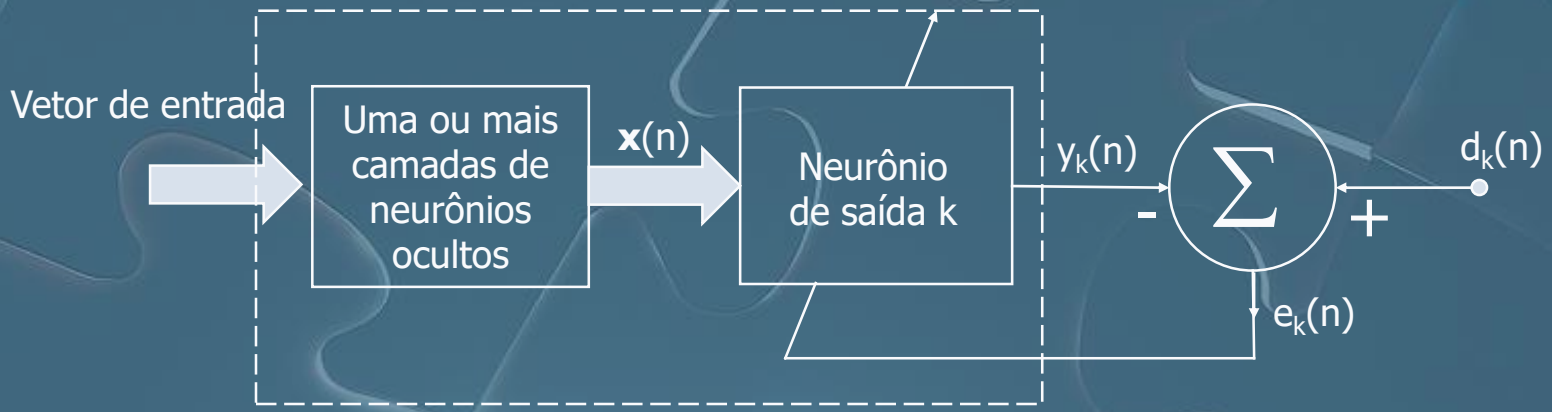
- Problemas mais complexos demandariam redes mais complexas

  Treinamento de redes torna-se um problema em aberto

  Backpropagation (1986)

Treinamento por Correção de Erros

Introdução



Sinal de erro $e_k(n)$ aciona um **mecanismo de controle**, cujo objetivo é aplicar uma seqüência de **ajustes corretivos** aos pesos ω do neurônio k

$$e_k = d_k(n) - y_k(n)$$

Treinamento por Correção de Erros

Ajuste dos pesos

$$\omega_{kj}(n+1) = \omega_{kj}(n) + \Delta\omega_{kj}(n)$$

Ajustes corretivos → aproximar a saída y da resposta desejada d

Isto é feito **minimizando-se** uma função de custo ou índice de desempenho E(n) em relação aos pesos ω

$$E(n) = \frac{1}{2} e_k^2(n)$$ ➡ Valor instantâneo da energia do erro

Ou seja

$$\Delta\omega_{ji}(n) \propto \frac{\partial E(n)}{\partial \omega_{ji}(n)}$$

e_j

Treinamento por Correção de Erros

A Regra Delta

A minimização de E(n) resulta na regra de aprendizagem conhecida como regra delta ou regra de Widrow-Hoff

A Regra Delta

$$\Delta\omega_{kj}(n) = \eta e_k x_j(n)$$

η taxa de aprendizagem
e_k sinal de erro ($e_k = d_k(n) - y_k(n)$)
x_k elemento do vetor de sinal **x**(n)

Os ajustes passo a passo persistem até o sistema atingir um **estado estável**

Treinamento por Correção de Erros

Redes com uma camada

Para redes com uma camada como o Perceptron ou o ADALINE, o ajuste de pesos assume uma forma simples

$$\Delta\omega_{kj}(n) = \eta e_k x_j(n)$$

Treinamento por Correção de Erros

Redes com múltiplas camadas

A extensão do método do gradiente ou regra delta para redes com múltiplas camadas é conhecida **Regra Delta Generalizada** ou ***Backpropagation***

Na 1ª fase do treinamento o sinal é propagado para a frente (fase *foward*) das entradas até a saída da rede

Como o valor desejado d para a camada de saída é conhecido, o erro pode ser calculado e os pesos da camada de saída ajustados

Treinamento por Correção de Erros

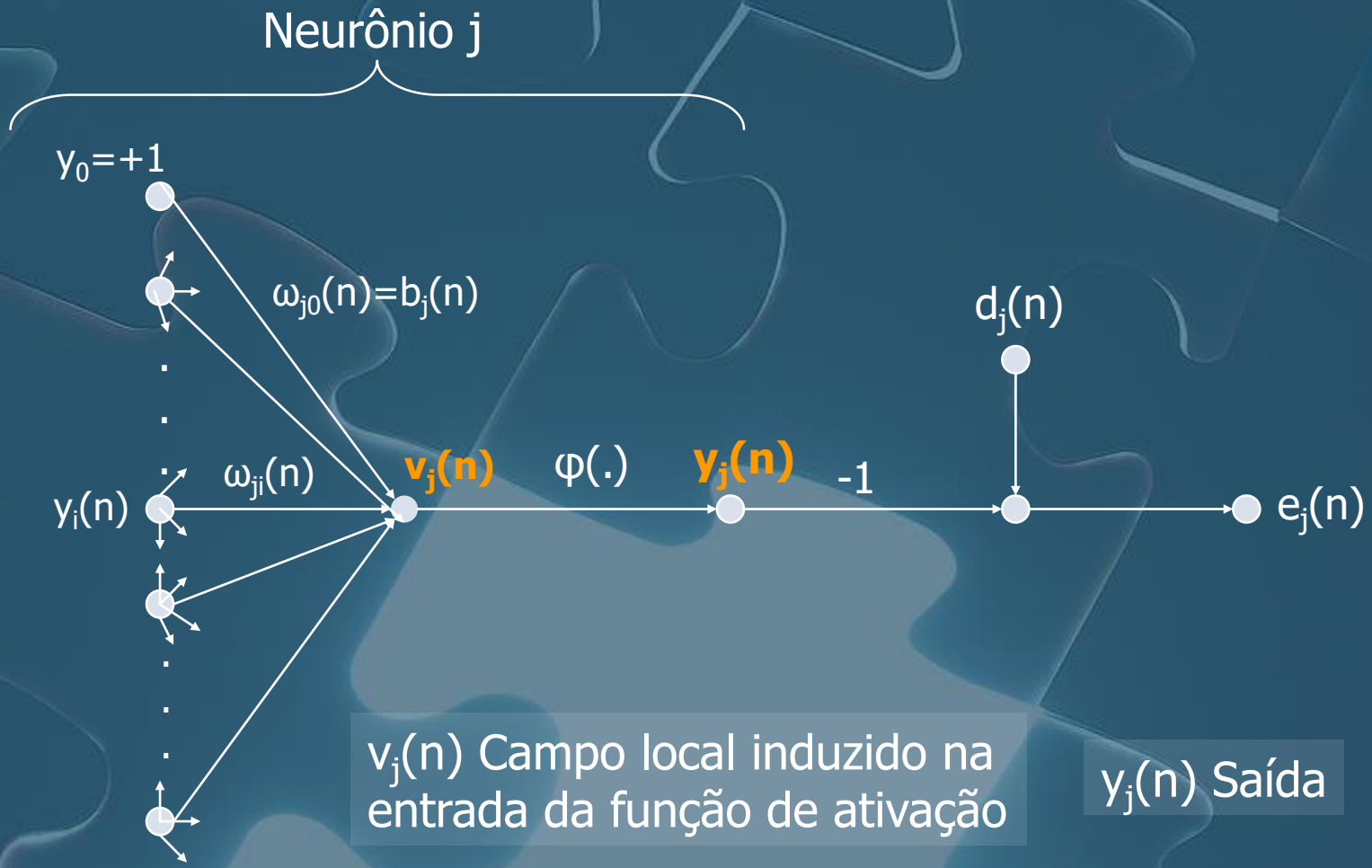
Redes com múltiplas camadas

Para as camadas intermediárias não existem valores desejados

O ajuste dos pesos das camadas intermediárias é feito através da propagação do erro para trás, o que caracteriza o treinamento com *backpropagation*

O algoritmo de retropropagação

Introdução



$v_j(n)$ Campo local induzido na entrada da função de ativação

$y_j(n)$ Saída

O algoritmo de retropropagação

Introdução

Para um neurônio

$e_k = d_k(n) - y_k(n)$ Sinal de erro

$E(n) = \frac{1}{2} e_k^2(n)$ Valor instantâneo da energia do erro

Para todos os neurônios

$$E(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$$

Se utilizarmos N padrões para treinamento da rede

$E_{med} = \frac{1}{N} \sum_{n=1}^{N} E(n)$ Energia média do erro

O algoritmo de retropropagação

Introdução

Para um total de m entradas (excluindo o bias)

$$v_j(n) = \sum_{i=0}^{m} \omega_{ij}(n) y_i(n)$$

$$y_j(n) = \varphi_j(v(n))$$

Ajuste dos pesos

$$\Delta\omega_{ji}(n) \propto \frac{\partial E(n)}{\partial \omega_{ji}(n)}$$

O algoritmo de retropropagação

Introdução

Pela regra da cadeia

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \omega_{ji}(n)}$$

$$\frac{\partial E(n)}{\partial e_j(n)} = \frac{\partial}{\partial e_j(n)} \left(\frac{1}{2} \sum_{k \in C} e_k^2(n) \right) = e_j(n)$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = \frac{\partial}{\partial y_j(n)} (d_j(n) - y_j(n)) = -1$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial}{\partial v_j(n)} \varphi(v_j(n)) = \varphi'(v_j(n))$$

$$\frac{\partial v_j(n)}{\partial \omega_{ji}(n)} = \frac{\partial}{\partial \omega_{ji}(n)} \left(\sum_{i=0}^{m} \omega_{ij}(n) y_i(n) \right) = y_i(n)$$

O algoritmo de retropropagação

Introdução

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n)$$

Assim a correção no peso pode ser escrita como:

$$\Delta\omega_{ji}(n) = -\eta \frac{\partial E(n)}{\partial \omega_{ji}(n)}$$

$$\Delta\omega_{ji}(n) = -\eta \frac{\partial E(n)}{\partial \omega_{ji}(n)} = -\eta e_j(n)\varphi_j'(v_j(n))y_i(n)$$

$$= -\eta\delta_j(n)y_i(n)$$

Gradiente local

O algoritmo de retropropagação

O gradiente local

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n)\varphi'_j(v_j(n))$$

Assim

(Correção de peso $\Delta\omega_{ji}(n)$) = (Parâmetro da taxa de aprendizagem η) · (Gradiente local $\delta_j(n)$) · (Sinal de entrada do neurônio j $y_i(n)$)

O fator chave é calcular o sinal de erro e_j!!!!!!!

O algoritmo de retropropagação

Casos 1 e 2

Existem 2 casos a serem considerados:

Caso 1

O neurônio j é um neurônio da **camada de saída** e

$e_j = d_j(n) - y_j(n)$

Fácil!!!!!!!!

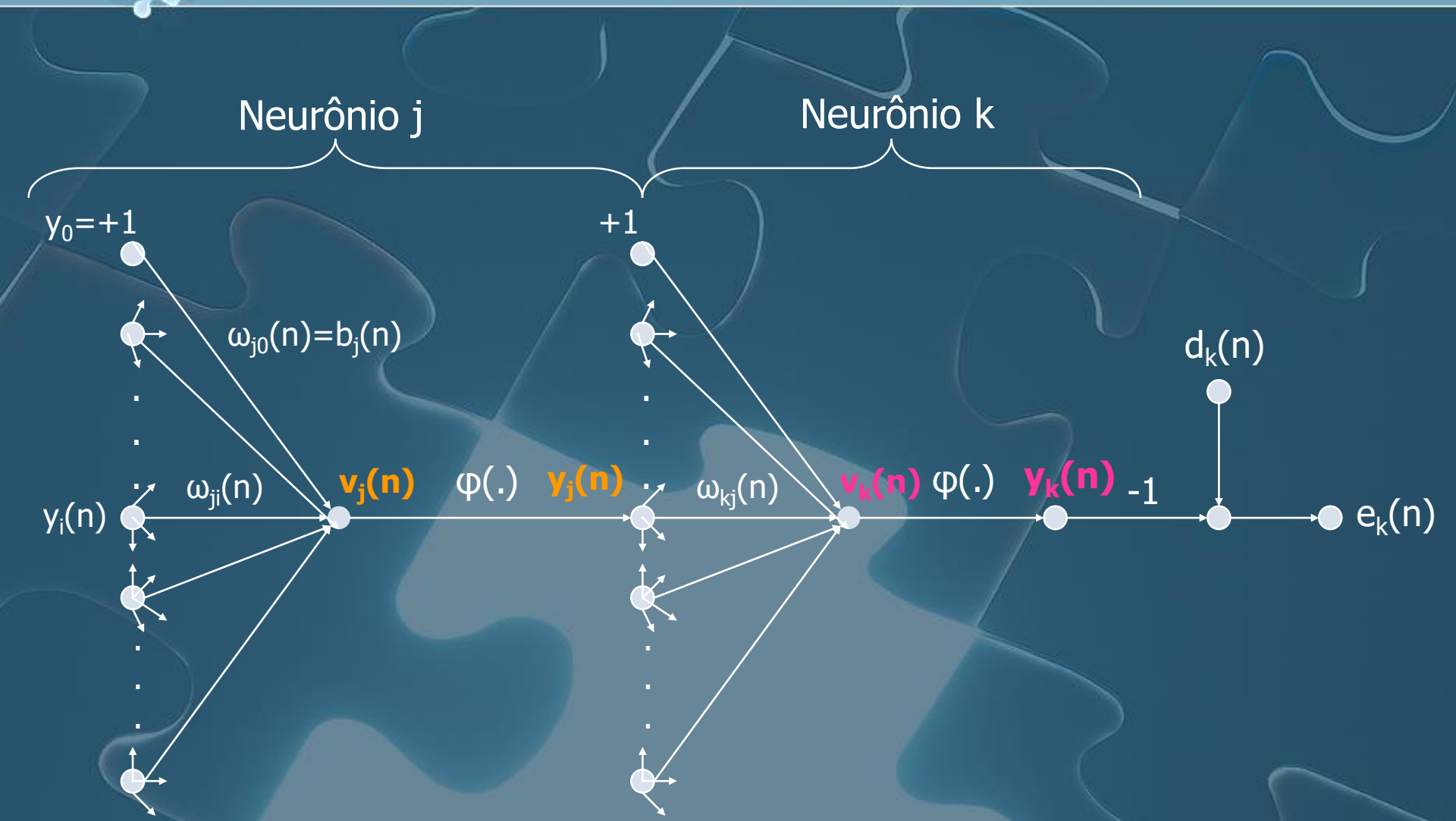
Caso 2

O neurônio j é um neurônio da **camada escondida**

Olhar melhor para este caso!!!!!!!

O algoritmo de retropropagação

Caso 2



O algoritmo de retropropagação

Gradiente local para o **neurônio oculto j**

Como calcular????????

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\varphi'_j(v_j(n))$$

$$v_j(n) = \sum_{i=0}^{m} \omega_{ij}(n) y_i(n) \qquad y_j(n) = \varphi_j(v(n))$$

Só tenho informação para E(n) na camada de saída, ou seja:

$$E(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n)$$

k é um neurônio da camada de saída

O algoritmo de retropropagação

Gradiente local para o **neurônio oculto j**

$$\frac{\partial E(n)}{\partial y_j(n)} = \frac{\partial}{\partial y_j} E(n) = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k\in C} e_k^2(n)\right) = \frac{\partial E(n)}{\partial e_k}\frac{\partial e_k}{\partial y_j}$$

$$= \sum_k e_k \frac{\partial e_k(n)}{\partial y_j}$$

Como calcular????????

Pela regra da cadeia

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j} = \sum_k e_k \frac{\partial e_k}{\partial v_k}\frac{\partial v_k}{\partial y_j}$$

O algoritmo de retropropagação

Gradiente local para o **neurônio oculto j**

$$\frac{\partial e_k}{\partial v_k} = \frac{\partial}{\partial v_k}(d_k(n) - y_k(n)) = \frac{\partial}{\partial v_k}(d_k(n) - \varphi_k(v_k(n))) = -\varphi'_k(v_k(n))$$

$$\frac{\partial v_k}{\partial y_j} = \frac{\partial}{\partial y_j(n)} \sum_{j=0}^{m} \omega_{kj}(n) y_j(n) = \omega_{kj}(n)$$

Assim

$$\delta_k(n)$$

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j} = \sum_k e_k \frac{\partial e_k}{\partial v_k} \frac{\partial v_k}{\partial y_j} = -\sum_k e_k \varphi'_k(v_k(n)) \omega_{kj}(n)$$

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k \delta_k(n) \omega_{kj}(n)$$

O algoritmo de retropropagação

Gradiente local para o **neurônio oculto j**

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\varphi'_j(v_j(n))$$

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k \delta_k(n)\omega_{kj}(n)$$

$$\delta_j(n) = \varphi'_j(v_k(n))\sum_k \delta_k(n)\omega_{kj}(n)$$

j neurônio oculto

k neurônio saída

O erro na camada oculta depende do erro na camada de saída!!

O algoritmo de retropropagação

Correção nos pesos

Assim

Correção de peso $\Delta\omega_{ji}(n)$ = Parâmetro da taxa de aprendizagem η · Gradiente local $\delta_j(n)$ · Sinal de entrada do neurônio j $y_i(n)$

→ 2 casos

Caso 1 - j neurônio saída

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n))$$

Caso 2 - j neurônio oculto

$$\delta_j(n) = \varphi'_j(v_k(n))\sum_k \delta_k(n)\omega_{kj}(n)$$

O algoritmo de retropropagação

Equação de ajuste dos pesos

$$\omega_{kj}(n+1) = \omega_{kj}(n) + \Delta\omega_{kj}(n) \qquad \Delta\omega_{kj}(n) = -\eta\delta_j(n)y_j(n)$$

Caso 1 - j neurônio saída

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n))$$

$$\omega_{kj}(n+1) = \omega_{kj}(n) + \eta e_j(n)\varphi'_j(v_j(n))y_j(n)$$

Caso 2 - j neurônio oculto

$$\delta_j(n) = \varphi'_j(v_k(n))\sum_k \delta_k(n)\omega_{kj}(n)$$

$$\omega_{kj}(n+1) = \omega_{kj}(n) + \eta\varphi'_j(v_j(n))\sum_k \delta_k(n)\omega_{kj}(k)y_j(n)$$

O algoritmo de retropropagação

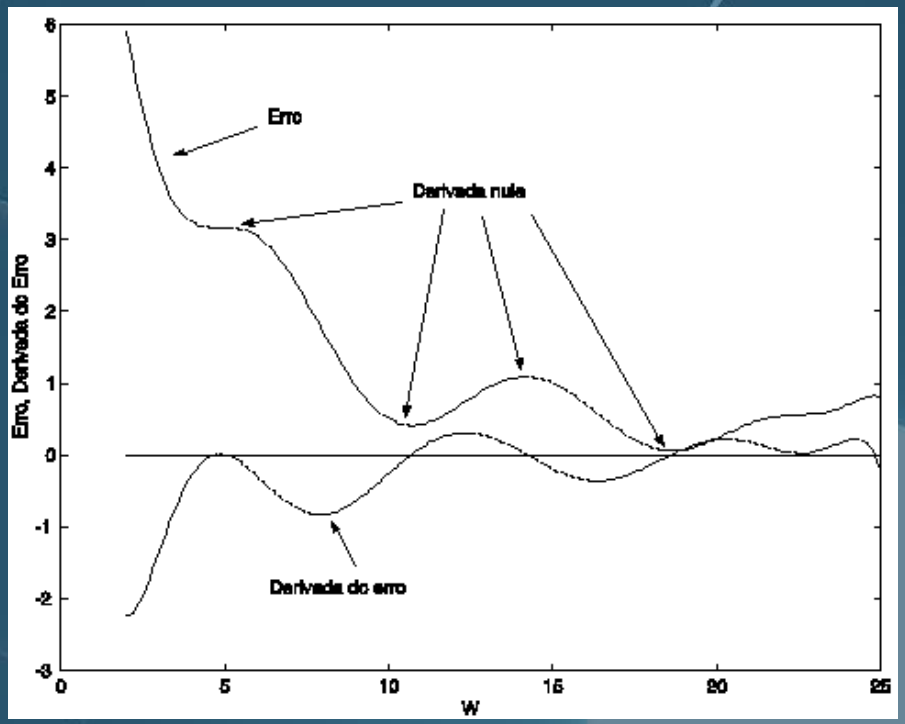
A superfície de erro

- A **forma irregular da superfície de erro** em razão das não-linearidades das funções de ativação causa **dificuldades para o treinamento** com algoritmos baseados no gradiente descendente, como o backpropagation.

- As não-linearidades podem provocar o surgimento de **mínimos locais** e **regiões planas** de **gradiente nulo**, dificultando o treinamento

O algoritmo de retropropagação

A superfície de erro



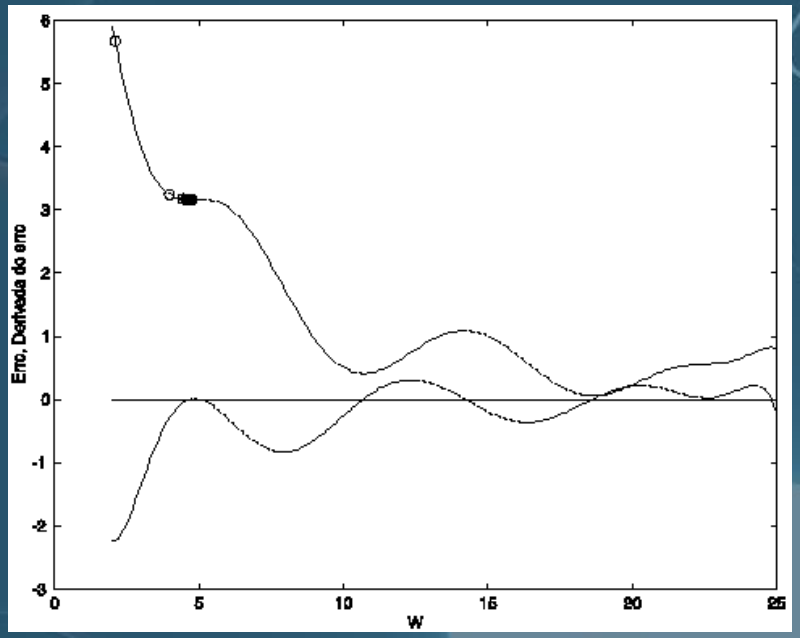
$$E(n) = \frac{1}{2} e_k^2(n)$$

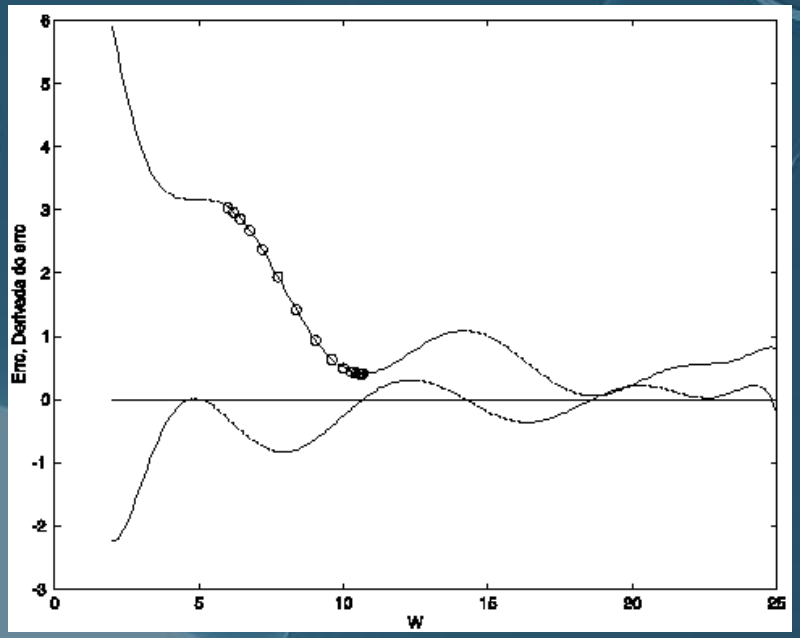
$$e_k = d_k(n) - y_k(n)$$

O algoritmo de retropropagação

A superfície de erro

Inicialização dos pesos



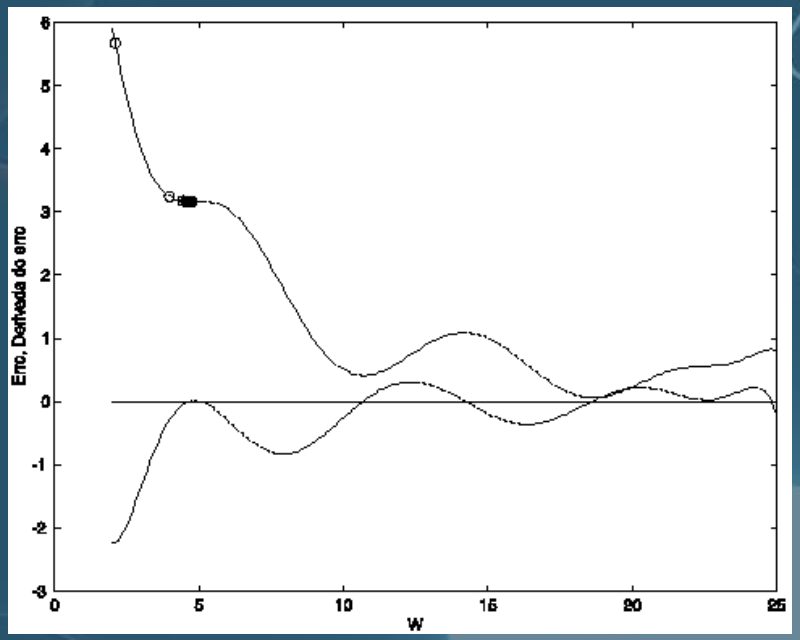


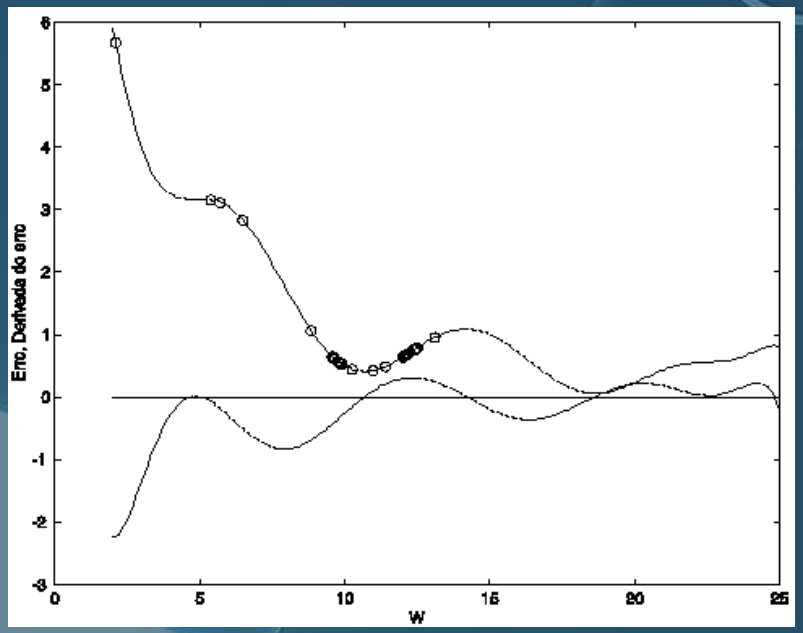
O desempenho depende das condições iniciais de treinamento

O algoritmo de retropropagação

A superfície de erro

A taxa de aprendizagem η





O desempenho depende da taxa de aprendizado

O algoritmo de retropropagação

A taxa de aprendizagem η

η peq. $\rightarrow$ maior sensibilidade às variações da superfície

η maior $\rightarrow$ evita regiões planas e mínimos locais

η influencia a velocidade de convergência

- η gde. $\rightarrow$ os ajustes de peso são de maior magnitude e o erro tende a diminuir mais rapidamente
- η peq. $\rightarrow$ convergência é lenta e o treinamento pode ficar presos a mínimos locais

Alguns algoritmos usam uma taxa de aprendizado adaptativa dependente do erro, começa em um valor grande e diminui próximo às regiões de mínimo

O algoritmo de retropropagação

O *momentum* α

- Alternativa para evitar regiões de gradiente nulo

$$\Delta\omega_{ij}(n) = -\eta \frac{\partial E}{\partial \omega_{ij}} + \alpha \Delta \omega_{ij}(n-1)$$

- Responsável pelo acumulo do histórico dos ajustes anteriores

  Resulta em um termo residual quando o ajuste é nulo por causa do gradiente

- Mínimos locais e regiões planas podem ser evitados

O algoritmo de retropropagação

O *momentum* α

- Se o **treinamento** passa por um **min. local** o **termo residual** forçará os ajustes na mesma direção dos ajustes **anteriores á passagem pelo mínimo**

- até que o gradiente se torne maior que o termo residual

- os ajustes passam a ser feitos novamente em direção ao mínimo, podendo passar por ele em direção contrária à anterior

- o treinamento permanecerá neste movimento em torno do mínimo até a convergência

O algoritmo de retropropagação

O *momentum* α

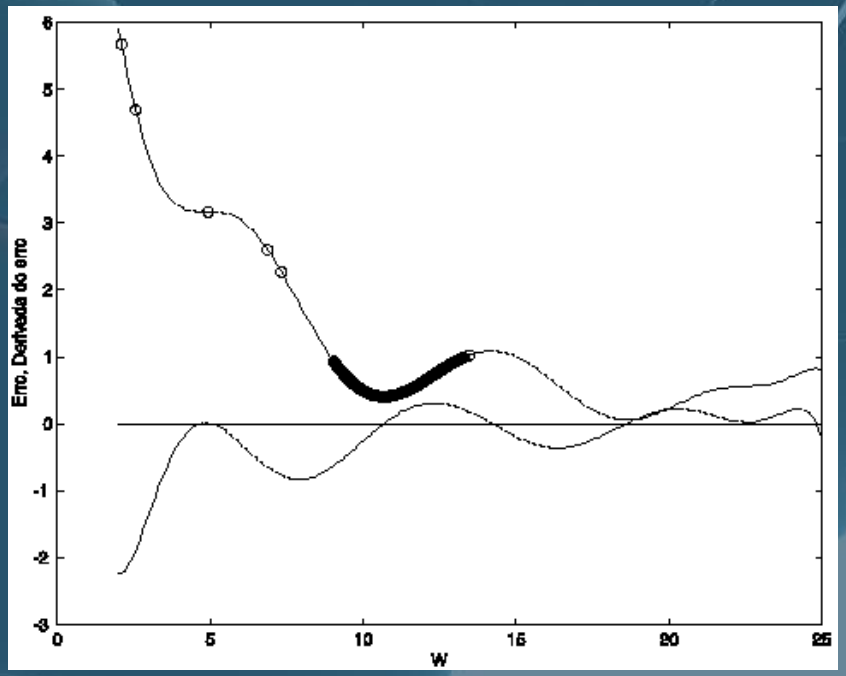
- Se o valor de α for grande o suficiente não haverá convergência para o mínimo local mas sim para o mínimo global

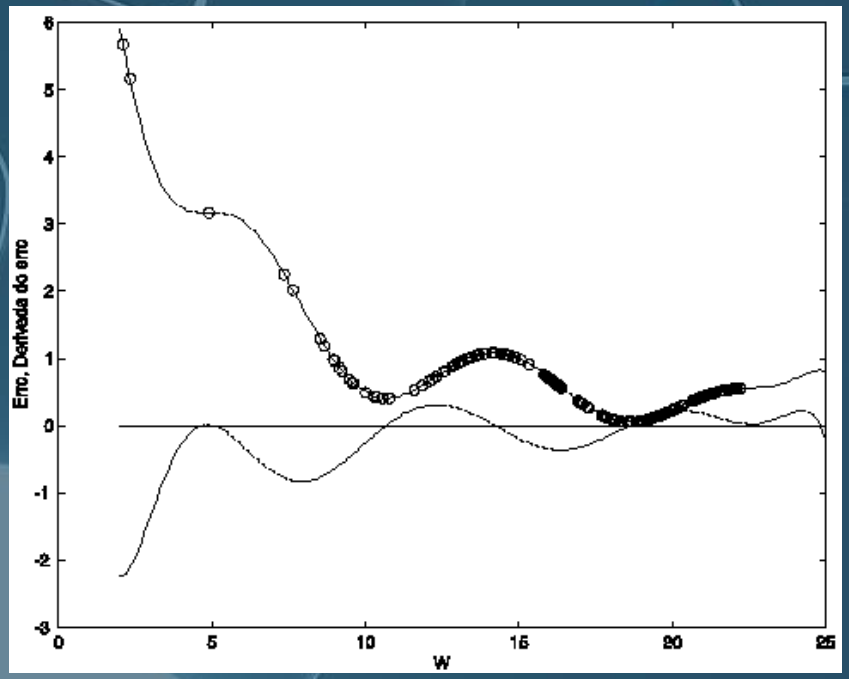
- pode resultar em melhor qualidade de treinamento

- Problema: é mais um parâmetro a ser ajustado pelo usuário

O algoritmo de retropropagação

O *momentum* α





Redes Neurais Artificiais

Aprendizado e Generalização

O objetivo principal do aprendizado em Redes Neurais Artificiais é a **obtenção de modelos com boa capacidade de generalização** tendo como base um conjunto de dados

Redes Neurais Artificiais

Aprendizado e Generalização

Problemas de aproximação, classificação e predição

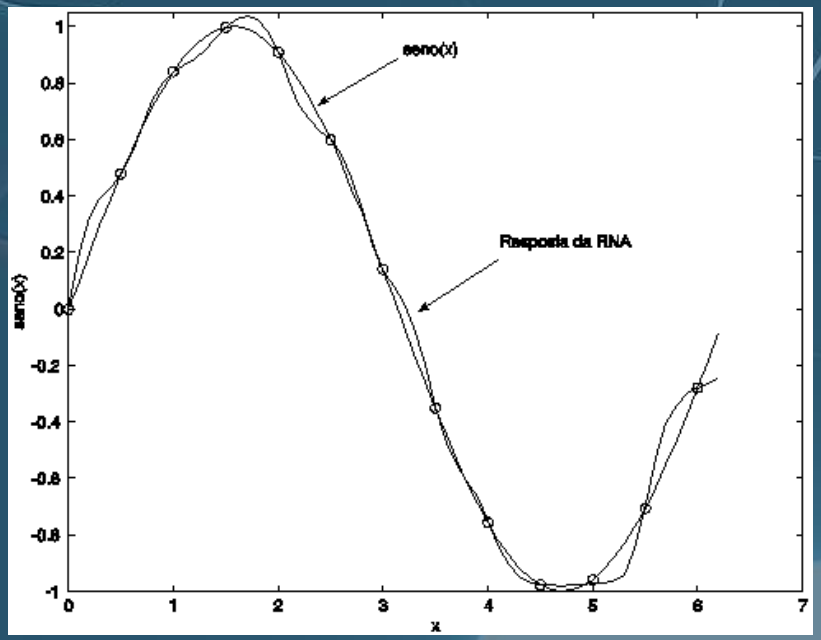
Conjunto de treinamento: pares (x,d)

Ajuste de pesos deve minimizar (y – d)

Minimização do erro pode não levar a resultados satisfatórios

Redes Neurais Artificiais

Aprendizado e Generalização



O erro foi minimizado, pois **a resposta da rede passa por todos os pontos**, mas a aproximação está distante da função *seno(x)*

É preciso mais que minimizar o erro para se obter boas respostas de generalização!!!!

Redes Neurais Artificiais

Aprendizado e Generalização

Treinamento

Minimizar o erro

Aproximar as funções geradoras dos dados

overfiting

A rede tem mais parâmetros (pesos) do que ela precisa

underfiting

A rede tem menos parâmetros (pesos) do que ela precisa

Encontrar o ajuste ideal

Redes Neurais Artificiais

Aprendizado e Generalização

Algoritmos que busquem o "*fit*" ideal

Métodos construtivos

Constroem a rede gradualmente. Iniciam em situação de *underfiting* e param em uma situação próxima ao *overfiting*

Algoritmos de poda

Visa a diminuição da estrutura da rede pela eliminação gradativa de pesos e neurônios

Redes Neurais Artificiais

Aprendizado e Generalização

Métodos construtivos e Algoritmos de poda

- Problemas
  - Não garantem a convergência da função geradora
  - Requerem o ajuste de parâmetros adicionais

Como encontrar o equilíbrio???

Redes Neurais Artificiais

Aprendizado e Generalização

Dilema entre polarização e variância

Modelos sobreparametrizados

Maior variabilidade na resposta

Modelos subparametrizados

Tem menos variabilidade e são polarizados para alguma resposta

O problema do treinamento de Redes Neurais é encontrar o equilíbrio entre polarização e variância

Redes Neurais Artificiais

Aprendizado e Generalização

Como encontrar este equilíbrio?

Associar algum outro fator (objetivo) além da minimização do erro ao treinamento

Sugestões

- Algoritmo *Weight Decay*

Um termo de custo que envolve a norma dos pesos é associado ao termo de energia do erro

$$E(n) = \frac{1}{2}(d_k(n) - y_k(n))^2 + \frac{\lambda}{2}\|\boldsymbol{\omega}\|^2$$

λ é uma parâmetro de treinamento que determina o peso da penalidade na obtenção da solução

Sugestões (cont.)

- SVMs (*Support Vector Machines*)

  Mapeiam os vetores de entrada em um espaço de mais alta dimensão (espaço de características), onde um hiperplano de separação é obtido para a resolução de problemas de classificação

  Um conceito fundamental no projeto de SVMs é o de **margem de separação** (associada ao erro permitido na classificação)

Redes Neurais Artificiais

Aprendizado e Generalização

Conclusão

O aprendizado de RNAs deve levar em conta algum outro objetivo, além do erro do conjunto de treinamento

Weight decay → norma dos pesos

SVMs → margem de classificação

O algoritmo de retropropagação

Simulação do backpropagation

Bibliografia

BRAGA AP, CARVALHO ACPLF, LUDERMIR, TB. Redes neurais artificiais. In: REZENDE SO. Sistemas inteligentes: princípios e aplicações. Barueri: Manole, 2003.

HAYKIN S. Redes Neurais: Princípios e prática. Porto Alegre: Bookman, 2001.