

# Redes Neurais Profundas – Deep Learning

## Aula 1 – Conceitos iniciais / Perceptron e Adaline

Prof. Anderson Soares

[www.inf.ufg.br/~anderson/deeplearning](http://www.inf.ufg.br/~anderson/deeplearning)

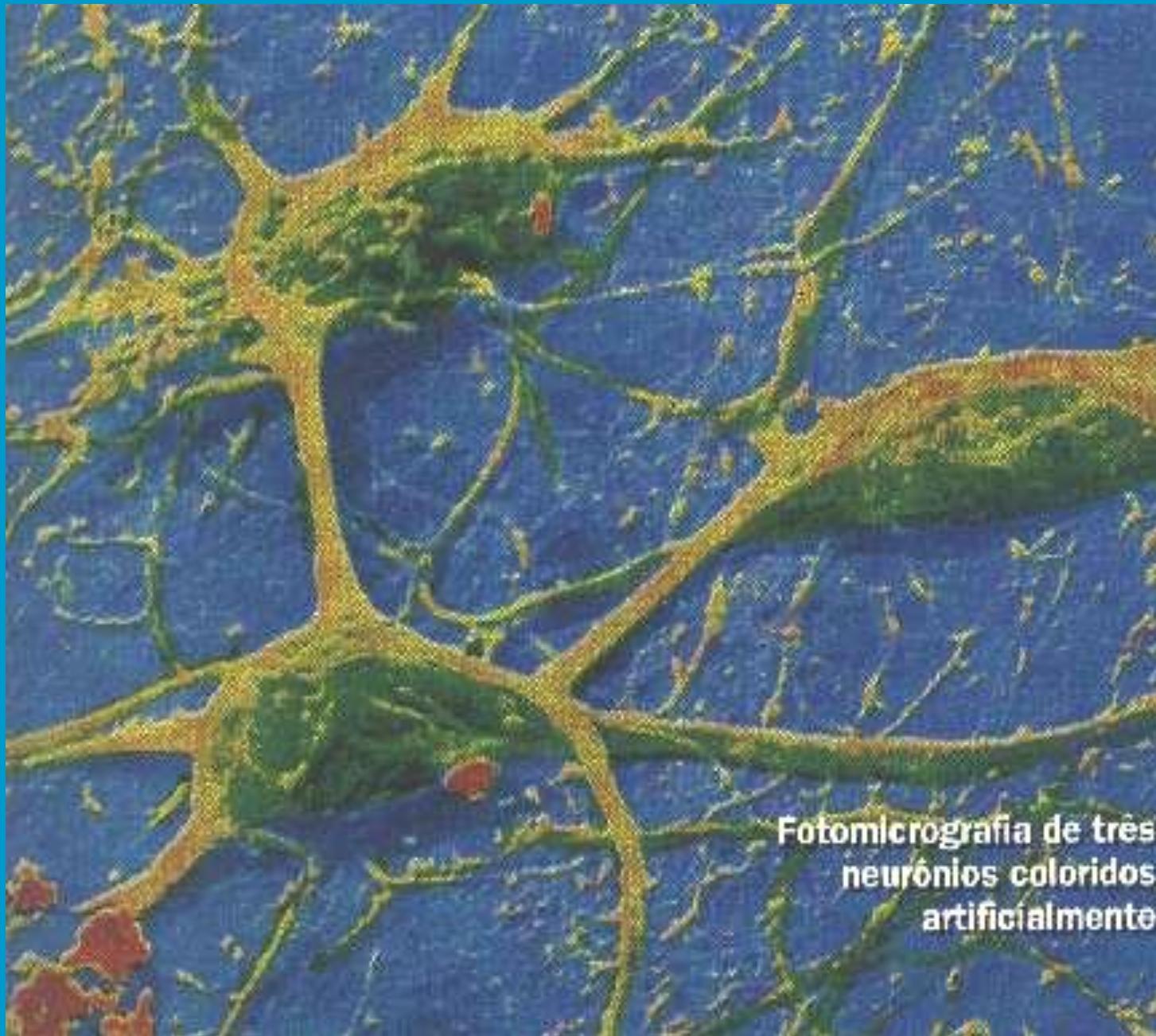
Instituto de Informática

Universidade Federal de Goiás

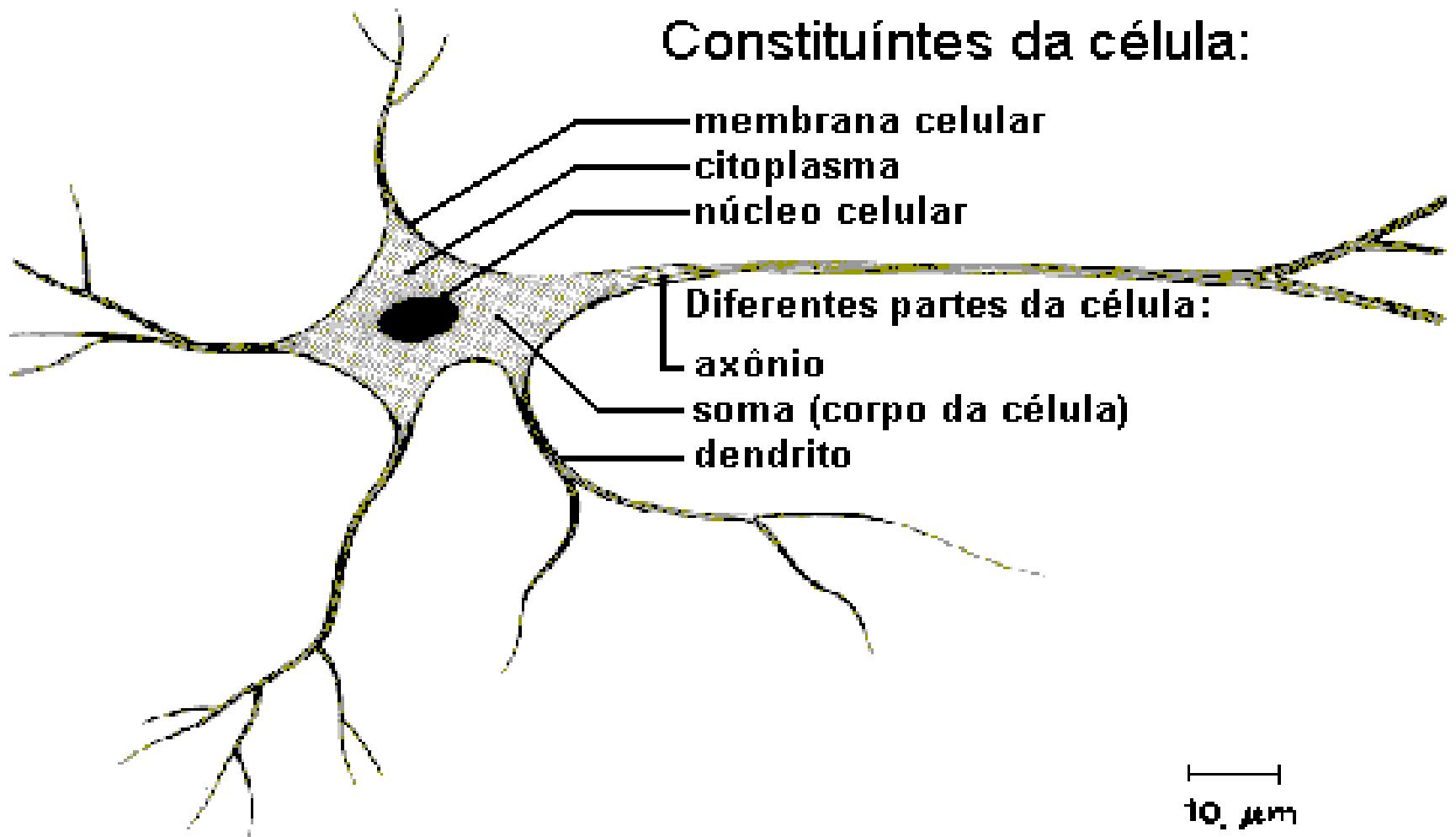
# Sumário

- Inspiração biológica
- Histórico
- Neurônio artificial
- Treinamento do neurônio

# Inspiração biológica



Fotomicrografia de três  
neurônios coloridos  
artificialmente

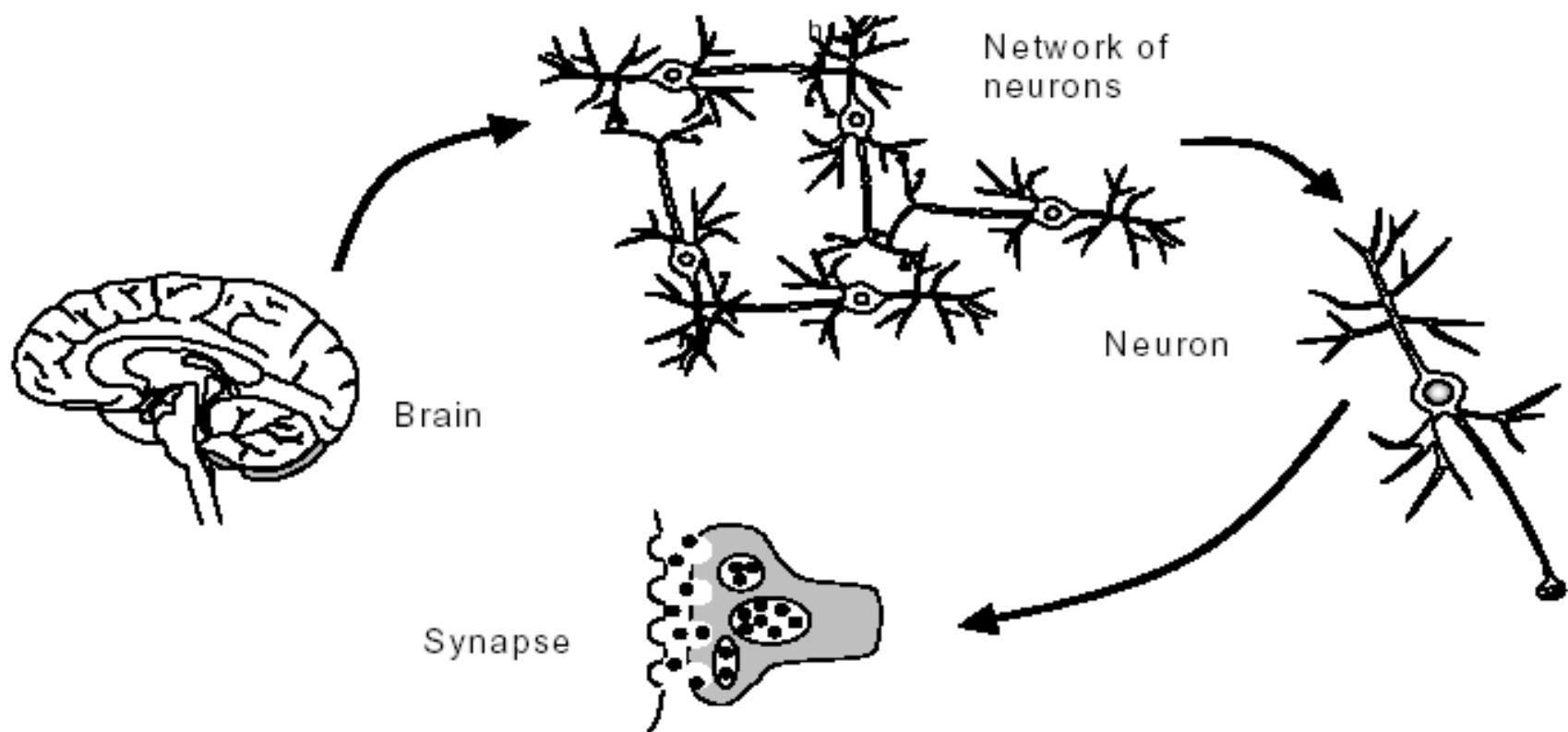


## **Os principais componentes dos neurônios são:**

- Os dentritos: recebem os estímulos transmitidos pelos outros neurônios;
- O corpo de neurônio, também chamado de soma: responsável por coletar e combinar informações vindas de outros neurônios;
- Axônio: constituído de uma fibra tubular que pode alcançar até alguns metros, e é responsável por transmitir os estímulos para outras células.

- Em média, cada neurônio forma entre mil e dez mil sinapses.
- O cérebro humano possui cerca de  $10^{11}$  neurônios, e o número de sinapses é de mais de  $10^{14}$ , possibilitando a formação de redes muito complexas.

# Cérebro humano

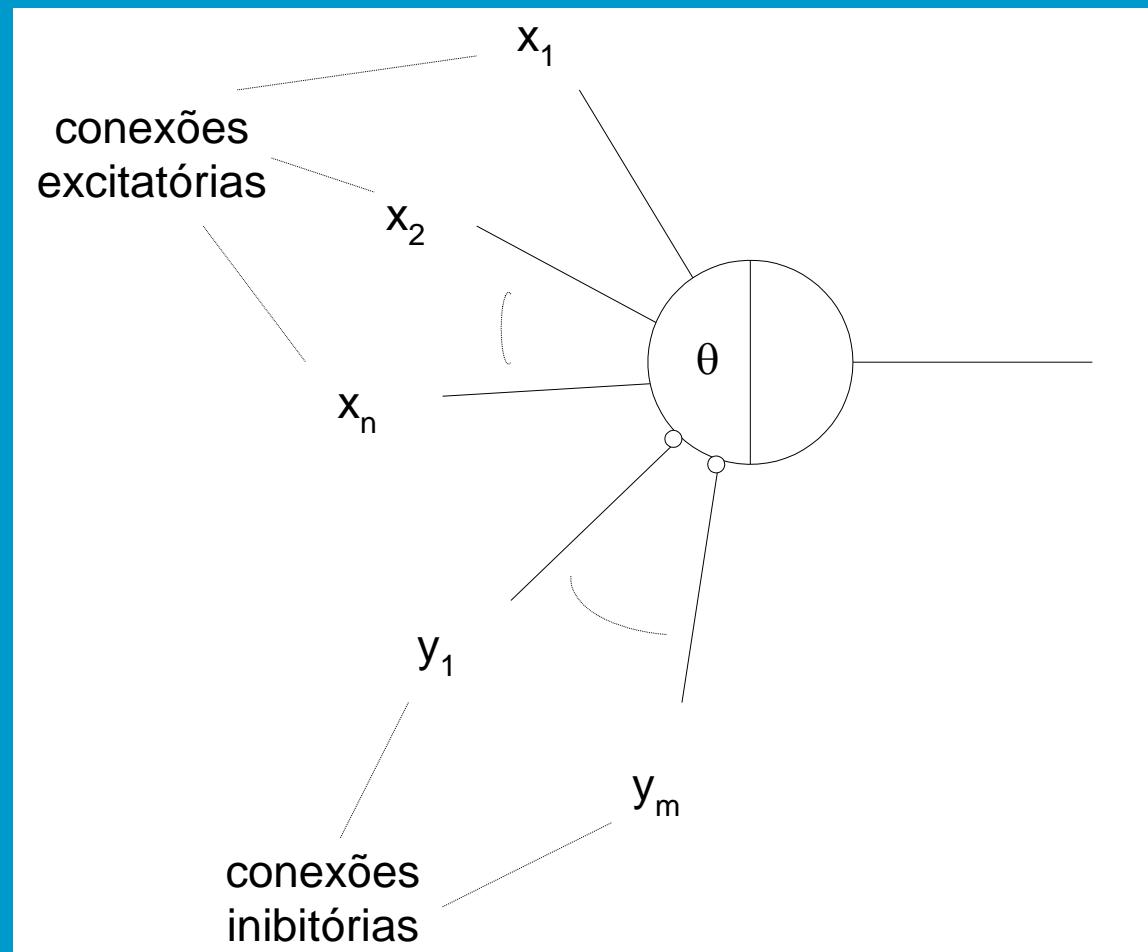


# Histórico



# O neurônio de McCulloch e Pitts

- Consiste basicamente de um neurônio que executa uma função lógica.
- Os nós produzem somente resultados binários e as conexões transmitem exclusivamente zeros e uns.
- As redes são compostas de conexões sem peso, de tipos excitatórios e inibitórios.
- Cada unidade é caracterizada por um certo limiar (*threshold*) q.



# Histórico (1949)

O psicólogo Donald Hebb, demonstrou que a capacidade da aprendizagem em redes neurais biológicas vem da alteração da eficiência sináptica, isto é, a conexão somente é reforçada se tanto as células pré-sinápticas quanto as pós-sinápticas estiverem excitadas;

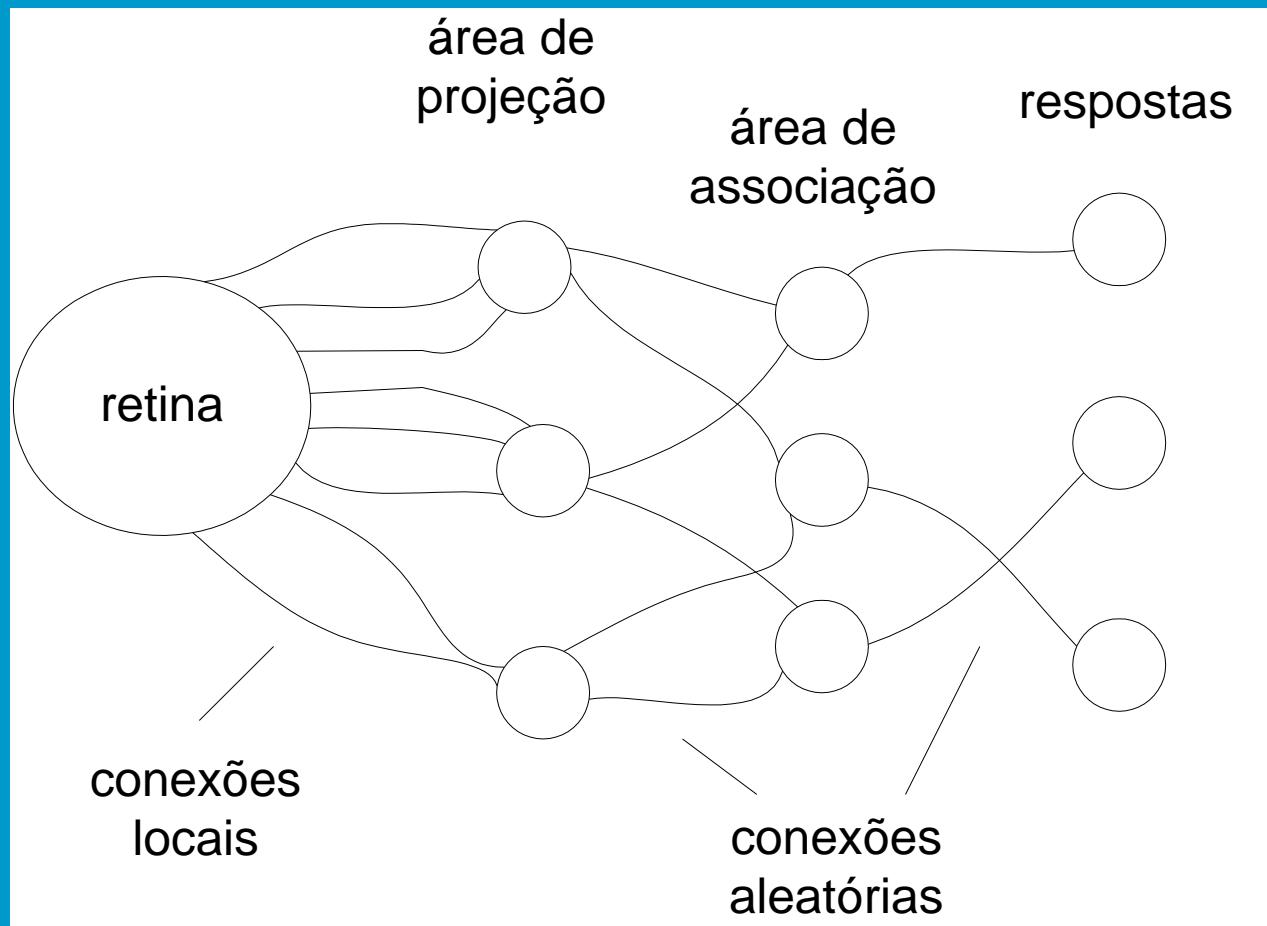
Hebb foi o primeiro a propor uma lei de aprendizagem específica para as sinapses dos neurônios.

# Histórico (1958)

Rosemblatt (1958) mostrou em seu livro (*Principles of Neurodynamics*) o modelo dos "Perceptrons".

Nele, os neurônios (Perceptrons) eram organizados em camada de entrada e saída, onde os pesos das conexões eram adaptados a fim de se atingir a eficiência sináptica usada no reconhecimento de caracteres.

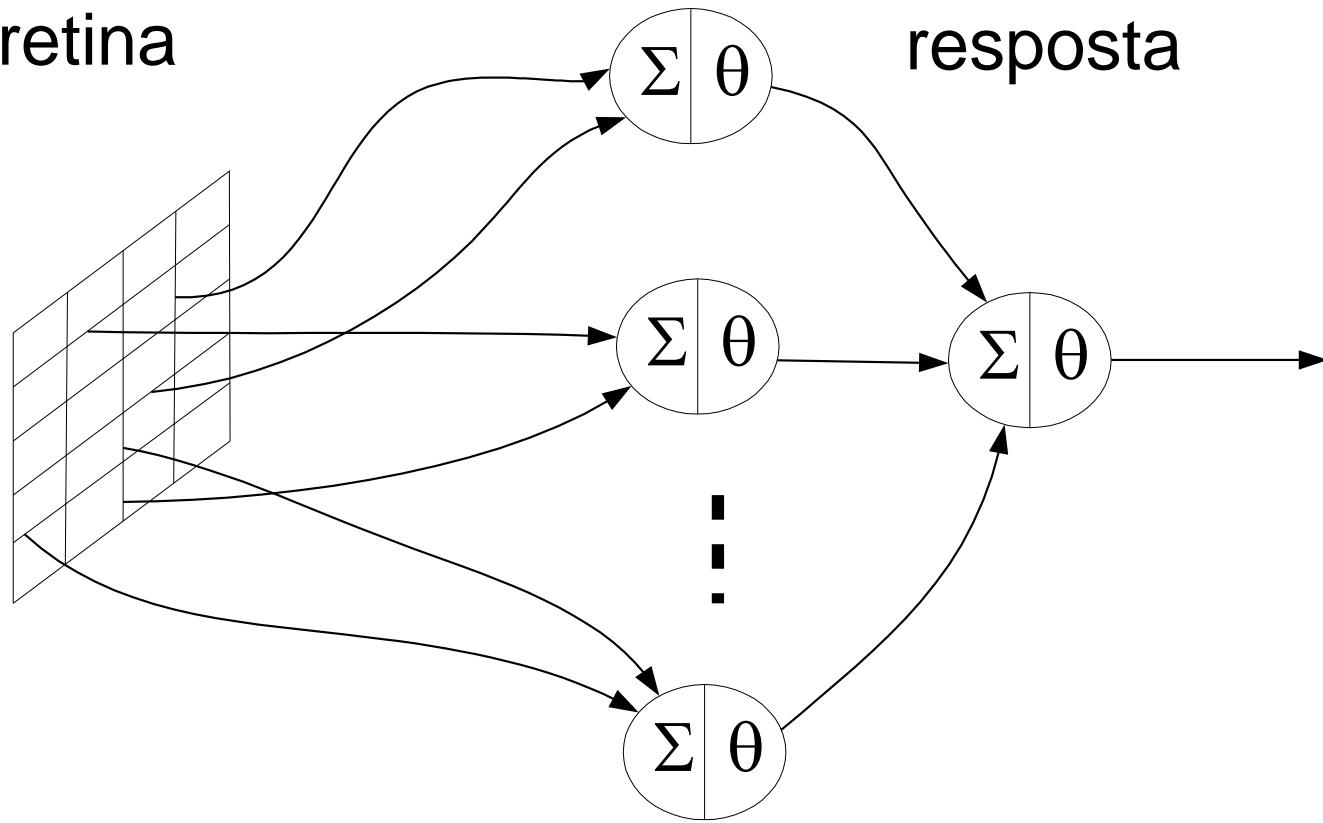
# Perceptron Clássico – Rosenblatt (1958)



associação

retina

resposta



# Histórico (1960)

Em 1960 surgiu a rede ADALINE (ADAptive LInear NEtwork) e o MADALINE (Many ADALINE), proposto por Widrow e Hoff.

O ADALINE/MADALINE utilizou saídas analógicas em uma arquitetura de três camadas.

# Histórico (1969)

- Foi constatado por Minsky & Papert que um neurônio do tipo Perceptron só é capaz de resolver problemas com dados de classes linearmente separáveis.

# Histórico (1960-1970)

Muitos historiadores desconsideraram a existência de pesquisa nessa área nos anos 60 e 70.

# Histórico (1982)

Retomada das pesquisas com a publicação dos trabalhos do físico e biólogo Hopfield relatando a utilização de redes simétricas para otimização, através de um algoritmo de aprendizagem que estabilizava uma rede binária simétrica com realimentação.

# Histórico (1986)

- Rumelhart, Hinton e Williams introduziram o poderoso método de treinamento denominado “Backpropagation”.
- Rumelhart e McClelland escreveram o livro “Processamento Paralelo Distribuído: Explorações na Microestrutura do Conhecimento”.

# Histórico (1988)

- Broomhead e Lowe descreveram um procedimento para o projeto de uma rede neural (feedforward) usando funções de base radial (Rede de Base Radial – RBF).

# Histórico (1995)

## ■ Yan Lecun – Redes Neurais Convolucionais

Convolutional Networks for Images, Speech, and  
Time-Series

Yann LeCun

Rm 4G332, AT&T Bell Laboratories

101 Crawfords Corner Road

Holmdel, NJ 07733

Yoshua Bengio

Dept. Informatique et Recherche

Opérationnelle, Université de Montréal,

Montreal, Qc, Canada, H3C-3J7

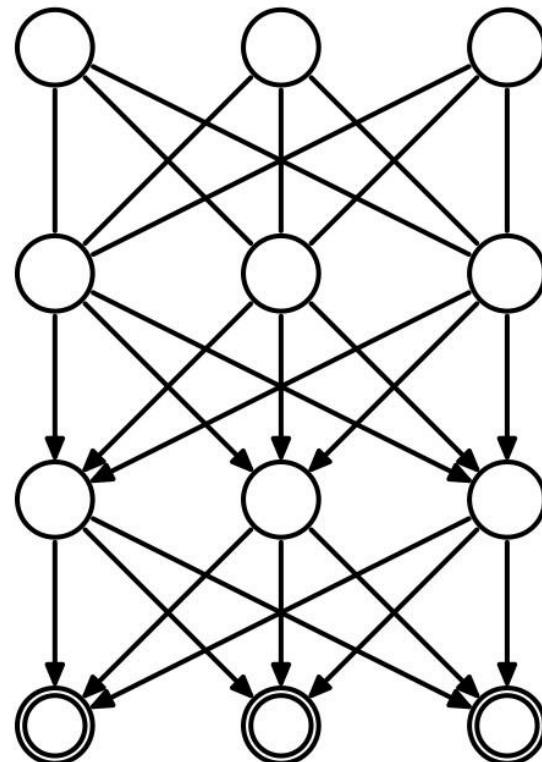
yann@research.att.com

bengioy@iro.umontreal.ca

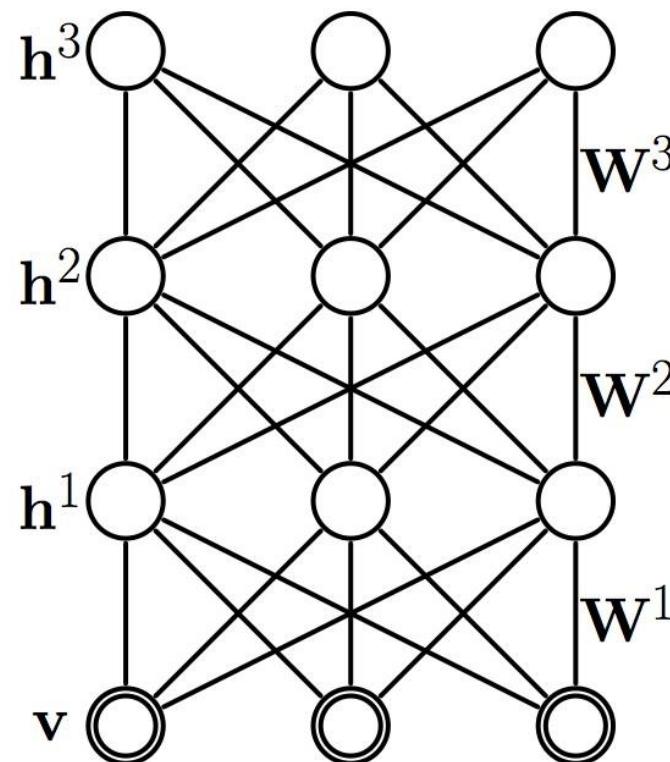
# Histórico (2006)

## ■ Deep Belief Neural Network

Deep Belief Network



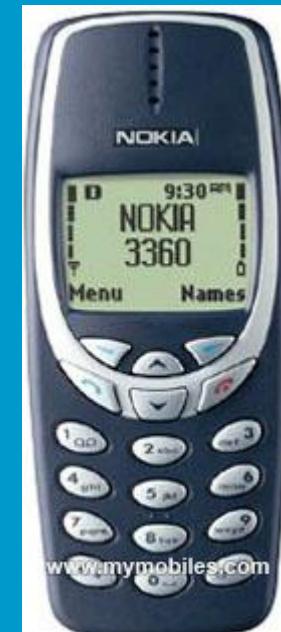
Deep Boltzmann Machine



# Histórico (2012)

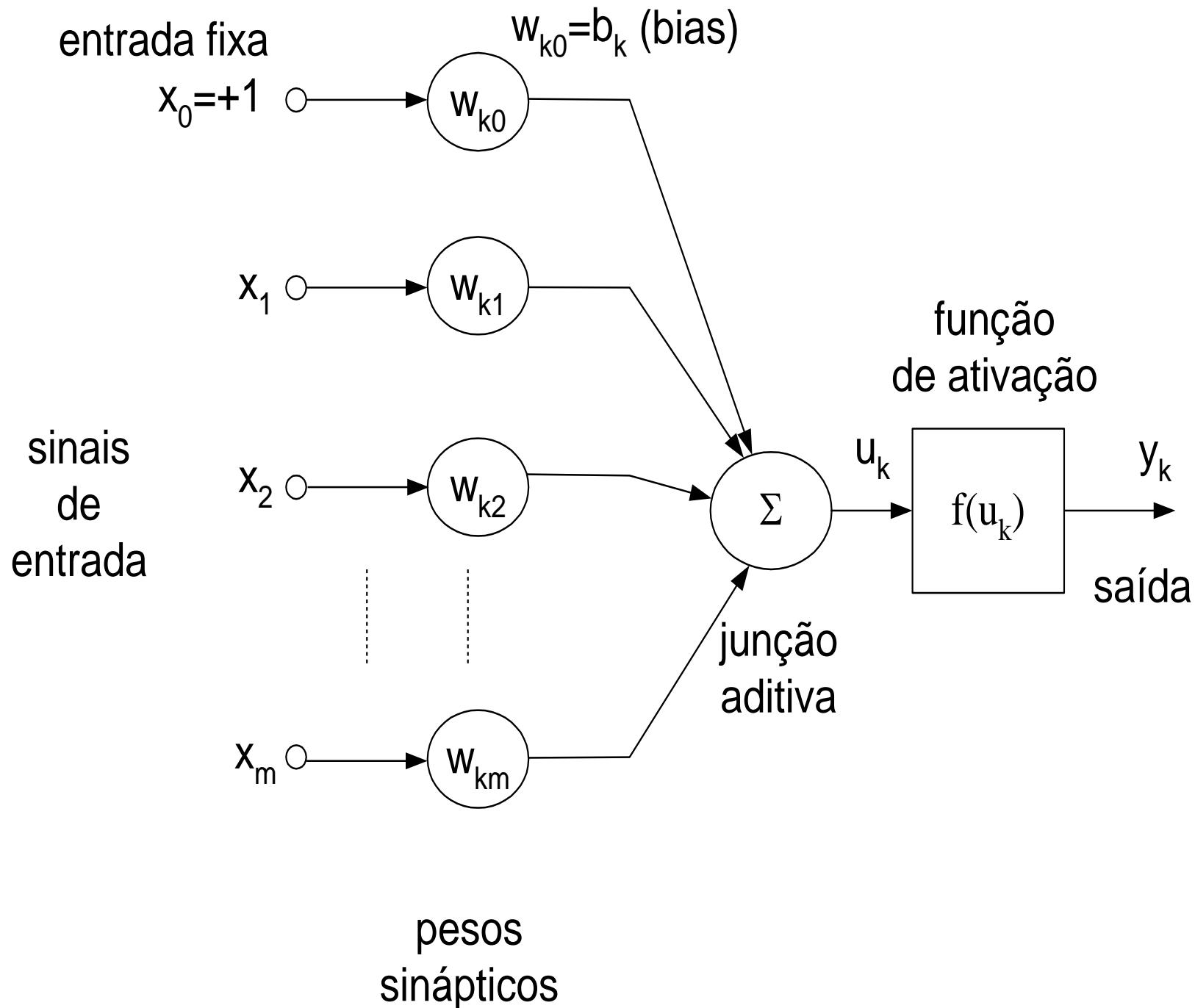
- Rede Neural Profunda Convolucional
- GooGleNet - 2014
- ResNet - 2014
- GAN - 2015

# Neurônio artificial – Resgatando as origens



# Componentes do neurônio artificial

- Entradas  $\{x_1, x_2, x_n\}$
- Pesos sinápticos  $\{w_1, w_2, w_n\}$
- Combinador  $\sum$
- Potencial de ativação  $\{u\}$
- Função de ativação  $f(u)$
- Sinal de saída  $y$



# Princípio de funcionamento

A operação de um neurônio artificial se resume em:

- Sinais são apresentados à entrada ( $x_1$  à  $x_m$ );
- Cada sinal é multiplicado por um peso que indica sua influência na saída da unidade ( $w_k$ );
- É feita a soma ponderada dos sinais que produz um nível de atividade ( $u_k$ );
- A função de ativação  $f(u_k)$  tem a função de limitar a saída e introduzir não-linearidade ao modelo.
- O bias  $b_k$  tem o papel de aumentar ou diminuir a influência do valor das entradas.
- É possível considerar o bias como uma entrada de valor constante 1, multiplicado por um peso igual a  $b_k$ .

# Expressão matemática do neurônio artificial

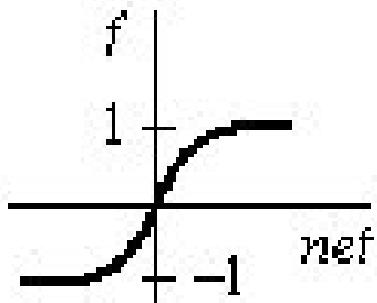
- Matematicamente a saída pode ser expressa por:

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

ou considerando o bias como entrada de valor  $x_0=1$  e peso  $w_{k0}=b_k$ ,

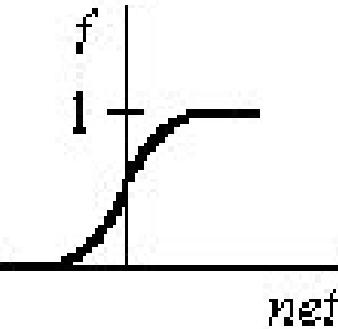
$$y_k = f(u_k) = f\left(\sum_{j=0}^m w_{kj}x_j\right)$$

# Funções de ativação



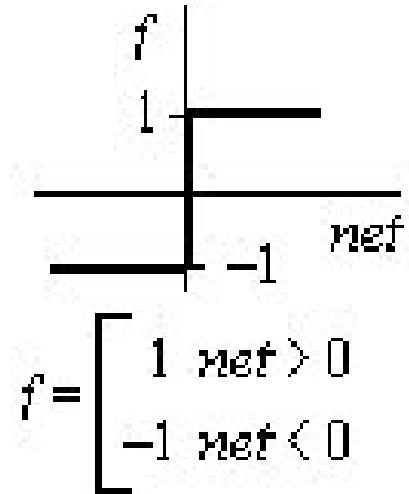
$$f = \tanh(\alpha \cdot \text{net})$$

tanh



$$f = \frac{1}{1 + \exp(-\alpha \cdot \text{net})}$$

logistic

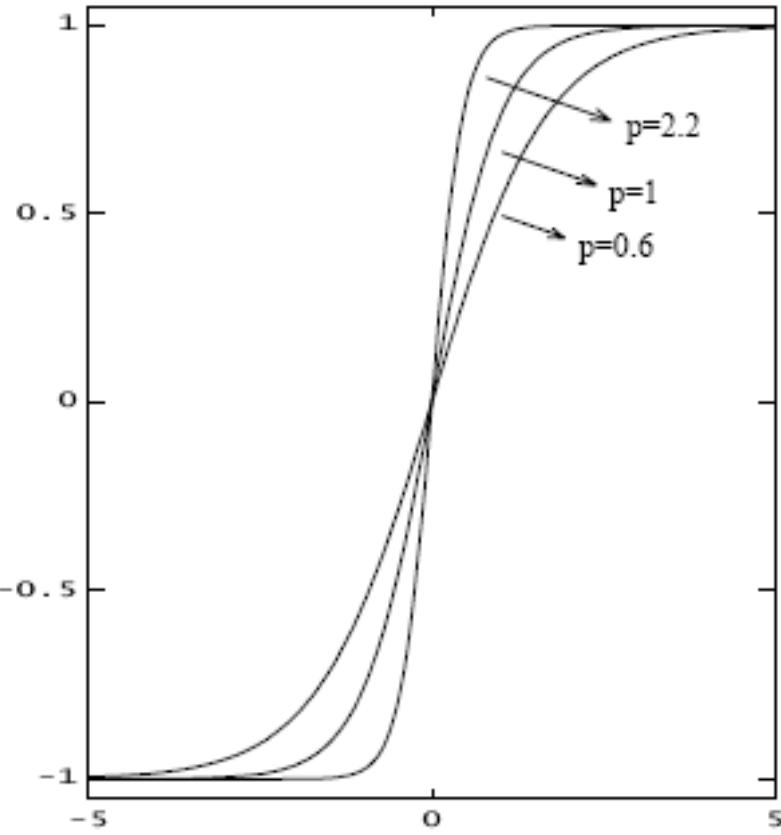


$$f = \begin{cases} 1 & \text{net} > 0 \\ -1 & \text{net} < 0 \end{cases}$$

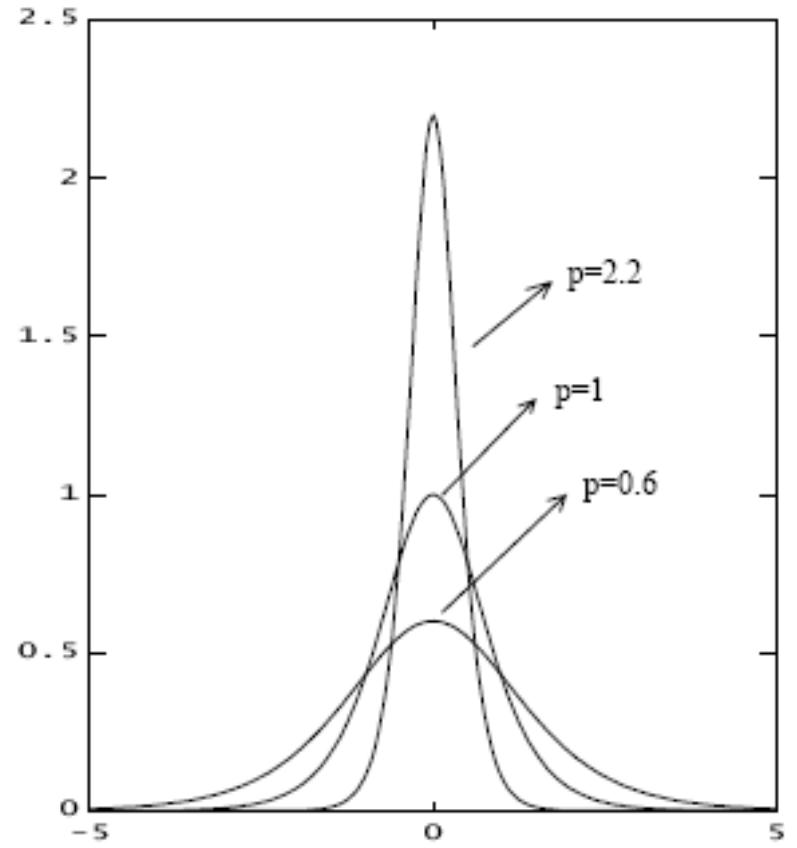
threshold

# Função de ativação tangente hiperbólica

$$f(u_k) = \tanh(pu_k) = \frac{e^{pu_k} - e^{-pu_k}}{e^{pu_k} + e^{-pu_k}}$$
$$\frac{\partial f}{\partial u_k} = p(1 - u_k^2) > 0$$



(a)

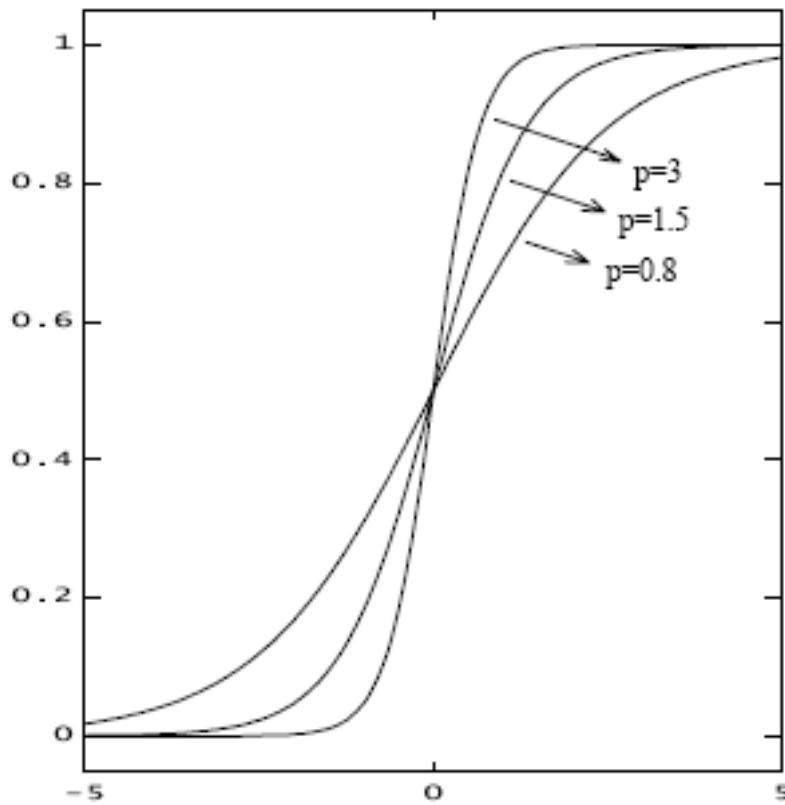


(b)

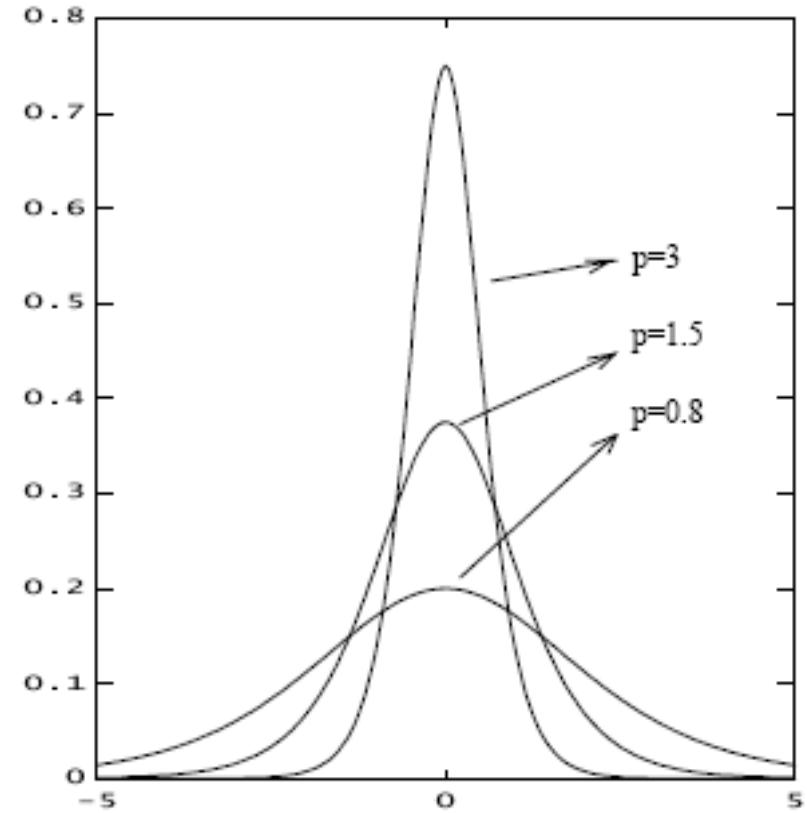
Tangente hiperbólica (a) e sua derivada (b).

# Função de ativação logística (sigmóide)

$$f(u_k) = \frac{e^{pu_k}}{e^{pu_k} + 1} = \frac{1}{1 + e^{-pu_k}}$$
$$\frac{\partial f}{\partial u_k} = pu_k(1 - u_k) > 0$$



(a)



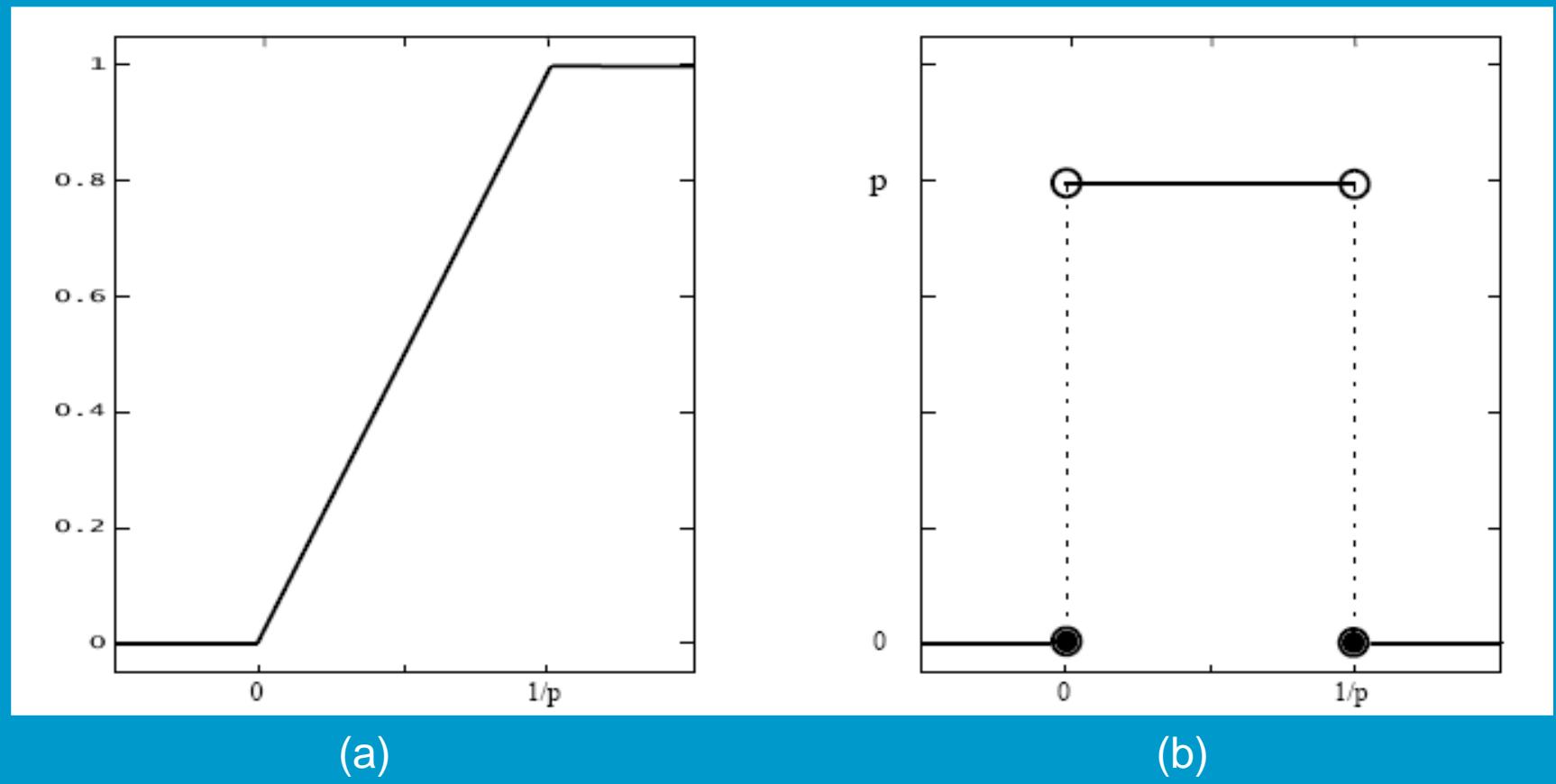
(b)

Sigmóide (a) e sua derivada (b).

# Função de ativação semi-linear

$$f(u_k) = \begin{cases} 1 & \text{se } pu_k \geq 1 \\ pu_k & \text{se } 0 < pu_k < 1 \\ 0 & \text{se } pu_k \leq 0 \end{cases}$$

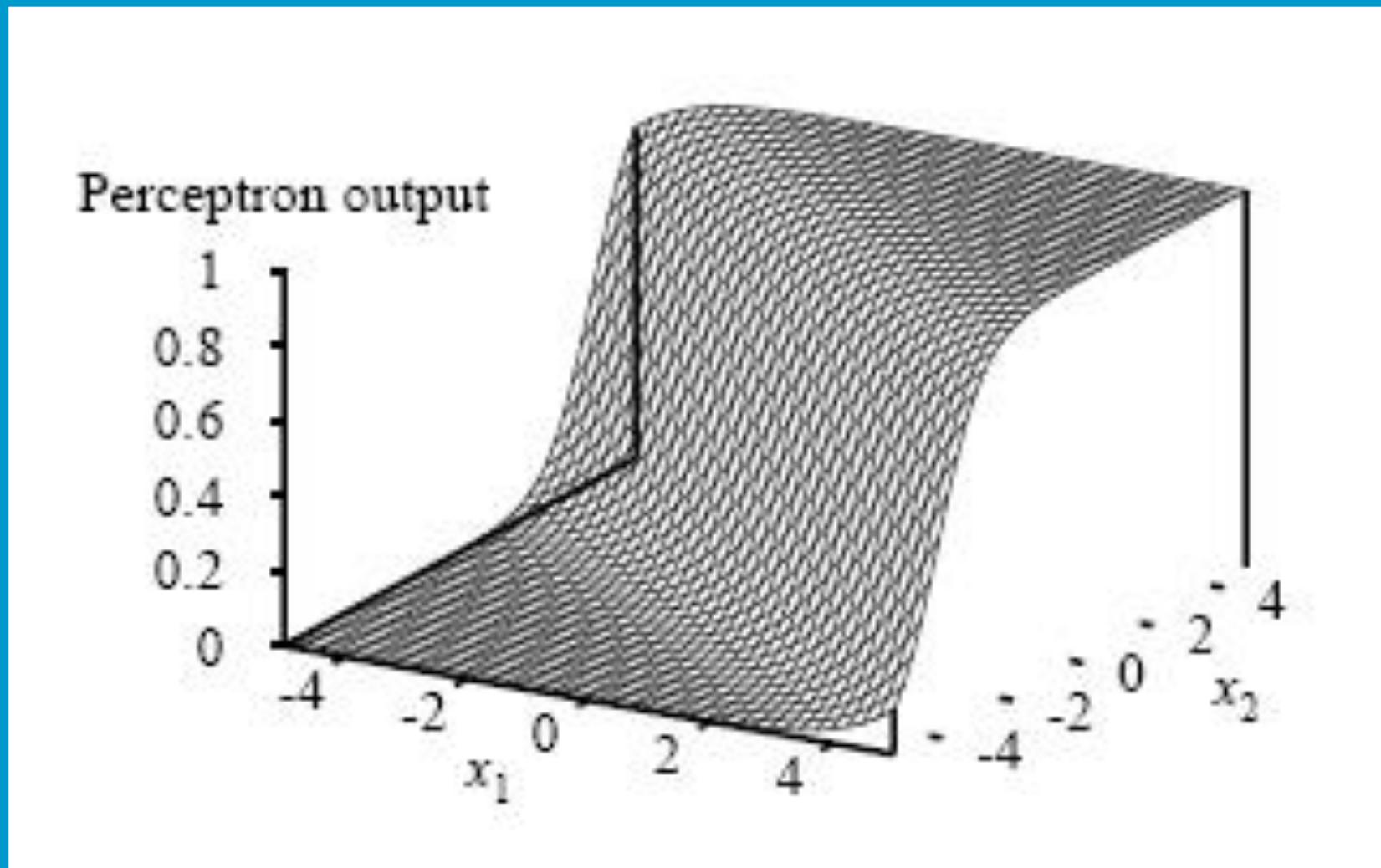
$p$  constante e positivo



Função de ativação semi-linear (a) e sua derivada (b).

# Resposta do neurônio (saída)

Considerando um neurônio artificial com duas entradas ( $x_1, x_2$ ) e função de ativação sigmóide:



# Separação Linear

Sabe-se que se formarmos uma combinação linear de duas variáveis, e igualá-la a um número, então os pontos no espaço bidimensional podem ser divididos em três categorias:

a) pontos pertencentes à linha com coordenadas tais que

$$w_1 \cdot x_1 + w_2 \cdot x_2 = q$$

b) pontos em um lado da linha tem coordenadas tais que

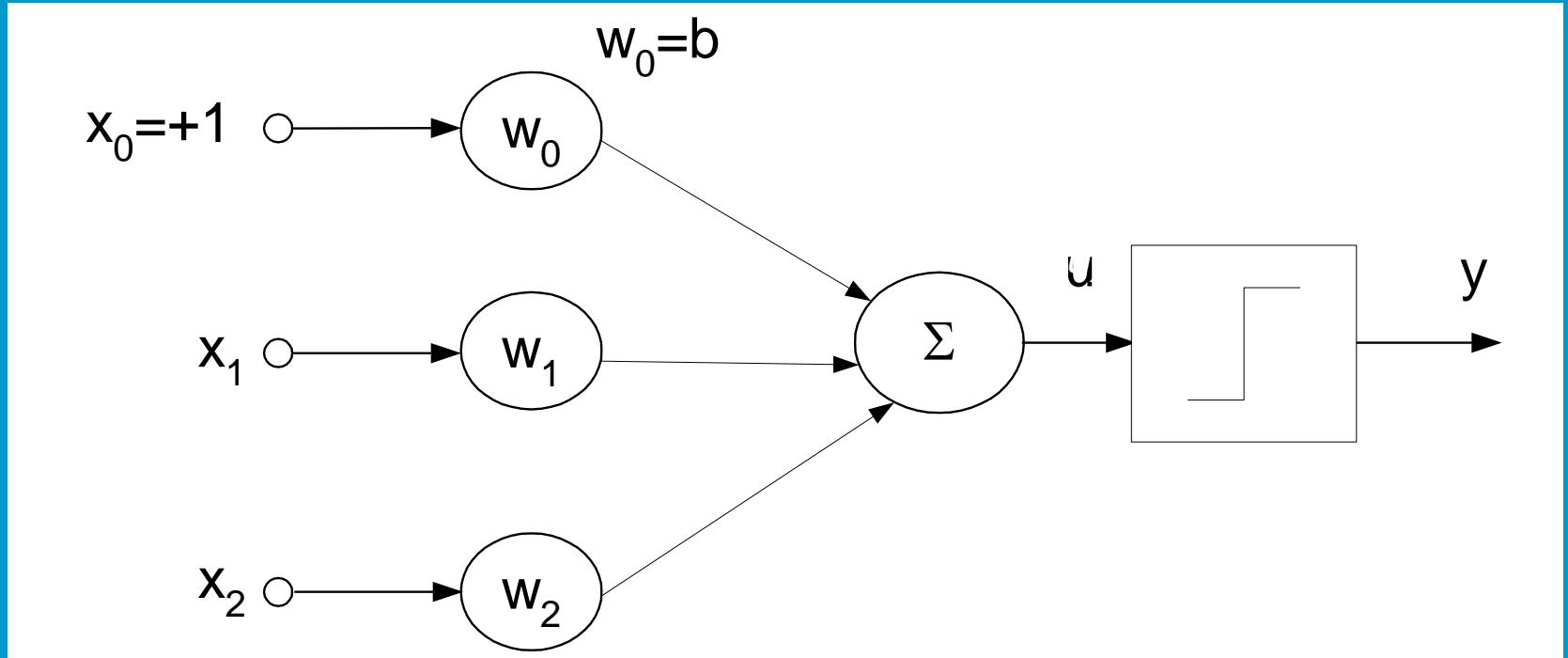
$$w_1 \cdot x_1 + w_2 \cdot x_2 < q$$

c) pontos no outro lado da linha tem coordenadas tais que

$$w_1 \cdot x_1 + w_2 \cdot x_2 > q.$$

Nota: bias  $b = -q$ , pois:

$$y = f(w_1 x_1 + w_2 x_2 + w_0), \text{ onde } w_0 = b$$



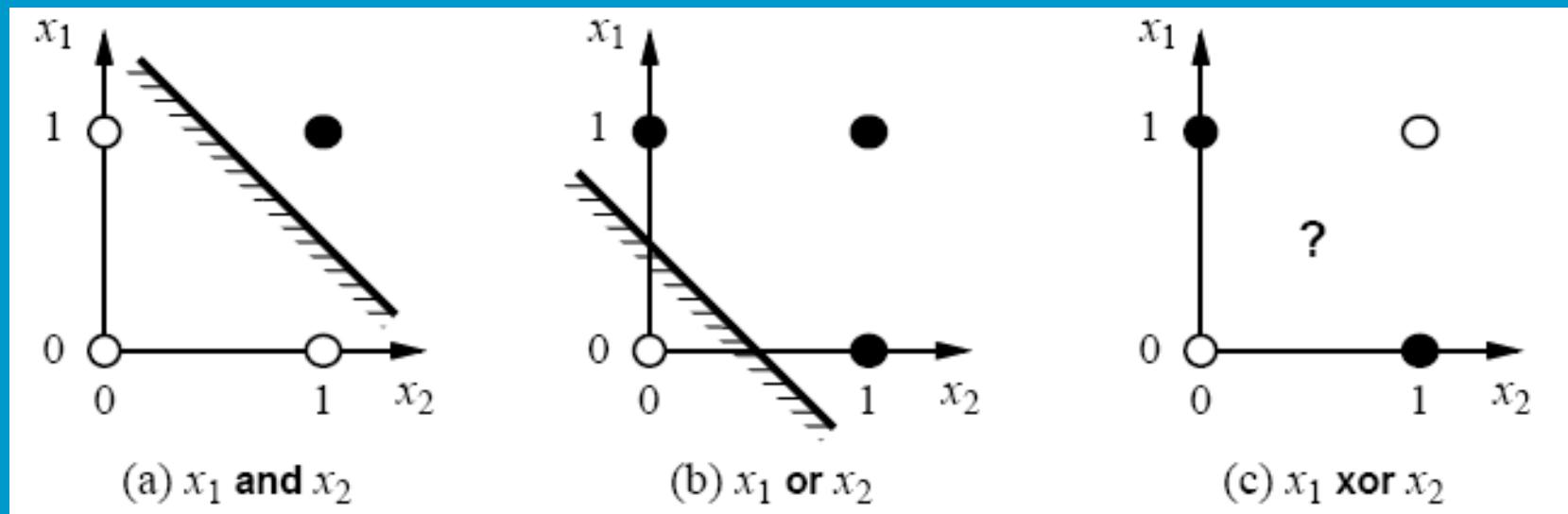
$$y = f(w_1x_1 + w_2x_2 + w_0), \text{ sendo } \begin{cases} f(u) = 1 & \text{se } u \geq 0 \\ f(u) = 0 & \text{se } u < 0 \end{cases}$$

Com os parâmetros  $w_0$ ,  $w_1$  e  $w_2$ , a função  $f(u)$  separa o espaço de entradas em duas regiões, usando uma linha reta dada por:

$$w_1x_1 + w_2x_2 + w_0 = 0$$

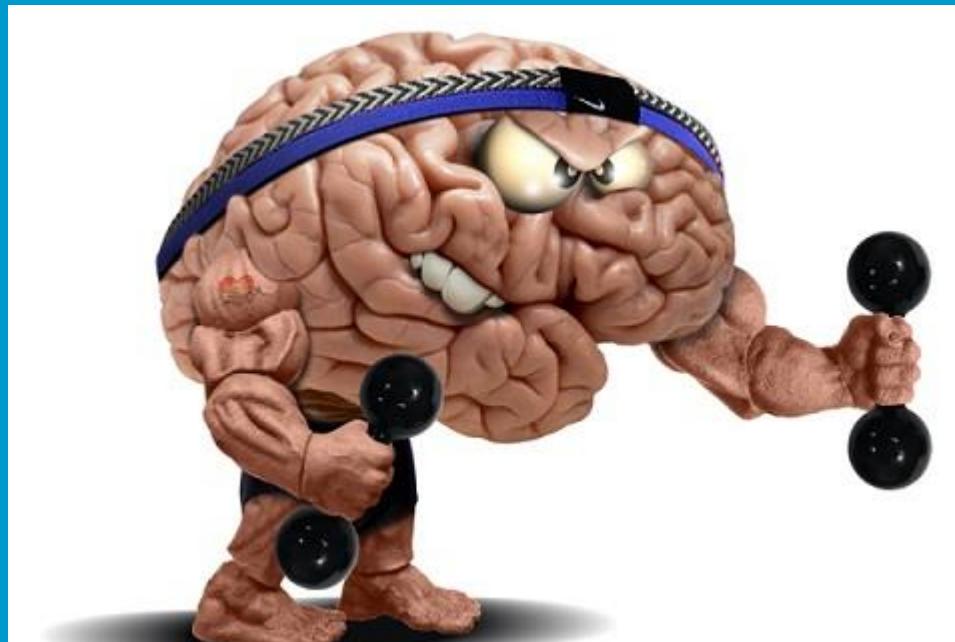
# Perceptron de limiar

- O perceptron de limiar é chamado **separador linear**
  - Porque traça um plano entre os pontos de entrada onde a saída é zero ou um



Funções linearmente separáveis (a) e (b)

# Treinamento do neurônio



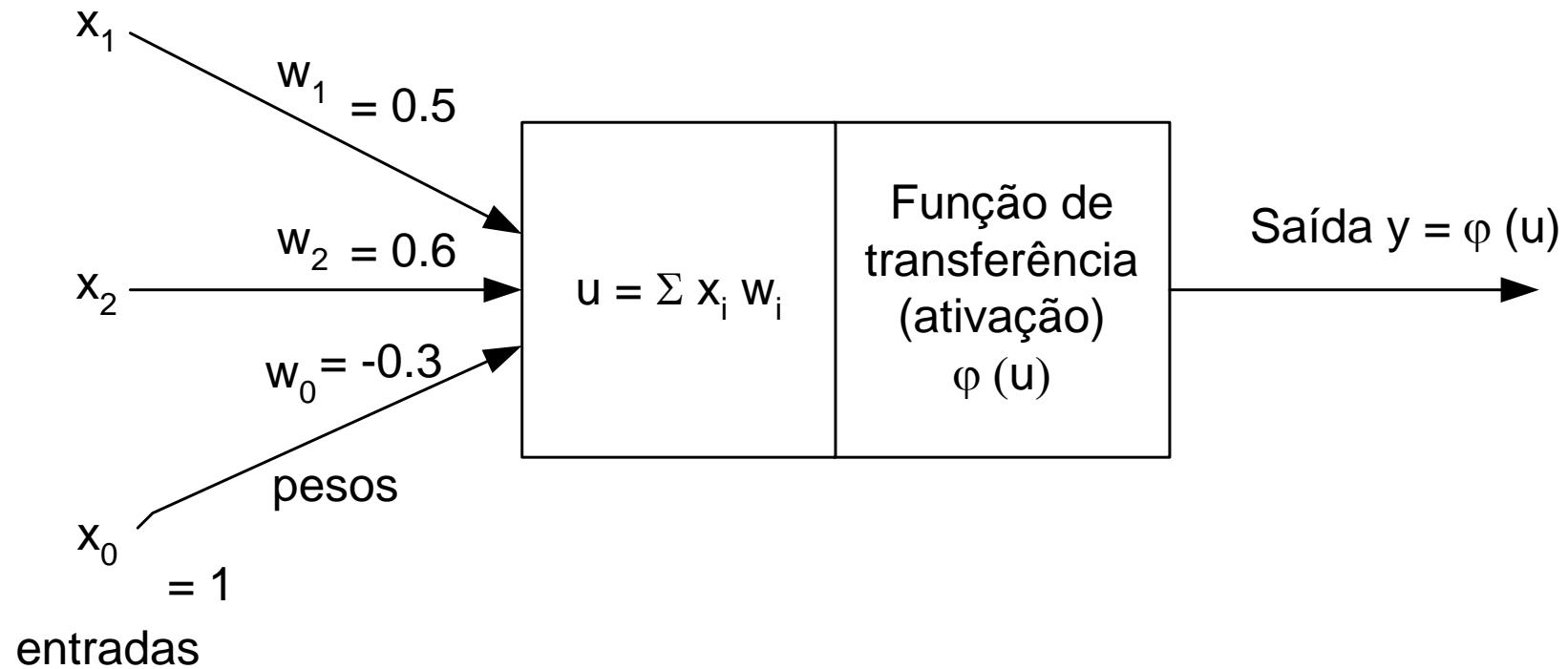
# Exemplo: um neurônio para a função AND de 2 entradas

Iniciamos o neurônio com os pesos 0.5 e 0.6 para as duas entradas, e -0.3 para o limiar ( $w_0$ ). Isso é equivalente à equação:

$$u = 0.5 \ x_1 + 0.6 \ x_2 - 0.3 \ x_0$$

onde  $x_0$  é a entrada estável sempre igual a 1.

Assim, a saída  $y$  deve disparar quando  $u \geq 0$ .



$x_1$	$x_2$	$u$	$y$
0	0	-0.3	0
0	1	0.3	1
1	0	0.2	1
1	1	0.8	1

a saída é 1 para os padrões 01, 10 e 11, enquanto desejamos que a saída seja 1 somente para o padrão 11.

# Seqüência de passos na aplicação do algoritmo

**Inicio**

Entrada 0 0     $u = -0.3$      $y = 0$  correta

Entrada 0 1     $u = 0.3$      $y = 1$  incorreta

**Correção dos pesos de 0.1 para baixo →**

Entrada 1 0     $u = 0.1$      $y = 1$  incorreta

**Correção dos pesos de 0.1 para baixo →**

Entrada 1 1     $u = 0.4$      $y = 1$  correta

Entrada 0 0     $u = -0.5$      $y = 0$  correta

Entrada 0 1     $u = 0$      $y = 1$  incorreta

**Correção dos pesos de 0.1 para baixo →**

Entrada 1 0     $u = -0.2$      $y = 0$  correta

Entrada 1 1     $u = 0.2$      $y = 1$  correta

Entrada 0 0     $u = -0.6$      $y = 0$  correta

Entrada 0 1     $u = -0.2$      $y = 0$  correta

Entrada 1 0     $u = -0.2$      $y = 0$  correta

Entrada 1 1     $u = 0.2$      $y = 1$  correta

**Fim**

$w_1 = 0.5$   $w_2 = 0.6$   $w_0 = -0.3$

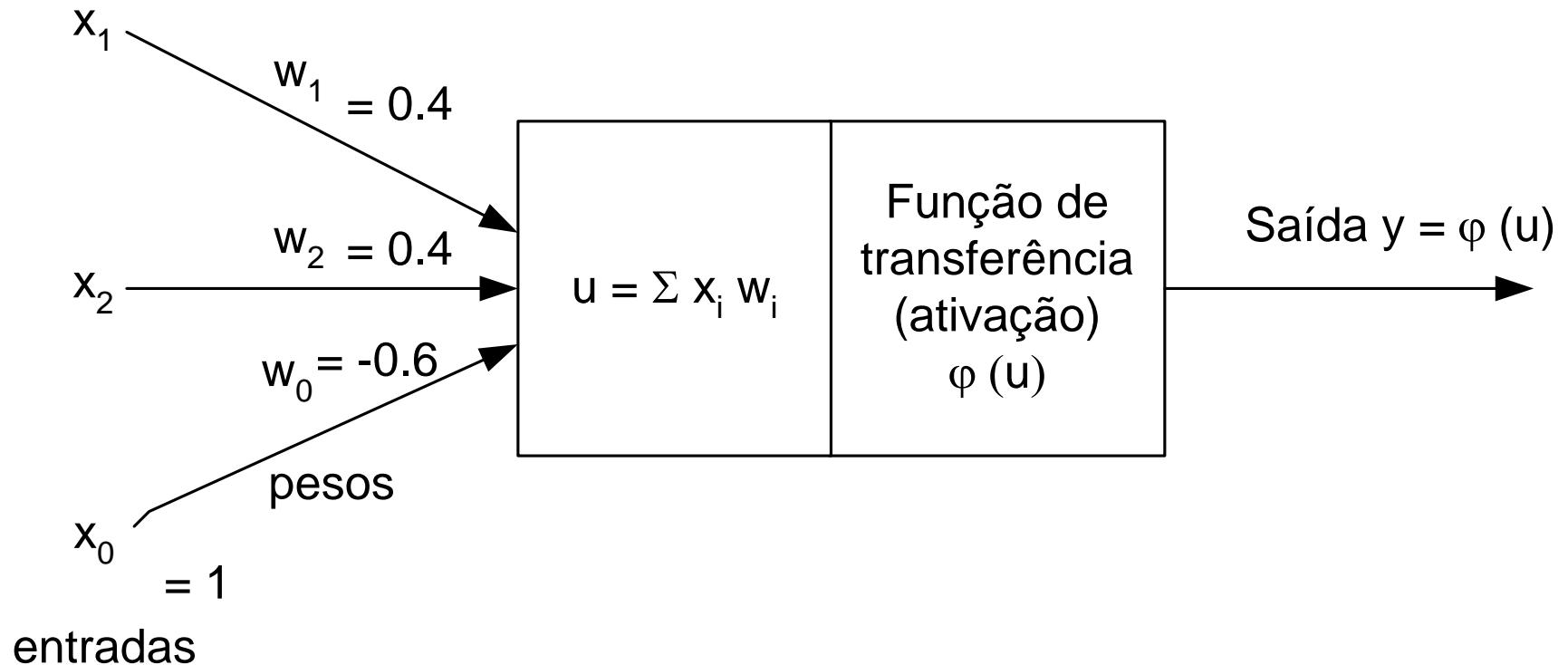
$w_1 = 0.5$   $w_2 = 0.5$   $w_0 = -0.4$

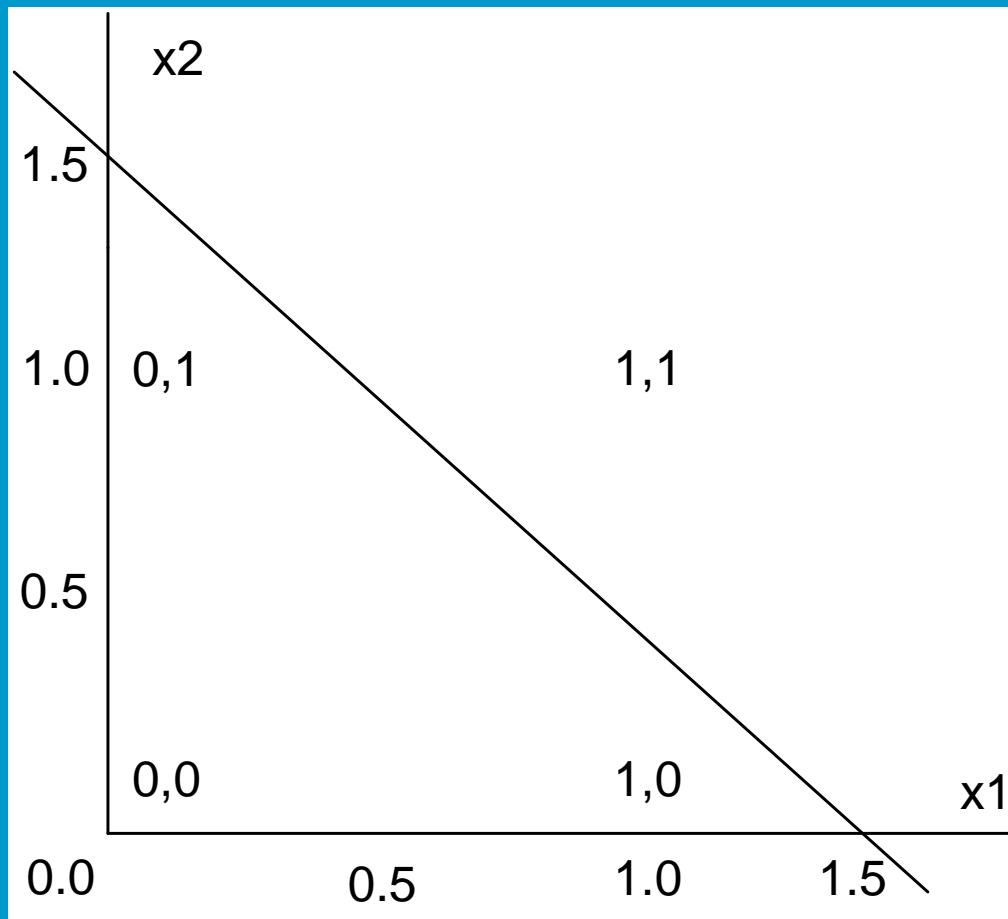
$w_1 = 0.4$   $w_2 = 0.5$   $w_0 = -0.5$

$w_1 = 0.4$   $w_2 = 0.4$   $w_0 = -0.6$

$w_1 = 0.4$   $w_2 = 0.4$   $w_0 = -0.6$

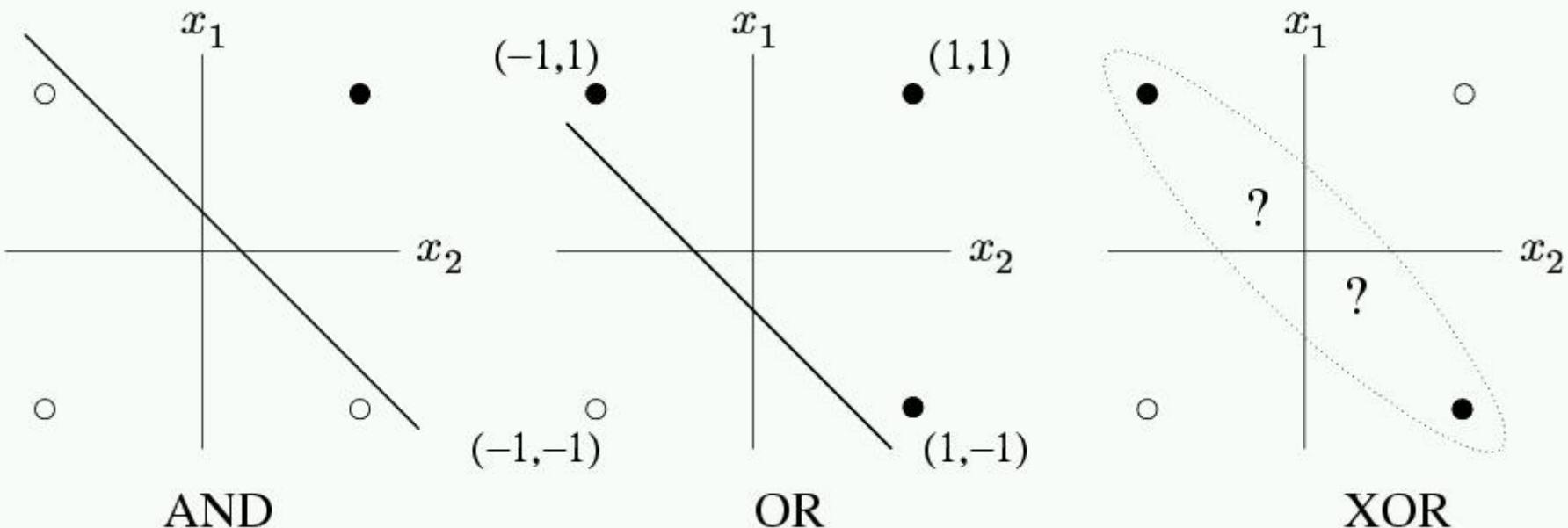
# Resultado do aprendizado





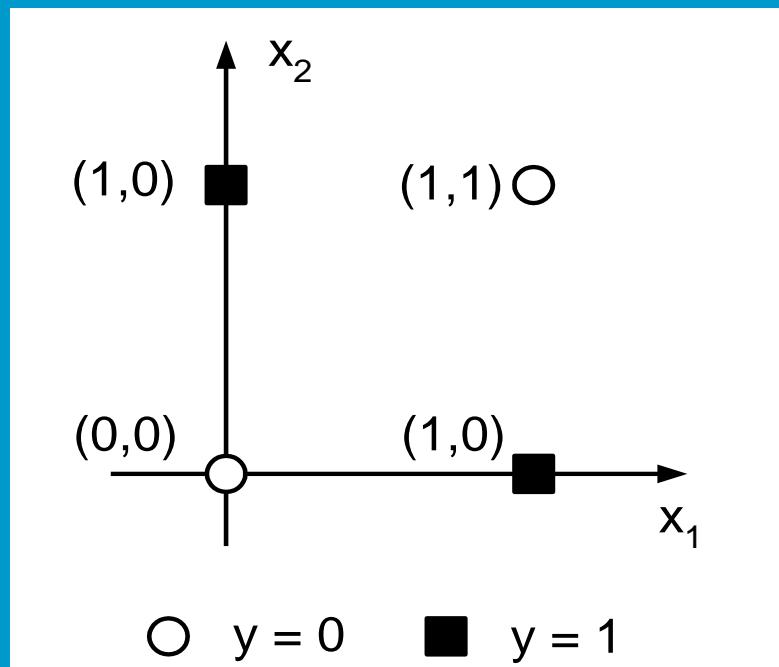
A reta  $0.4x_1 + 0.4x_2 - 0.6 = 0$  separa os pontos  $00$ ,  $01$  e  $10$ , do ponto  $11$ .

# Exemplo: um neurônio para a função XOR



No caso do XOR, não existe uma única reta que divide os pontos  $(0,0)$  e  $(1,1)$  para um lado, e  $(0,1)$  e  $(1,0)$  do outro lado.

Função xor:



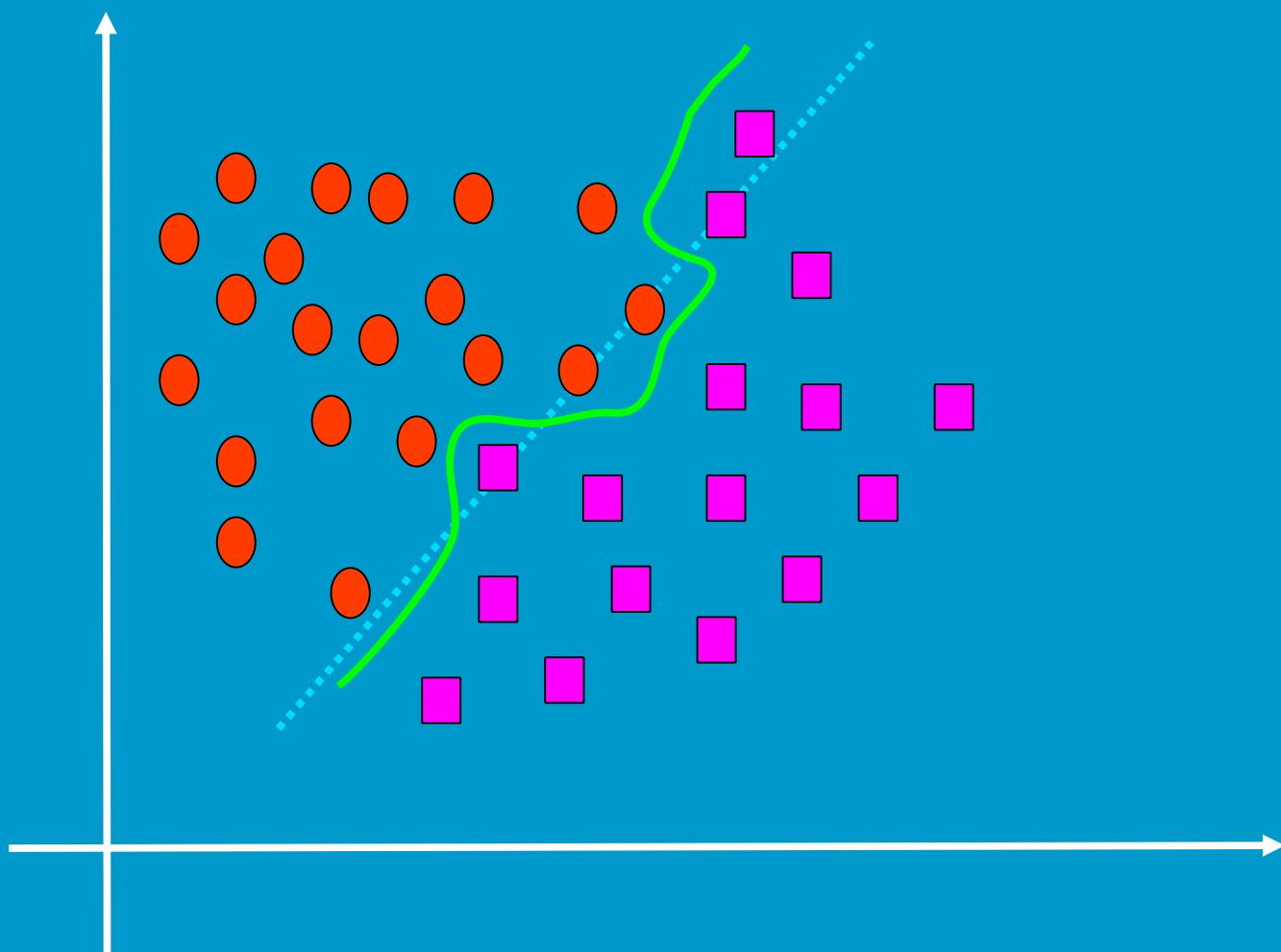
Conclui-se que um neurônio do tipo Perceptron não implementa uma função ou-exclusivo (constatado por Minsky & Papert, em 1969).

# Algoritmo de treinamento do Perceptron

Para classificação padrões de entrada como pertencentes ou não a uma dada classe, considere o conjunto de treinamento formado por N amostras  $\{\mathbf{x}_1, d_1\}, \{\mathbf{x}_2, d_2\}, \dots, \{\mathbf{x}_N, d_N\}$ , onde  $\mathbf{x}_j$  é o vetor de entradas e  $d_j$  a saída desejada (classe), que em notação vetorial tem-se  $\{\mathbf{X}, \mathbf{d}\}$ , onde:

$$\mathbf{X} \in \Re^{m \times N}$$

$$\mathbf{d} \in \Re^{1 \times N}$$



# Aprendizagem de Perceptrons

## ■ O algoritmo

- Executa os exemplos de treinamento através da rede;
- Ajusta os pesos depois de cada exemplo para reduzir o erro;
- Cada ciclo através dos exemplos é chamado de **época**;
- As épocas são repetidas até que se alcance algum critério de parada.
  - Em geral, quando as mudanças nos pesos forem pequenas.

**Função [w] = perceptron (max\_it, E, α, X,d)**

**inicializar w // para simplicidade, com zeros**

**inicializar b // para simplicidade, com zero**

**t  $\leftarrow$  1**

**while t < max\_it & E > 0 do**

**for i from 1 to N do**

**$y_i \leftarrow f(w x_i + b)$**

**$e_i \leftarrow d_i - y_i$**

**$w \leftarrow w + \alpha e_i x_i$**

**$b \leftarrow b + \alpha e_i$**

**end for**

**$E \leftarrow \text{sum } (e_i)$**

**$t \leftarrow t + 1$**

**end while**

**end procedure**

**// para cada padrão de entrada**

**// determinar a saída**

**// determinar o erro**

**// atualizar o vetor peso**

**// atualizar o bias**

**//quantidade de erros**

# Adaline

- Na mesma época em que Rosenblatt propôs o Perceptron, Widrow e Hoff propuseram o algoritmo dos mínimos quadrados (regra delta) para a rede Adaline (Adaptive Linear Element), similar ao Perceptron, porém com função de ativação linear ao invés de função degrau.
- **O objetivo do algoritmo de treinamento é minimizar o erro quadrático médio (MSE) entre a saída de rede e a saída desejada.**

- A soma dos erros quadráticos para um determinado padrão é dada por:

$$E = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (d_i - y_i)^2$$

- O gradiente de E, também denominado de índice de desempenho ou função custo, fornece a direção de crescimento mais rápido de E.
- Portanto, a direção oposta ao gradiente de E é a direção de maior decrescimento.

# Adaline (cont.)

O erro pode ser reduzido ajustando-se os pesos da rede de acordo com:

$$w_{IJ} = w_{IJ} - \alpha \frac{\partial E}{\partial w_{IJ}}$$

onde  $w_{IJ}$  é o peso específico para o neurônio pós-sináptico  $I$ , da entrada  $J$ , e  $\alpha$  é a taxa de aprendizagem.

Como  $w_{IJ}$  influencia apenas o neurônio I,

$$\frac{\partial E}{\partial w_{IJ}} = \frac{\partial}{\partial w_{IJ}} \sum_{i=1}^n (d_i - y_i)^2 = \frac{\partial}{\partial w_{IJ}} (d_I - y_I)^2$$

Como

$$y_I = f(\mathbf{w}_I \cdot \mathbf{x}) = f\left(\sum_j w_{Ij} x_j\right) = \sum_j w_{Ij} x_j$$

$$\frac{\partial E}{\partial w_{IJ}} = -2(d_I - y_I) \frac{\partial y_I}{\partial w_{IJ}} = -2(d_I - y_I) x_J$$

# Regra delta

Portanto a regra delta para o Adaline resume-se em:

$$w_{IJ} = w_{IJ} + \alpha(d_I - y_I)x_J$$
$$b_I = b_I + \alpha(d_I - y_I)$$

Em notação vetorial tem-se:

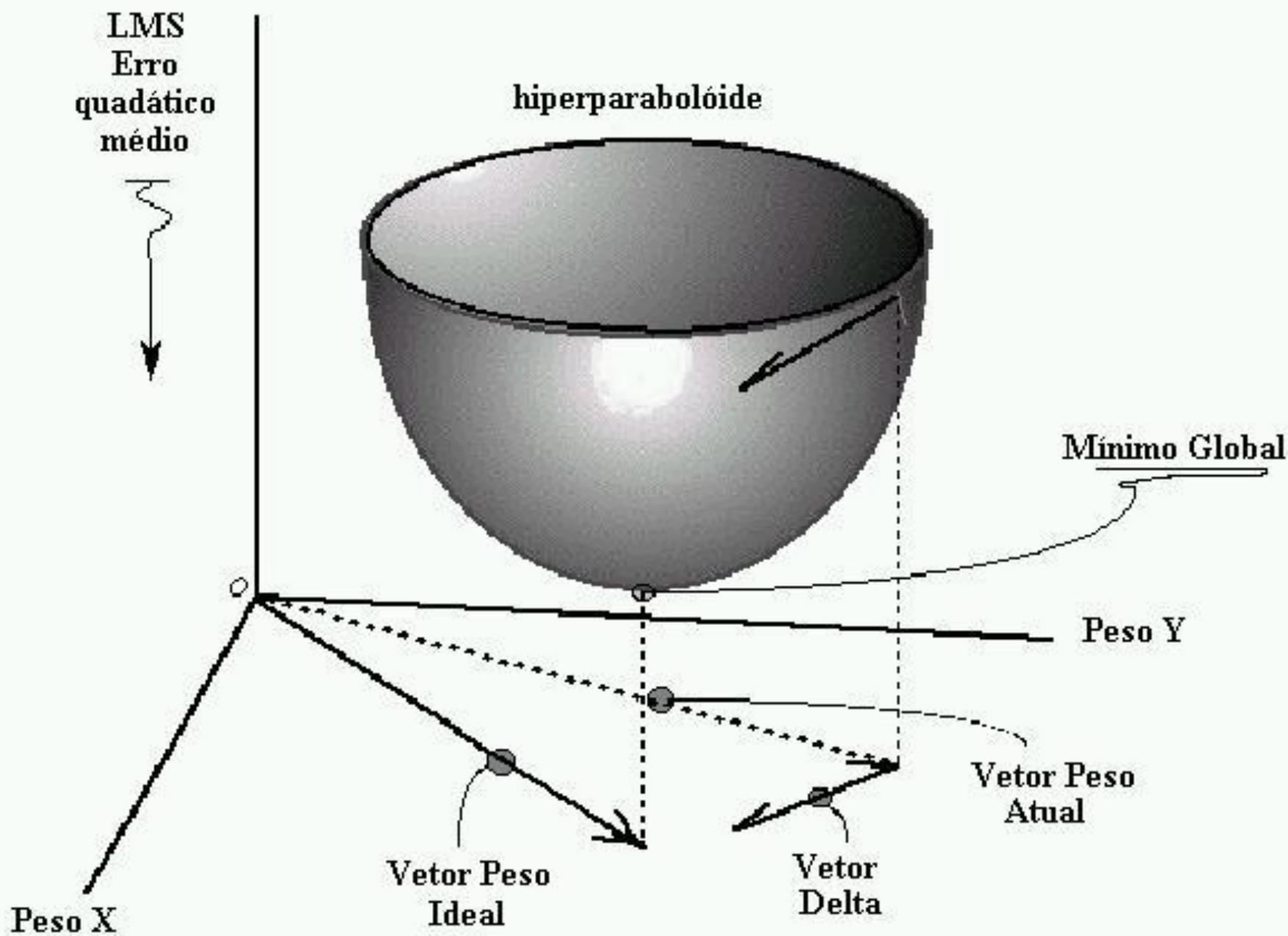
$$\mathbf{W} = \mathbf{W} + \alpha \mathbf{e}_i \mathbf{x}_i^T$$

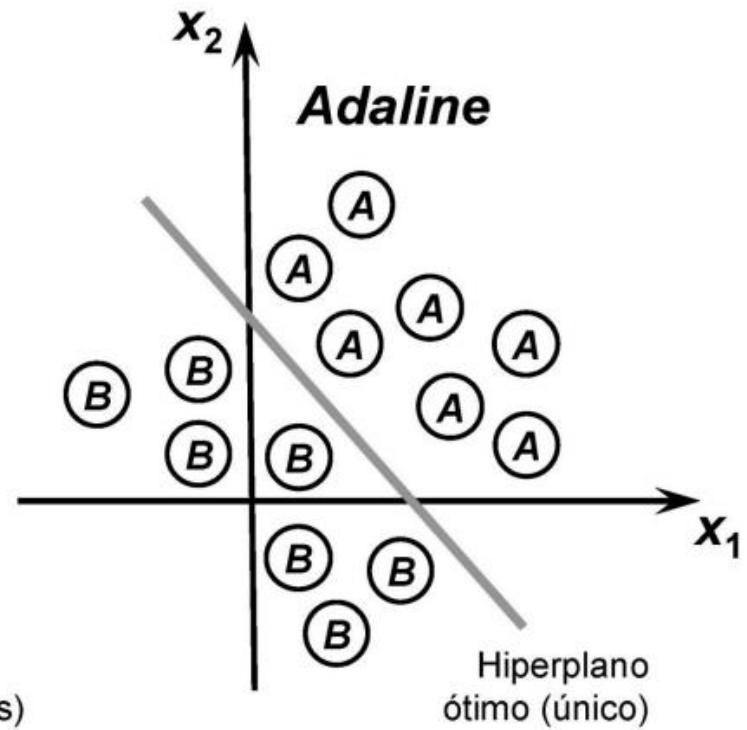
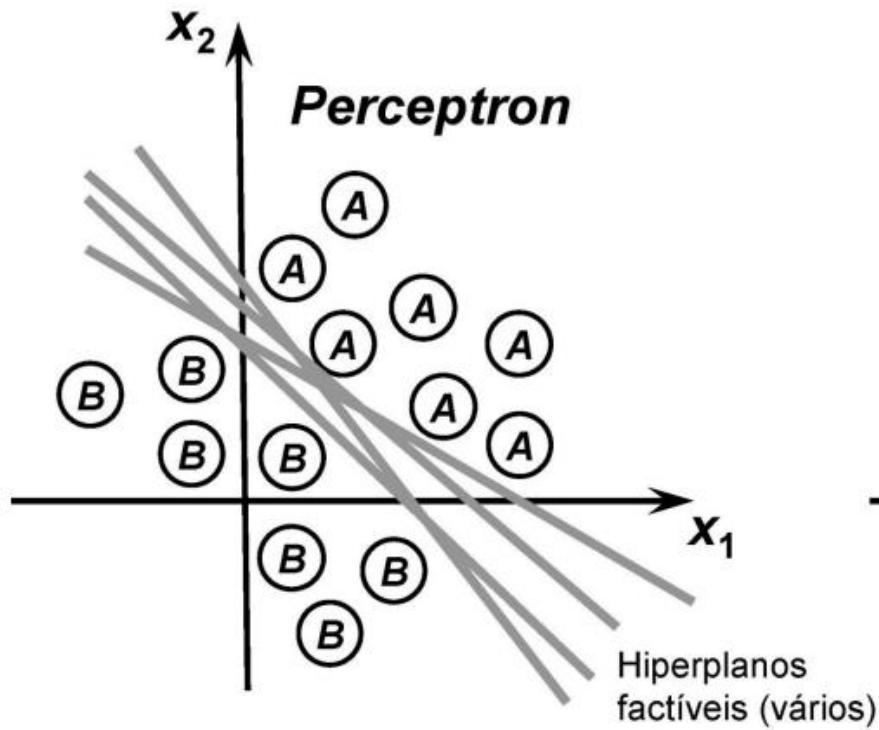
$$\mathbf{b} = \mathbf{b} + \alpha \mathbf{e}_i$$

onde :

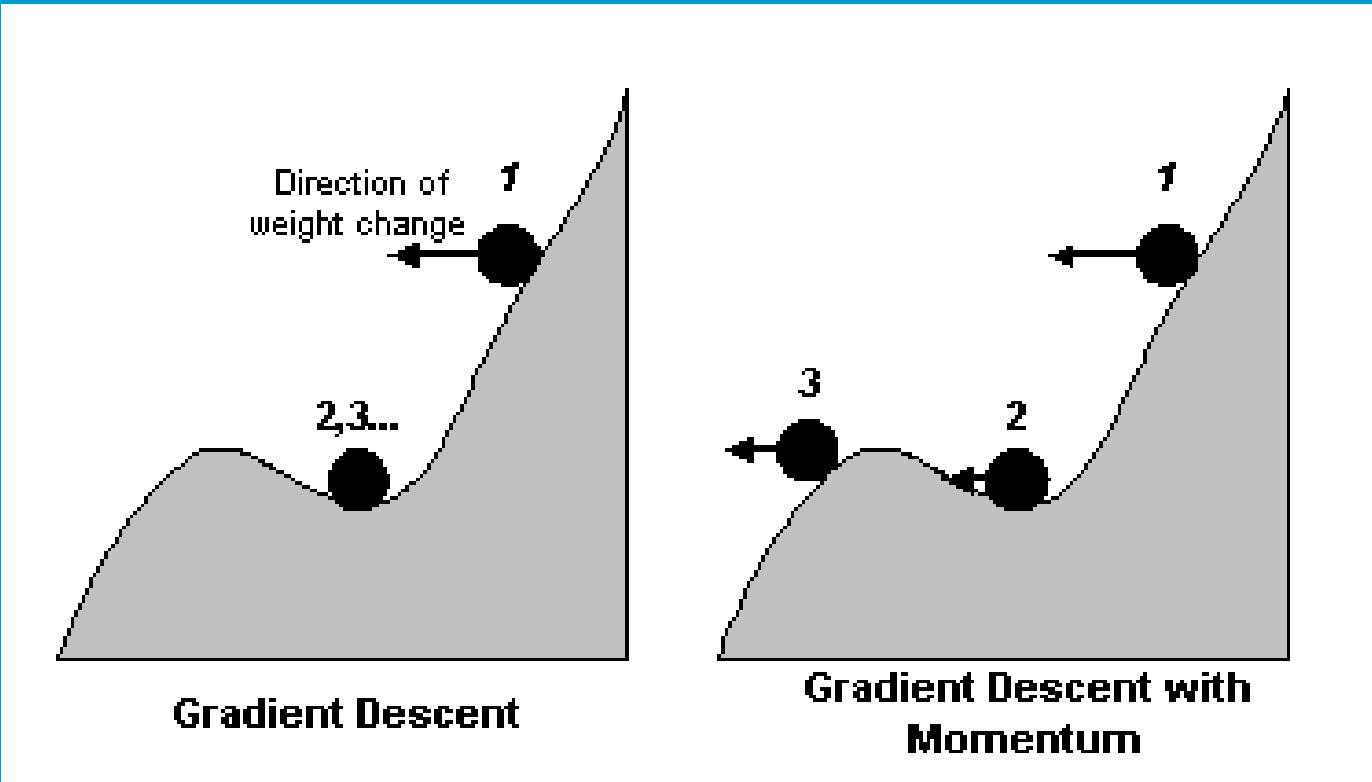
$$\mathbf{W} \in \mathcal{R}^{oxm}, \mathbf{x}_i \in \mathcal{R}^{mx1}, i = 1, \dots, N,$$

$$\mathbf{e}_i \in \mathcal{R}^{ox1}, \text{ e } \mathbf{b} \in \mathcal{R}^{ox1}$$





# Aprendizagem com Momento



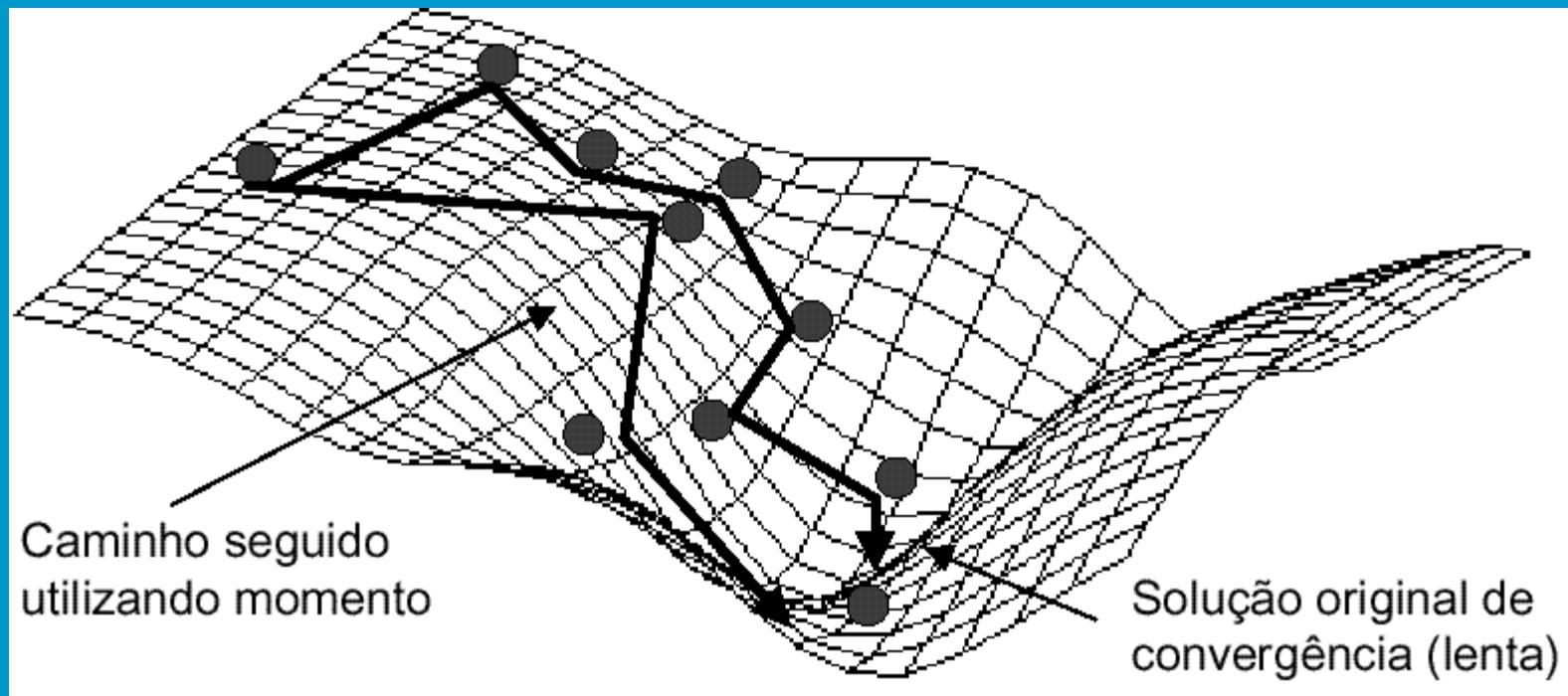
# Aprendizagem com Momento

- *Aprendizagem com momento* usa uma memória (incremento anterior) para aumentar a velocidade e estabilizar a convergência
- Equação de correção dos pesos:

$$w_{ij}(n+1) = w_{ij}(n) + \alpha e_i(n)x_j(n) + \beta [w_{ij}(n) - w_{ij}(n-1)]$$

onde  $\beta$  é a constante de momento

## ■ Momentum



- Próxima aula:
  - Como treinar uma rede neural com vários neurônios (Fortes emoções).



# Fim