IA- Redes Neurias

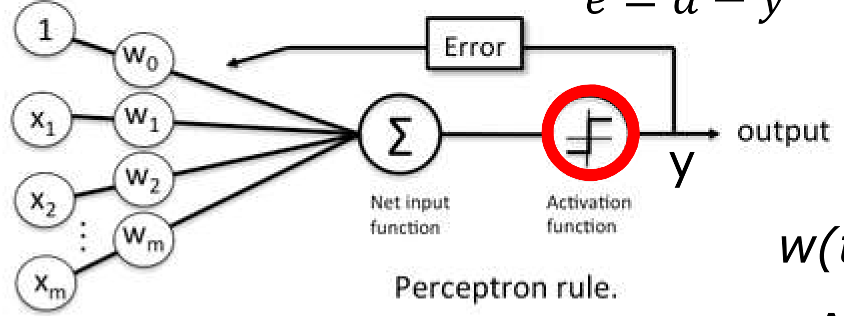
Rede Multi-Layer Perceptron (MLP)

Prof. Myriam Delgado

RNAs: Arquitetura (revisão)

Perceptron e Adaline

$$e = d - y$$

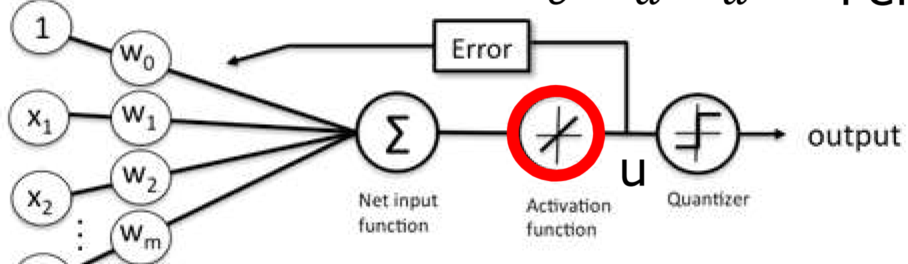


Perceptron rule.

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

Ambos treinados
Pela regra delta

$$e = d - u$$



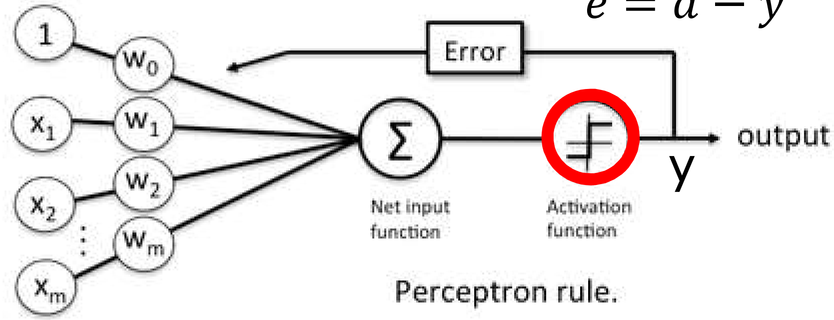
Adaline.
Delta Rule

RNAs: Arquitetura (revisão)

Perceptron e Adaline

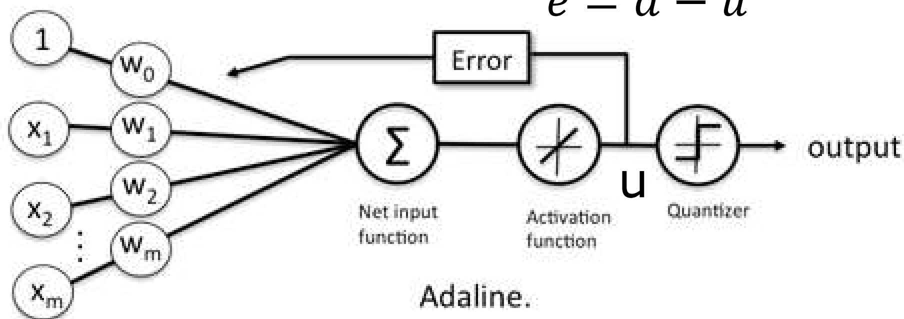
Y∈ {0,1}
Aplicação em Problemas de Classificação Linearmente Separáveis

$$e = d - y$$



Perceptron rule.

$$e = d - u$$

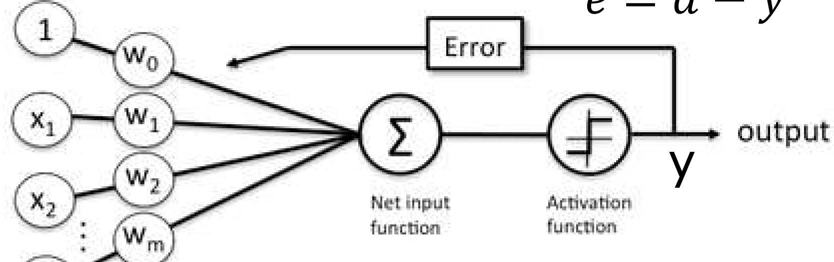


Adaline.
Delta Rule

RNAs: Arquitetura (revisão)

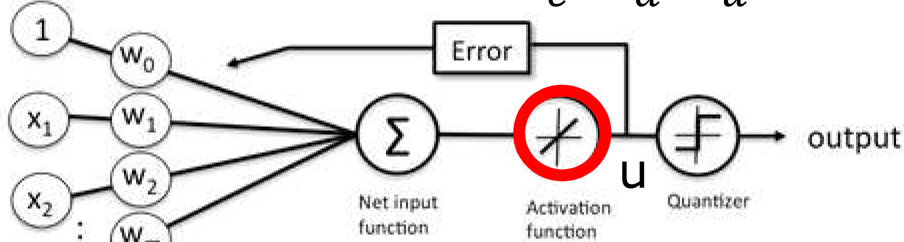
Perceptron e Adaline

$$e = d - y$$



Perceptron rule.

$$e = d - u$$

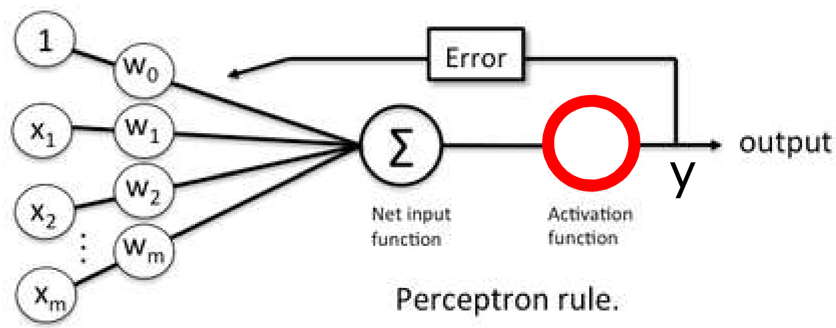


Adaline.
Delta Rule

$Y \in \{-\propto, +\propto\}$
Aplicação em
Aproximação
De Funções
(combinação
Linear de
Funções não
Lineares)

RNAs: Arquitetura (revisão)

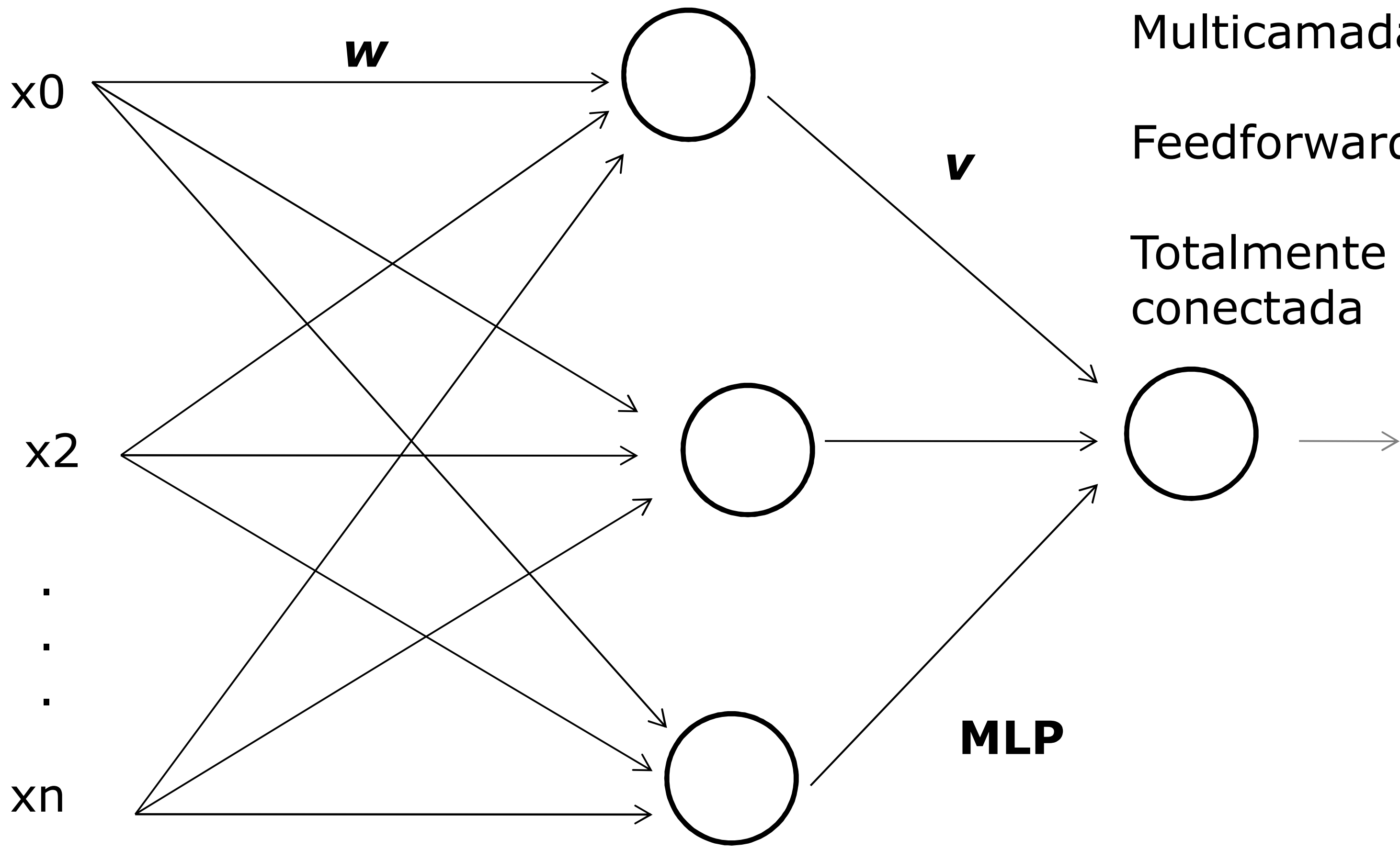
Perceptron' e Adaline



Perceptron rule.

Perceptron': MLP de camada única
(aceita diferentes funções de ativação)

Multi-Layer Perceptron



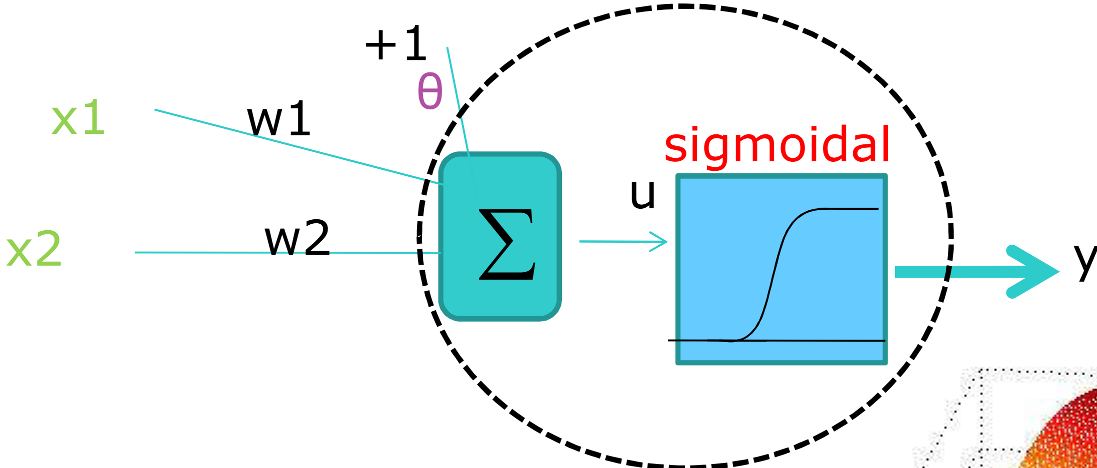
Rede Multicamadas

Feedforward

Totalmente conectada

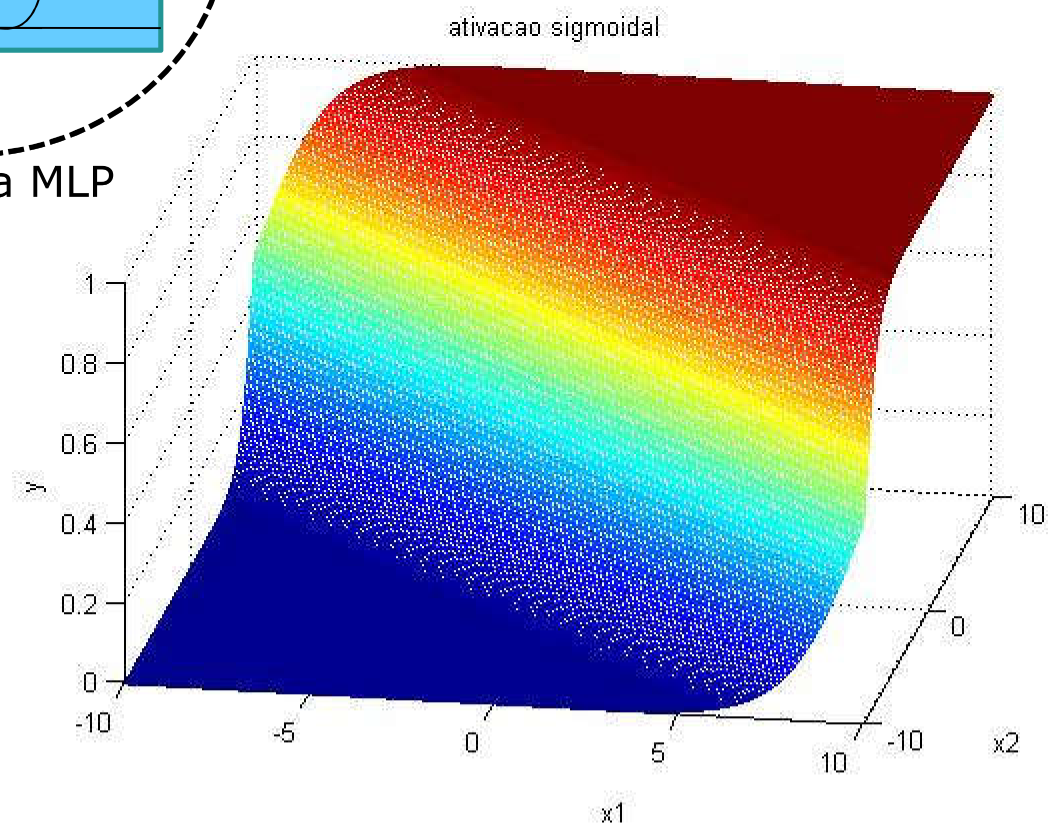
MLP

Neurônio na MLP: função de ativação Sigmoide



Neurônio na MLP

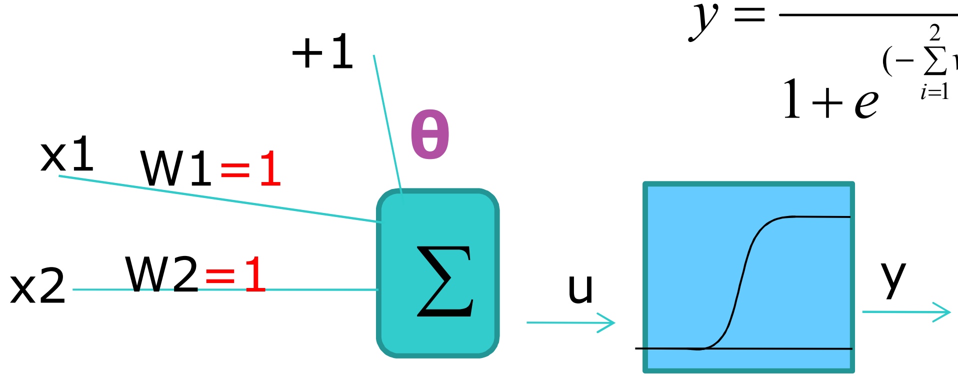
Função Ativação no espaço



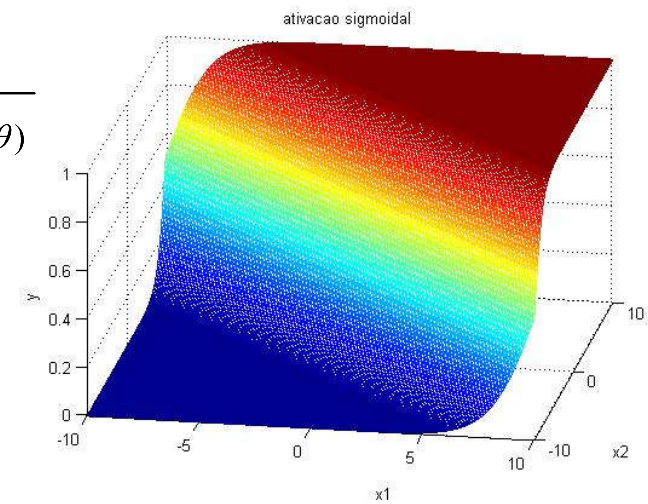
$$y = \frac{1}{1 + e^{(-u)}}$$

onde $u = \left(\sum_{i=1}^{2} w_i x_i + \theta \right)$

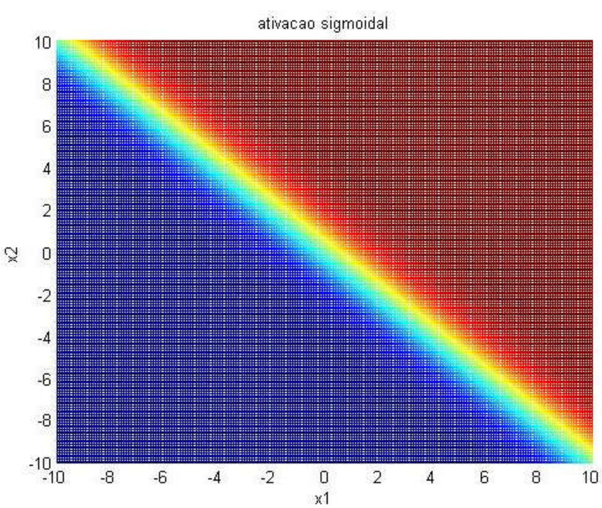
Neurônio na MLP: partição *x1* x *x2*



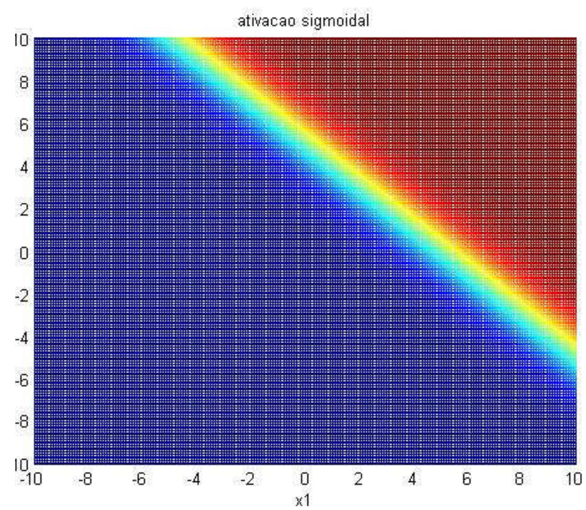
$$y = \frac{1}{1 + e^{(-\sum_{i=1}^{2} w_i x_i + \theta)}}$$



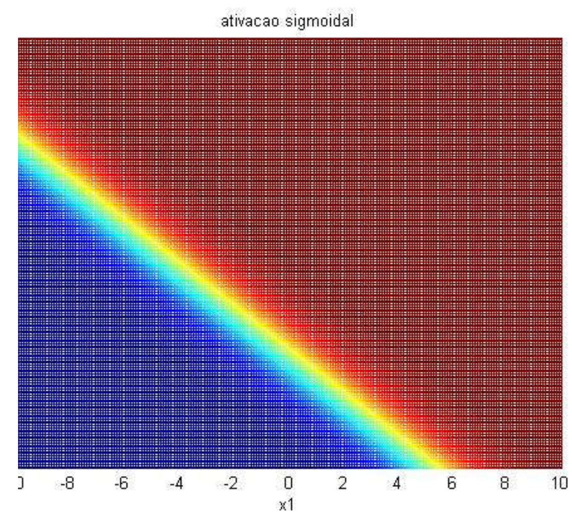
Θ =0



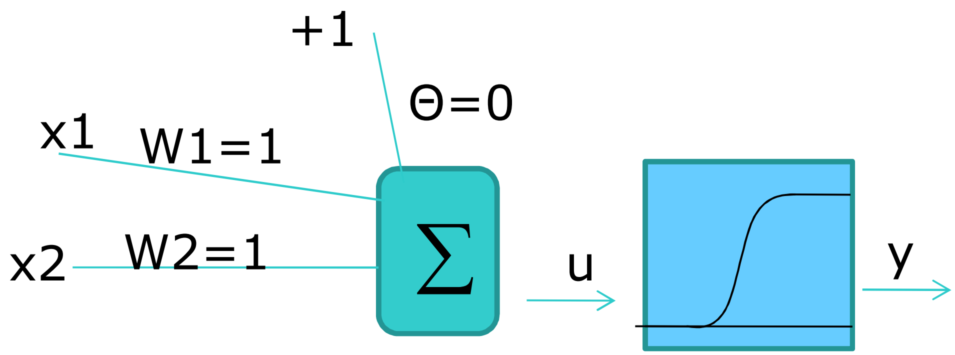
Θ =-5



Θ =+5

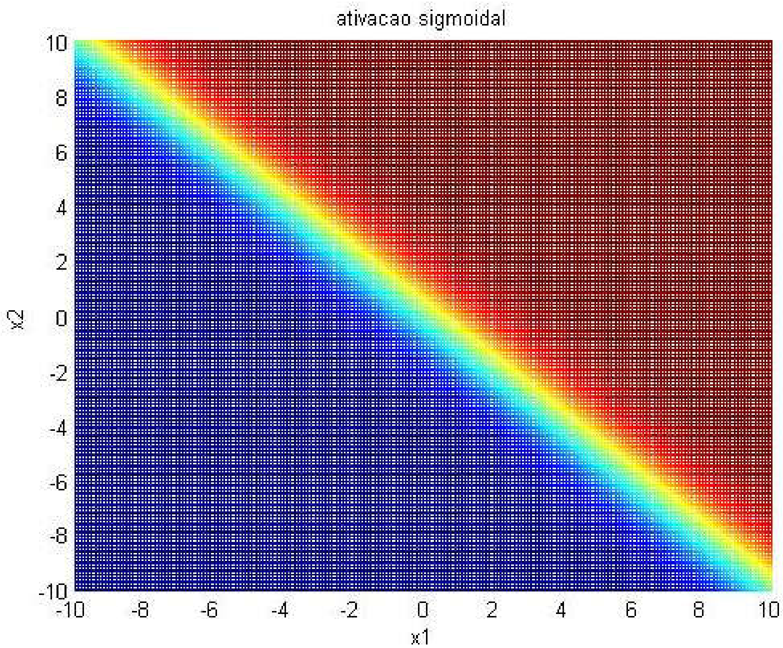


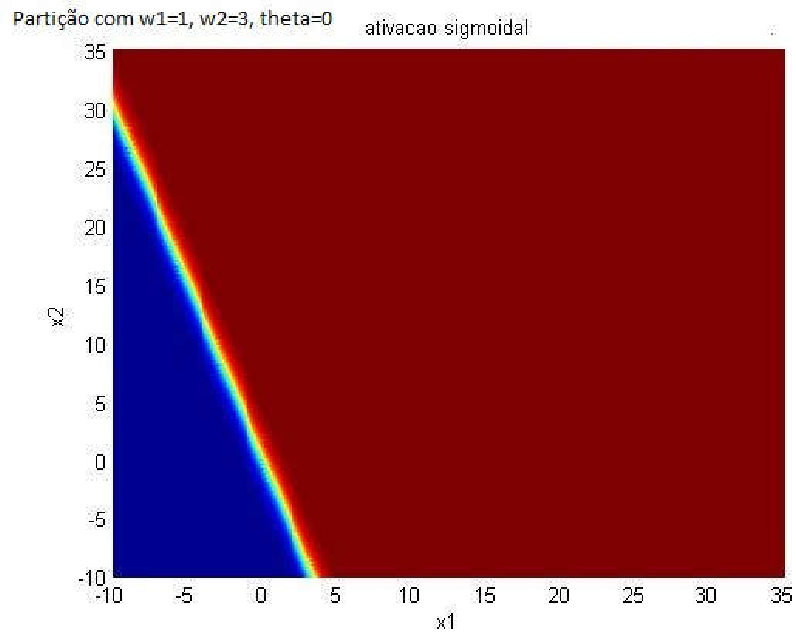
Neurônio na MLP: Abstração na partição



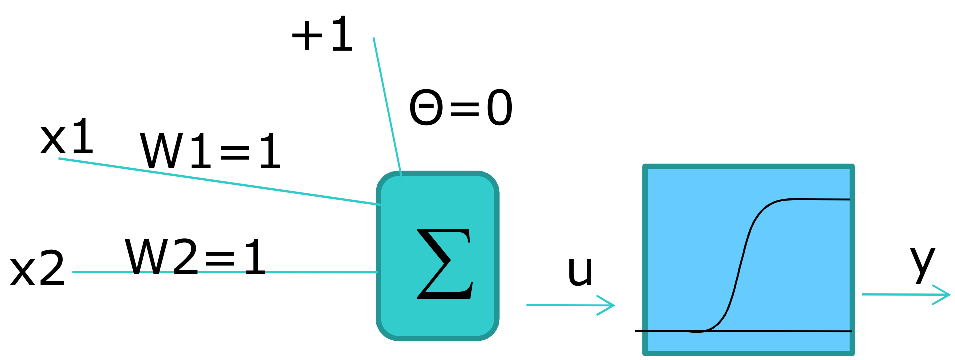
O que acontece se alterarmos os pesos w1, w2?

Θ=0, w1=1, w2=3



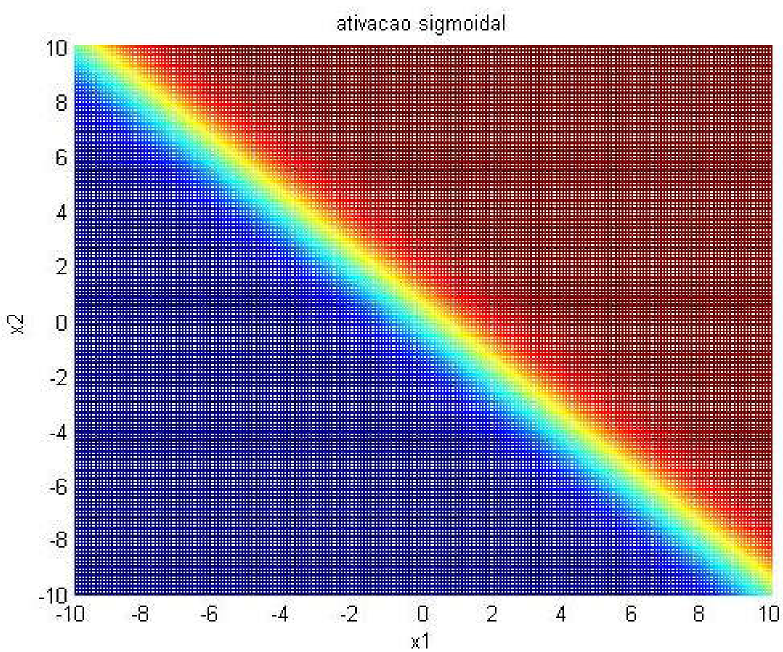


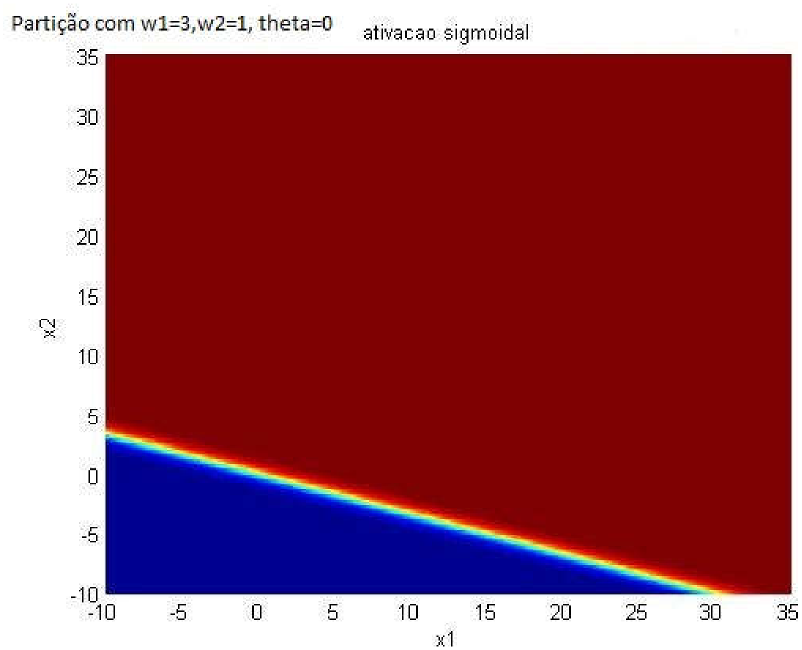
Neurônio na MLP: Abstração na partição



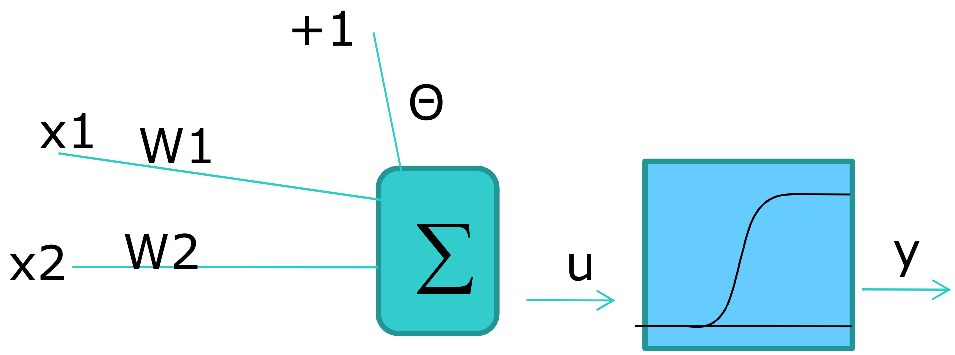
O que acontece se alterarmos os pesos w1, w2?

$\Theta=0$, $W1=3$, $w2=1$



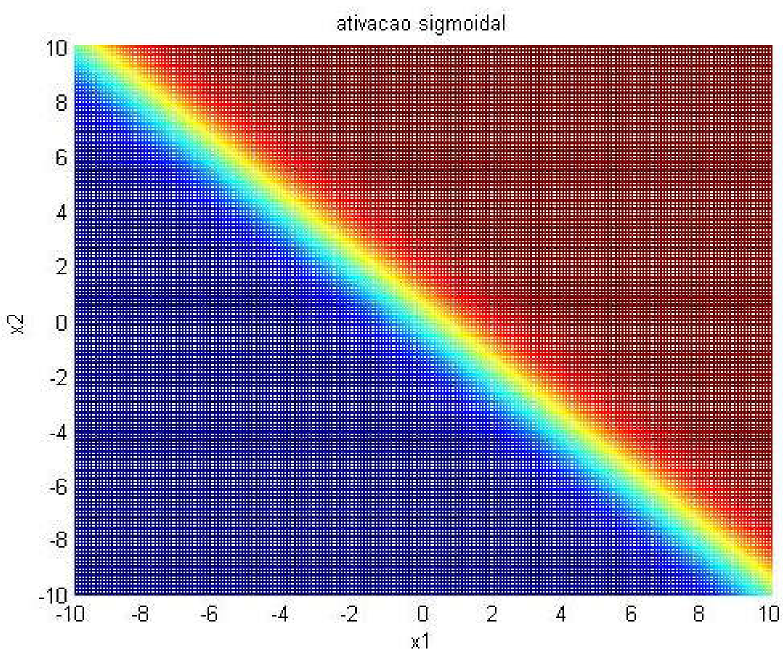


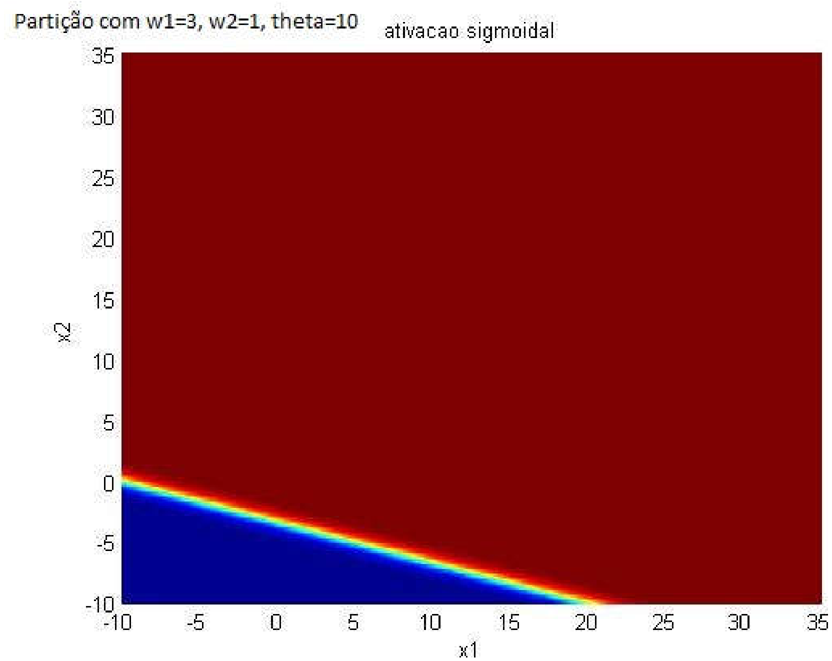
Neurônio na MLP: Abstração na partição



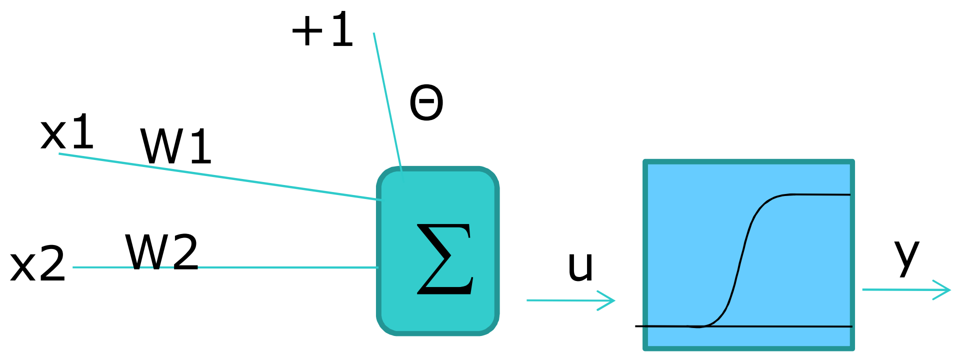
O que acontece se alterarmos o limiar Θ, e os Pesos w1, w2?

Θ=10, W1=3, w2=1



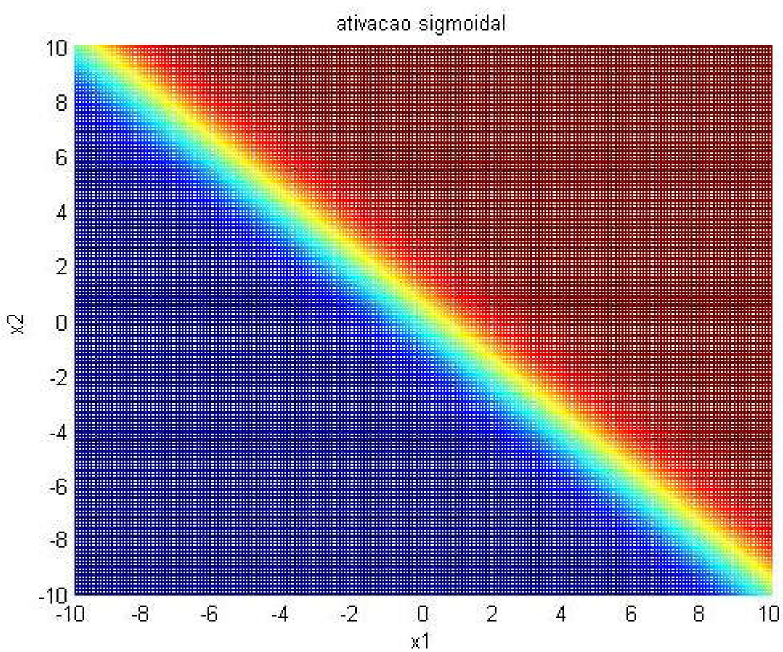


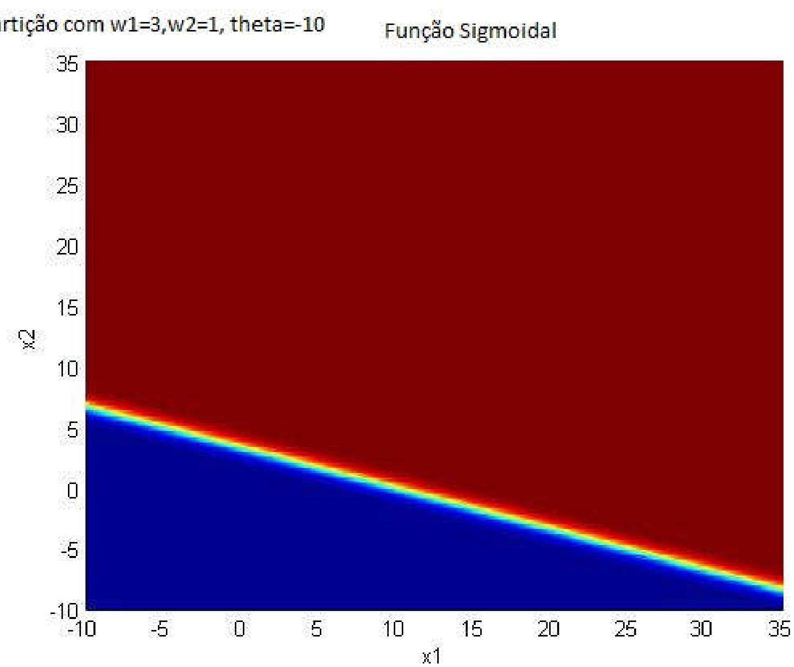
Neurônio na MLP: Abstração na partição



O que acontece se alterarmos o limiar Θ, e os Pesos w1, w2?

Θ=-10, W1=3, w2=1

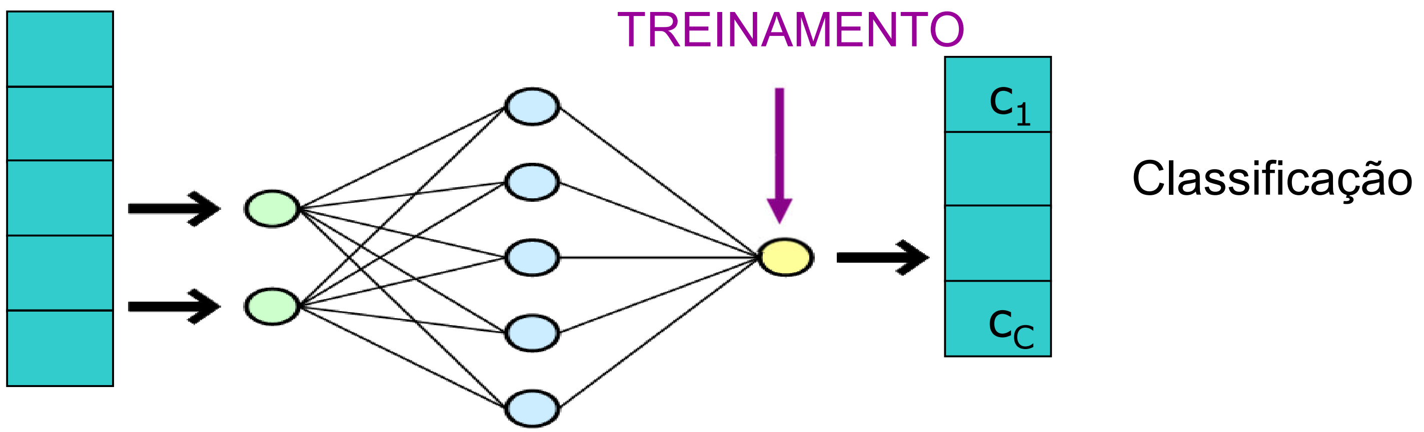




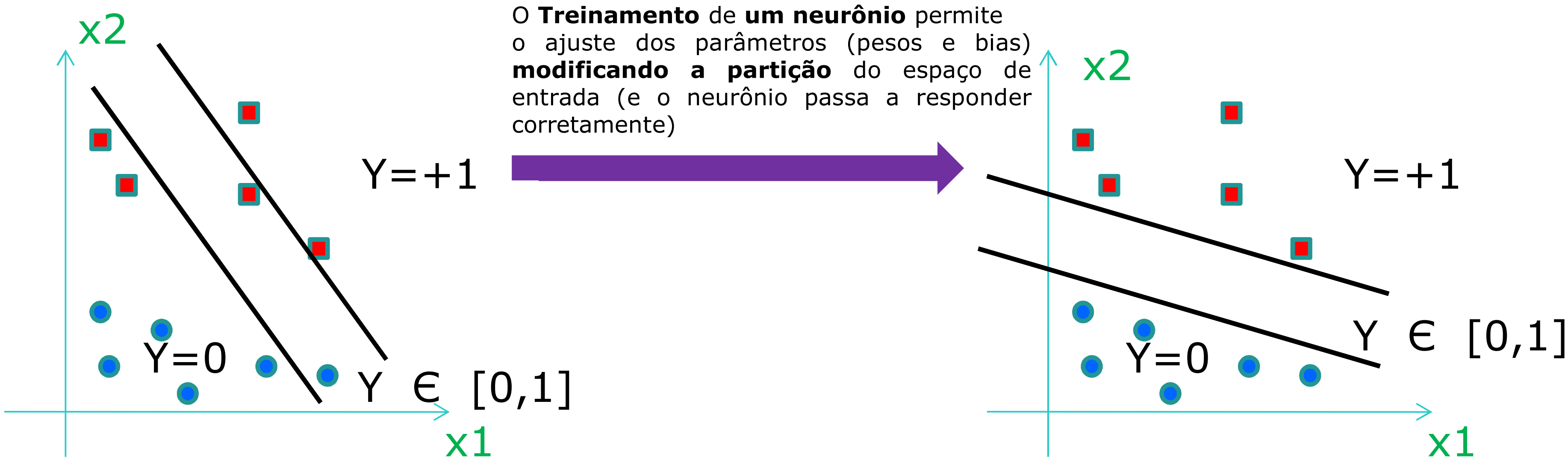
Multi-Layer Perceptron

- Arquitetura da rede
- Modelo do neurônio: função de ativação
- **Treinamento**
- **Aplicação**

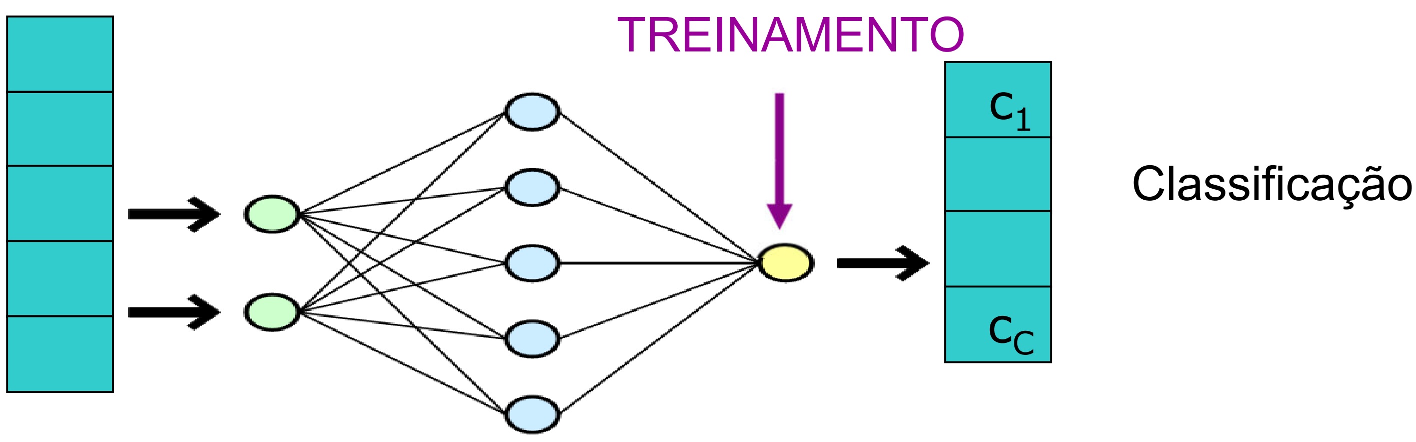
MLP: Treinamento e Aplicação

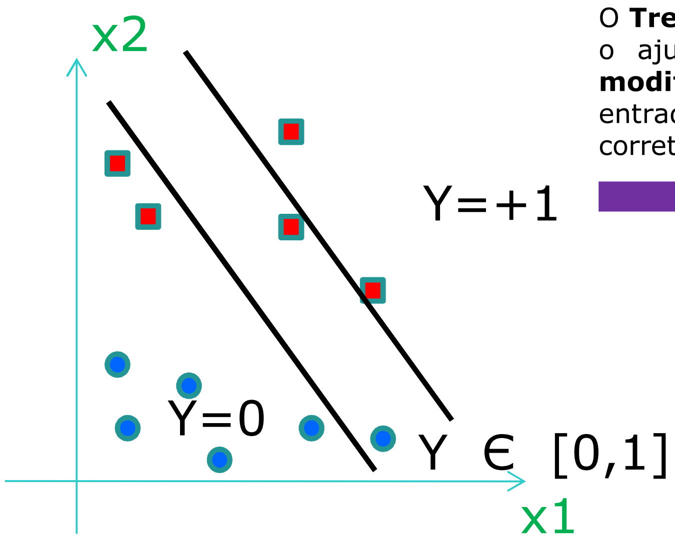


O **Treinamento** de **um neurônio** permite o ajuste dos parâmetros (pesos e bias) **modificando a partição** do espaço de entrada (e o neurônio passa a responder corretamente)

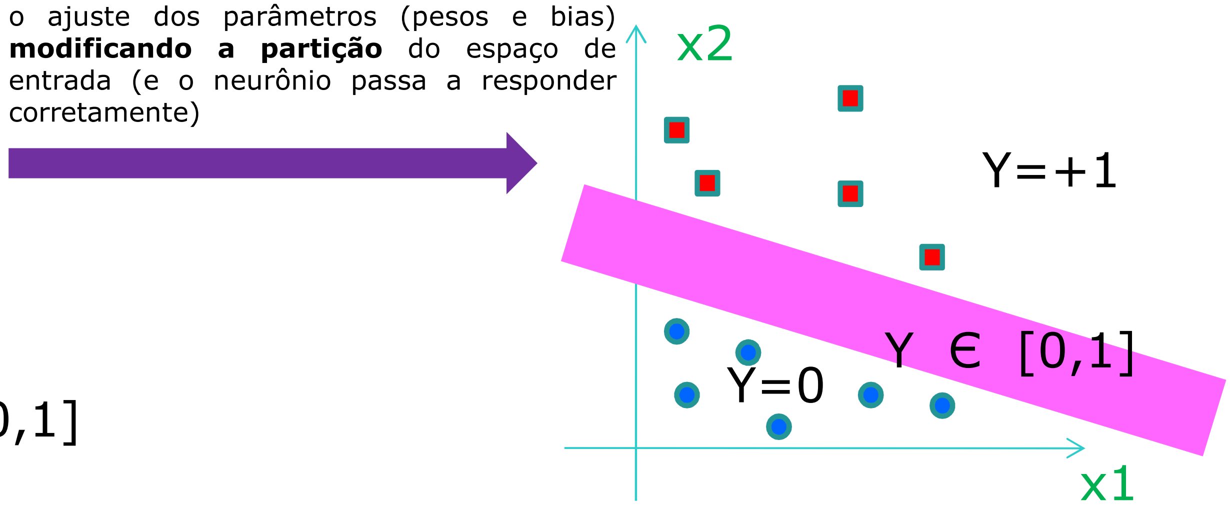


MLP: Treinamento e Aplicação

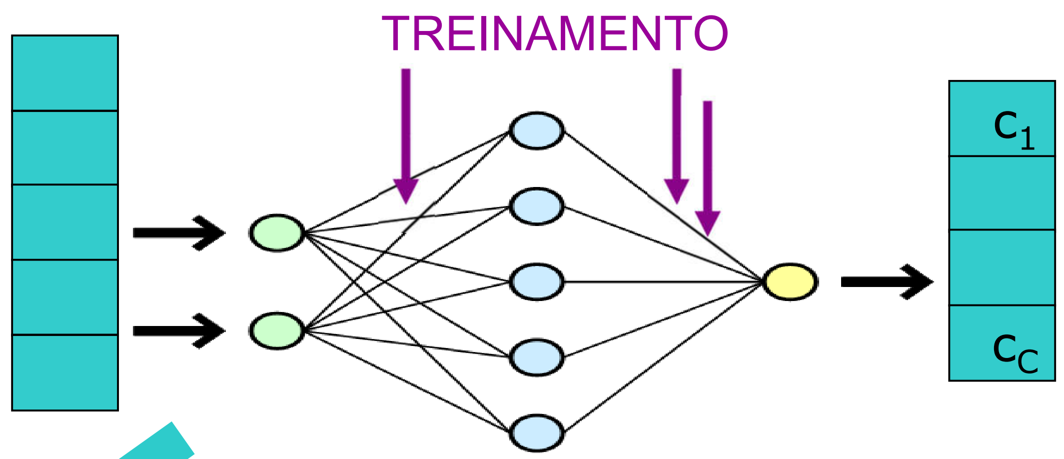




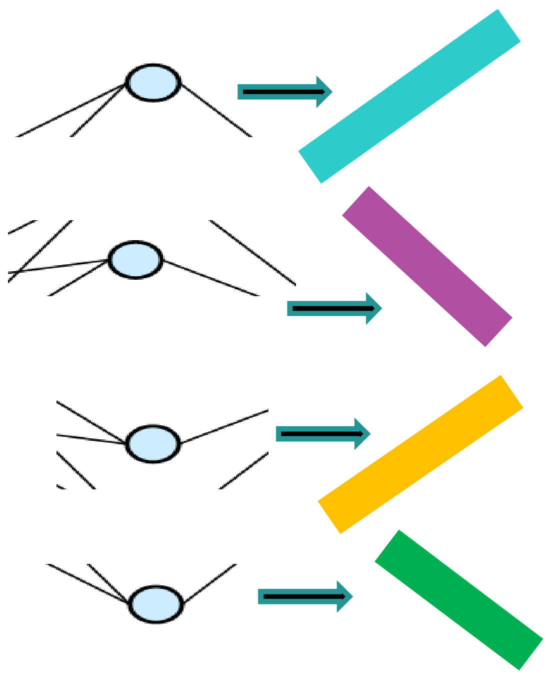
O **Treinamento** de **um neurônio** permite o ajuste dos parâmetros (pesos e bias) **modificando a partição** do espaço de entrada (e o neurônio passa a responder corretamente)



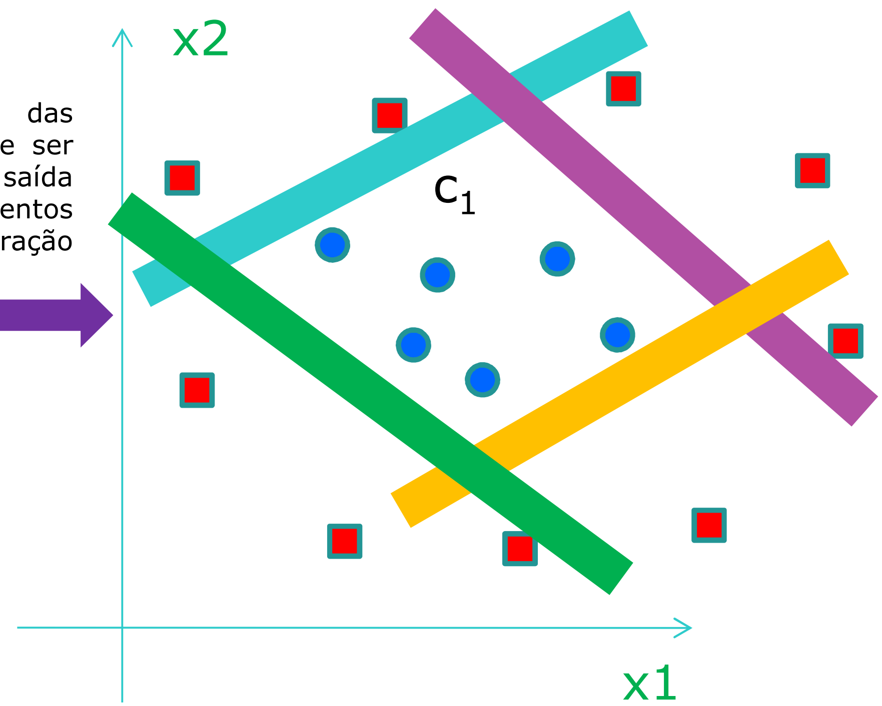
MLP: Treinamento e Aplicação



Classificação

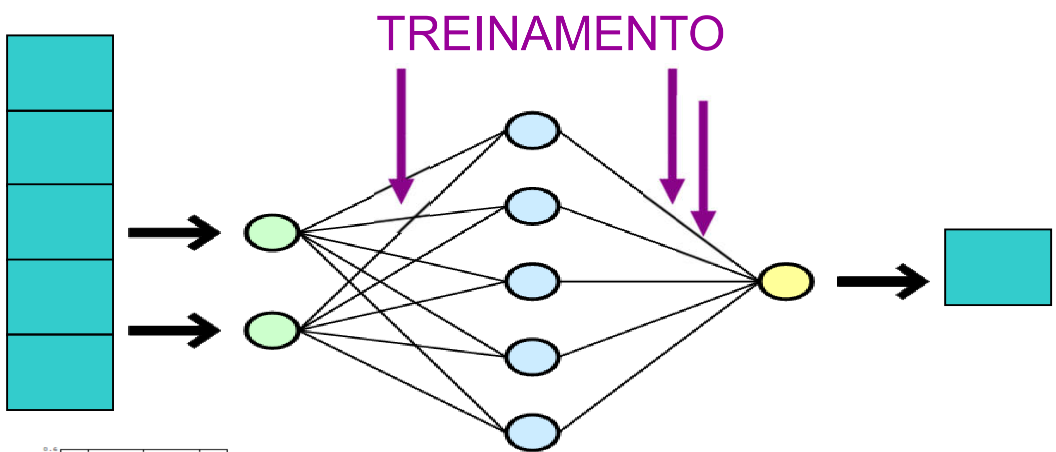


Os mapeamentos das camadas anteriores pode ser usados pelo neurônio de saída para formar mapeamentos mais complexos (separação não-linear)



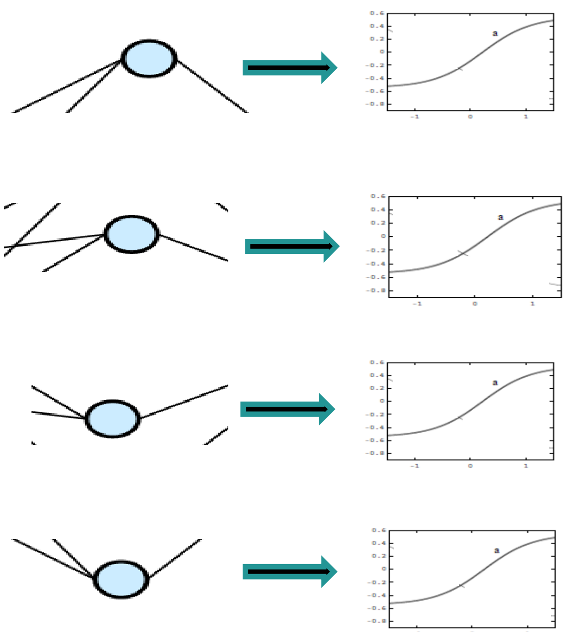
O Treinamento de vários neurônios permite o ajuste dos parâmetros (pesos e bias) modelando diferentes partições

MLP: Treinamento e Aplicação

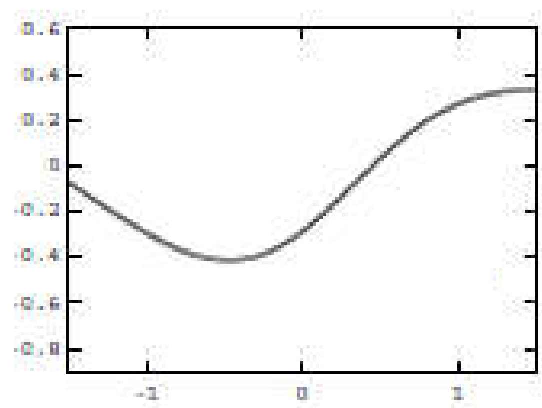


Aproximação

De Funções

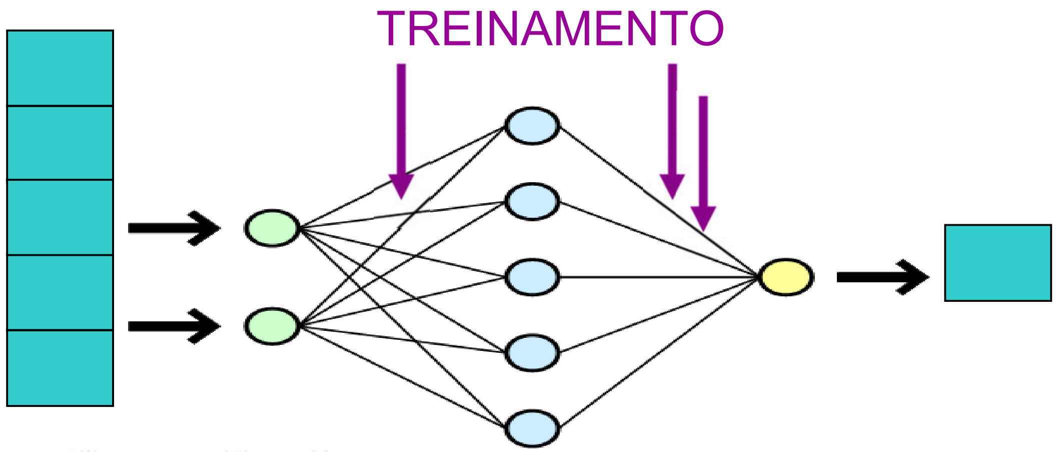


Os mapeamentos das camadas anteriores podem ser usados pelo neurônio de saída para formar **mapeamentos mais complexos** (funções aproximadas complexas)

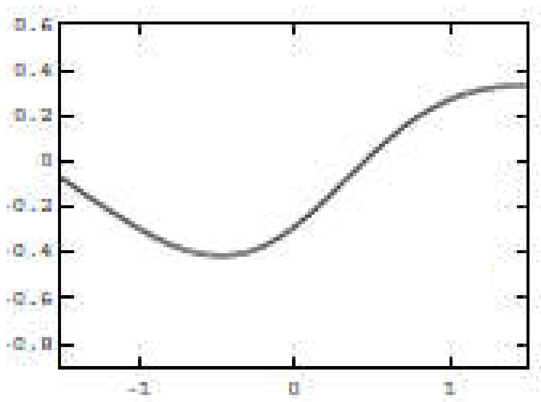


O Treinamento de vários neurônios permite o ajuste dos parâmetros (pesos e bias) modelando diferentes saídas no nível hidden

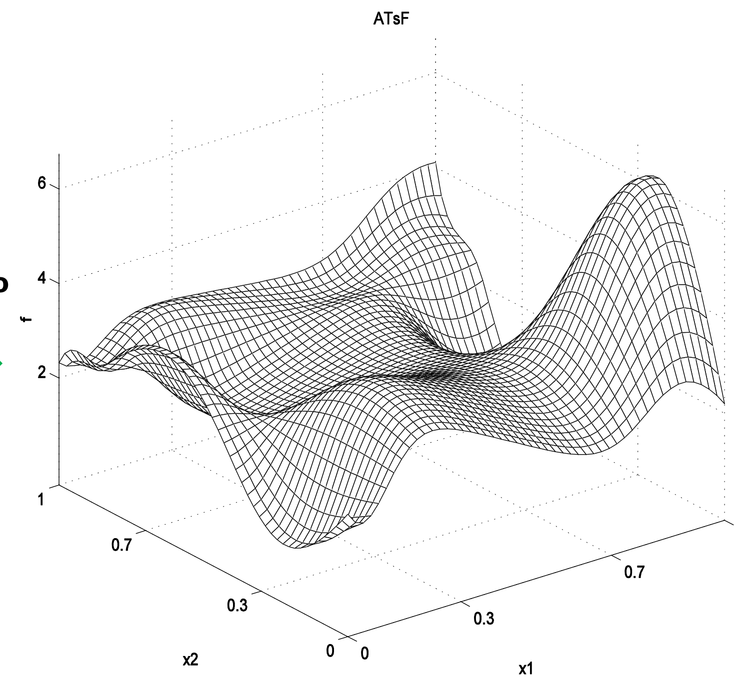
MLP: Treinamento e Aplicação



Aproximação
de Funções

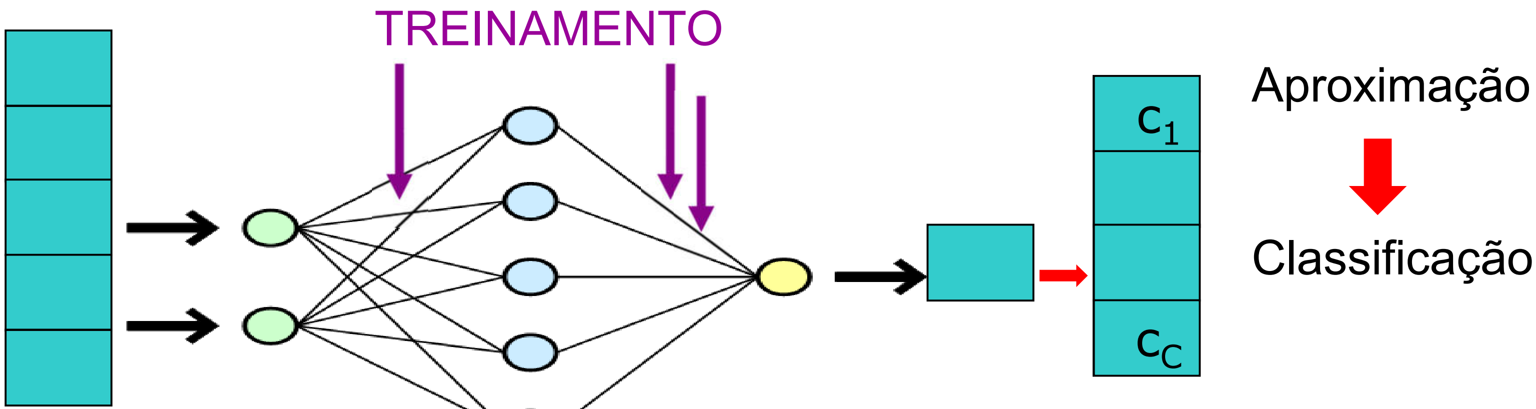


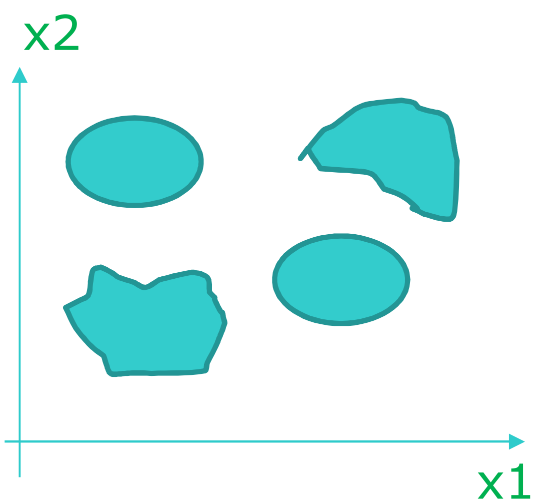
Mas esta ideia pode ser estendida para modelos multidimensionais (**2 entradas 1 saída como no exemplo ao lado**)



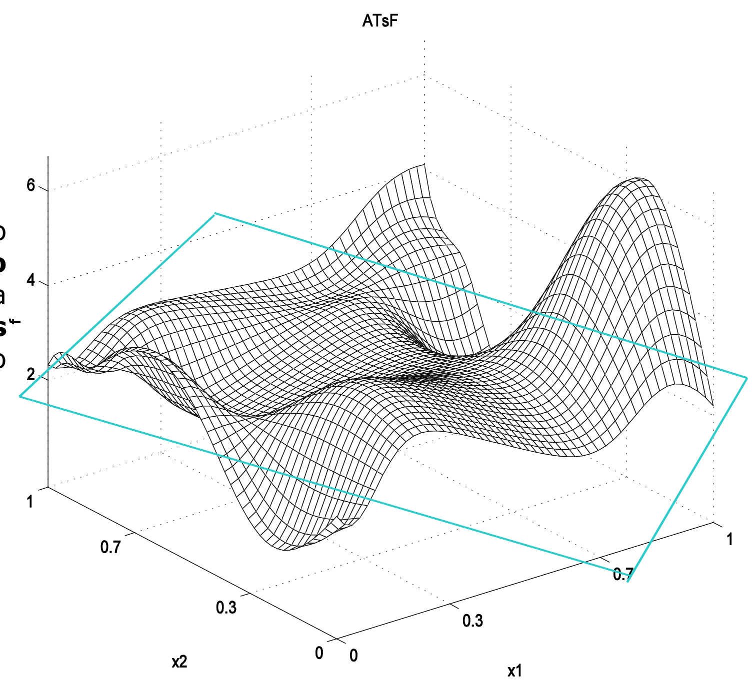
O treinamento permite o ajuste dos parâmetros (pesos e bias) modelando a
curva do mapeamento entrada – saída
(**1 entrada e 1 saída no exemplo acima**)

MLP: Treinamento e Aplicação

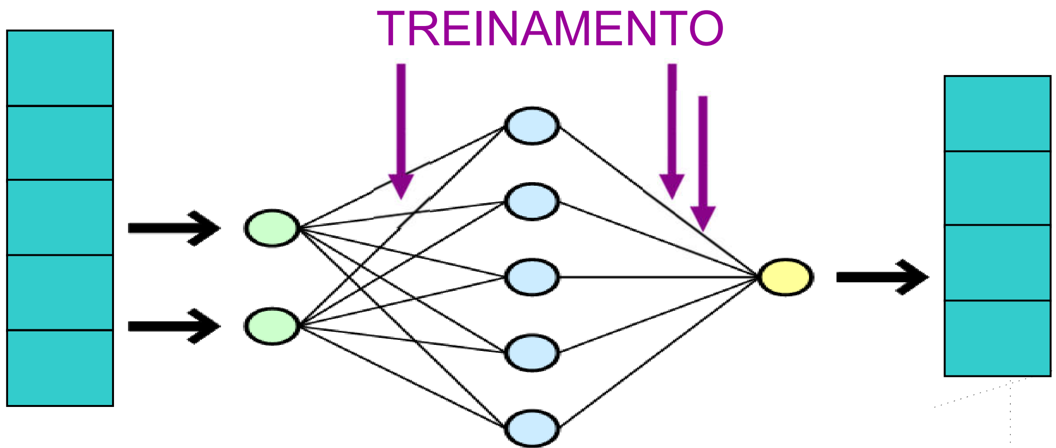




O mapeamento aproximado pode ser **limiarizado** (corte num nivel) de forma a produzir **diferentes grupos** (classes) no espaço de entrada

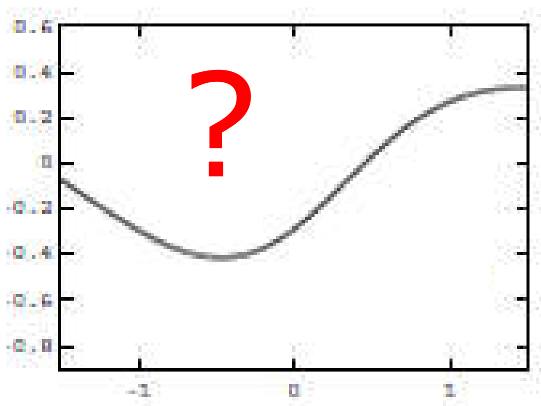


MLP: Treinamento e Aplicação

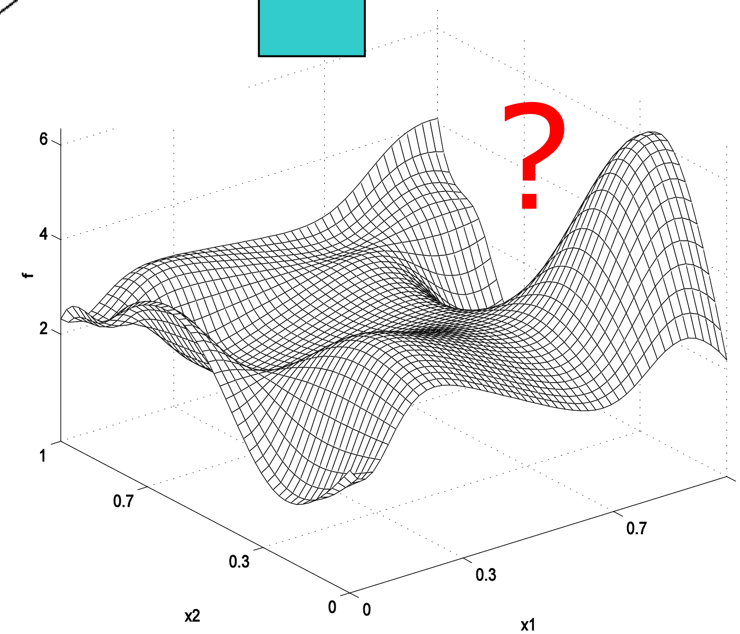


Aproximação

de Funções

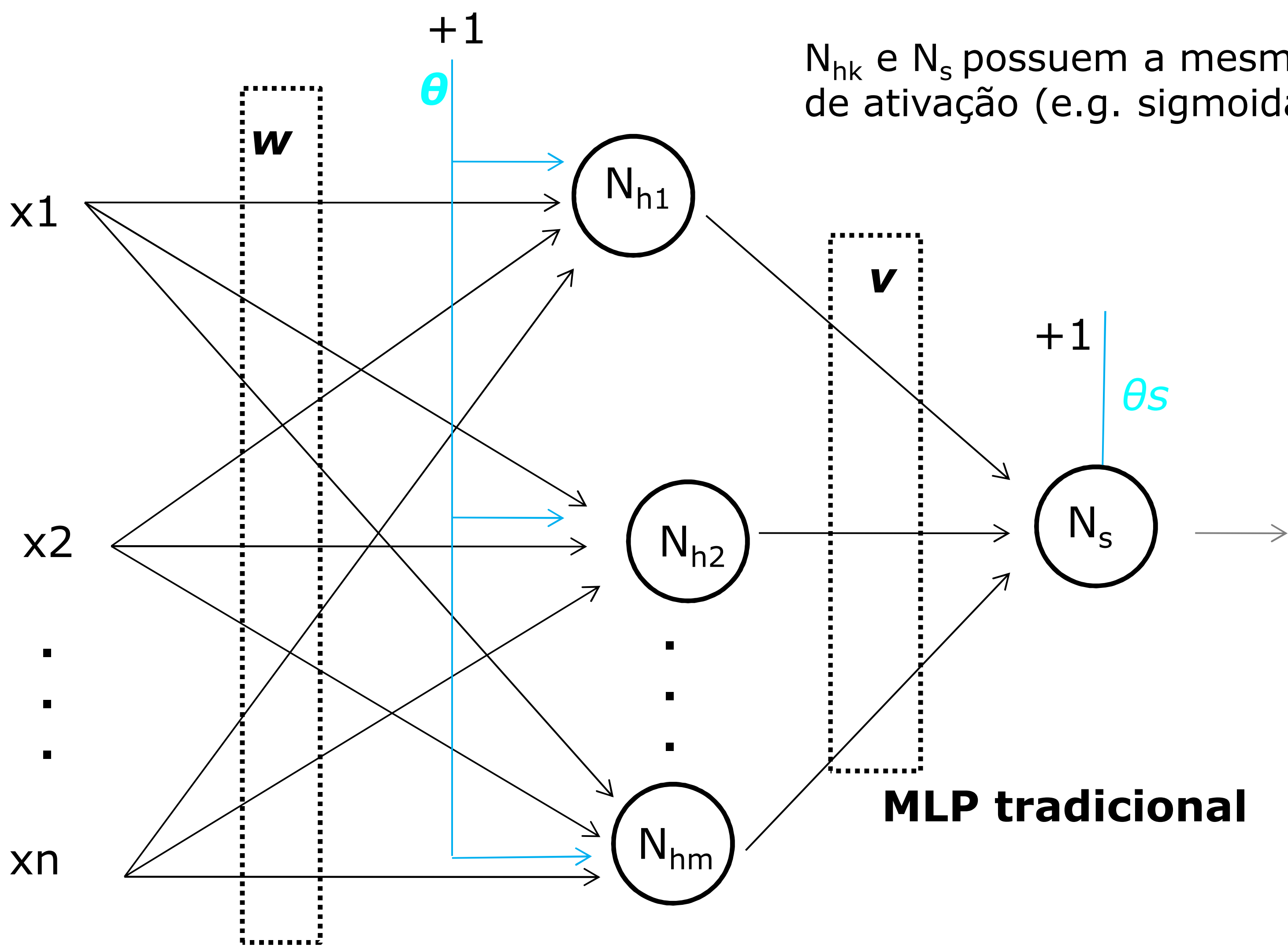


(1 entrada e 1 saída no exemplo acima)



2 entradas 1 saída como no exemplo ao lado

Multi-Layer Perceptron

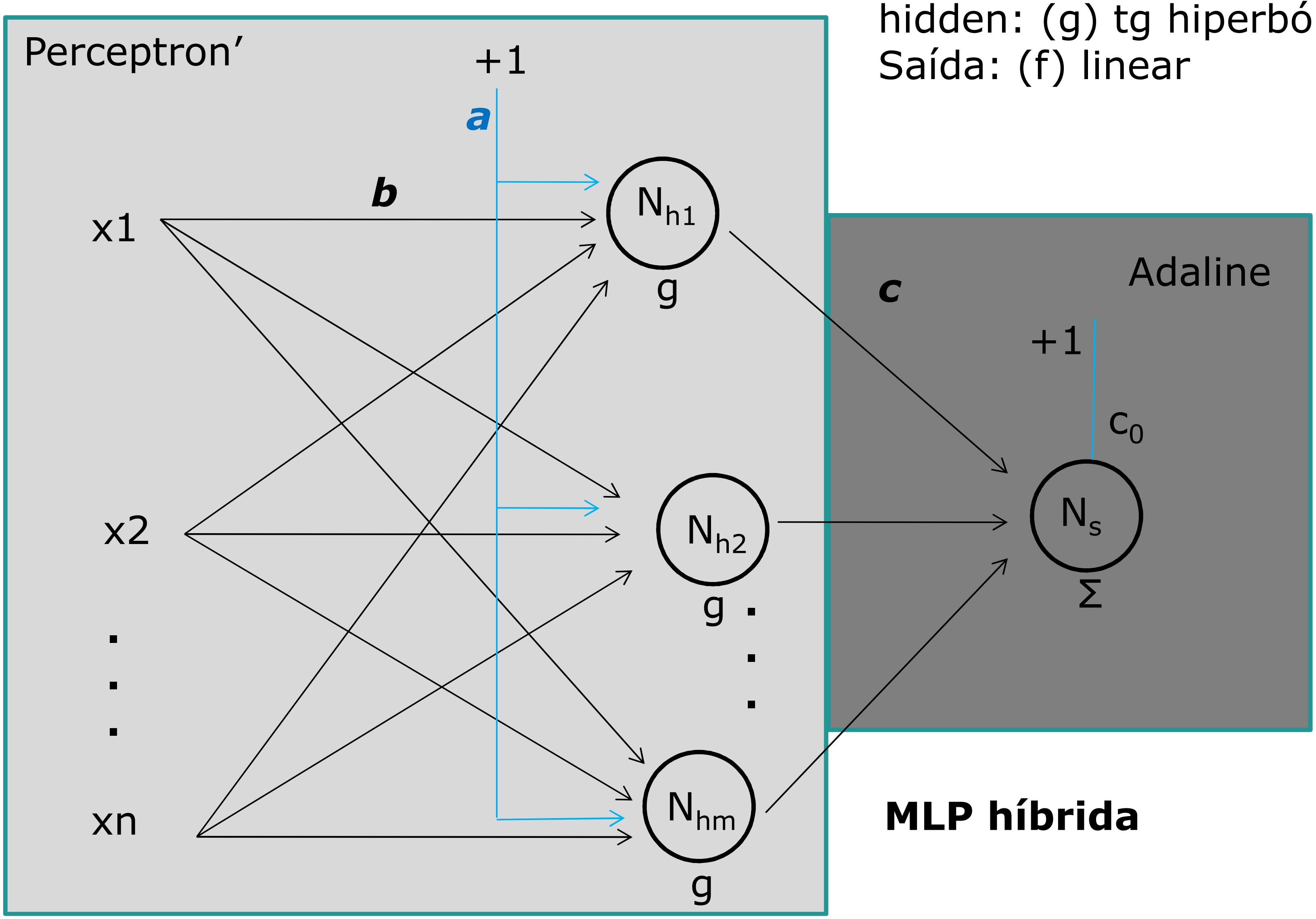


N_{hk} e N_s possuem a mesma função de ativação (e.g. sigmoidal)

MLP tradicional

MLP para aproximação de funções

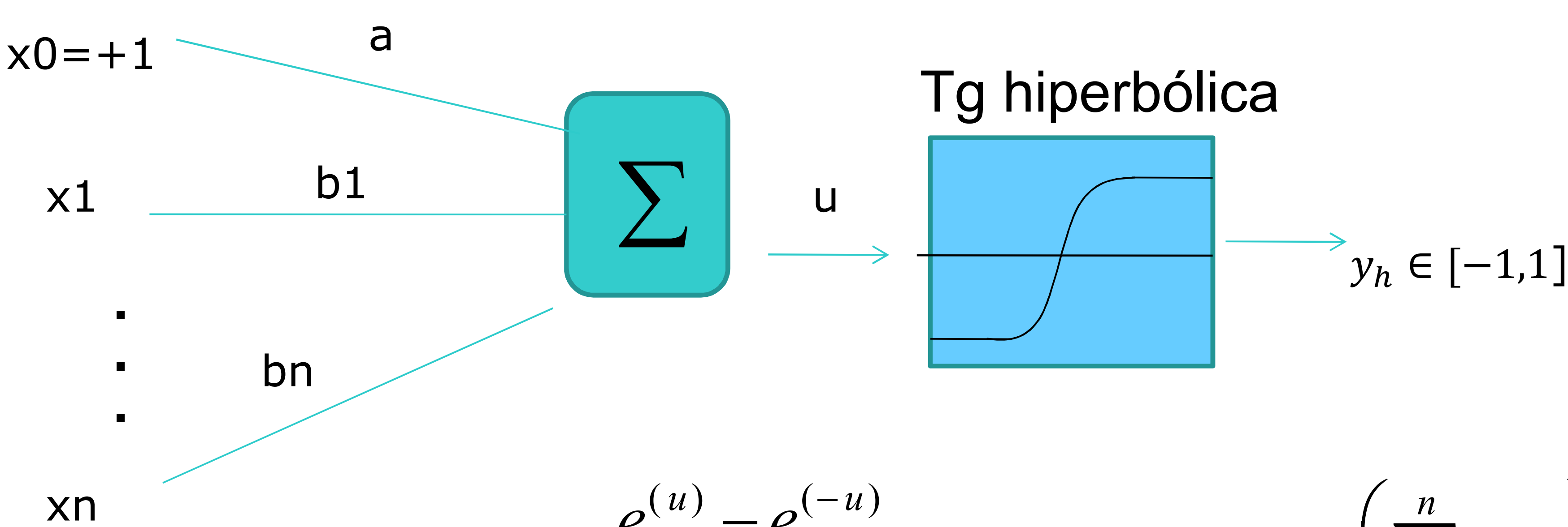
hidden: (g) tg hiperbólica
Saída: (f) linear



MLP híbrida

MLP híbrida: cam Tg Hiperbólica (N_h)

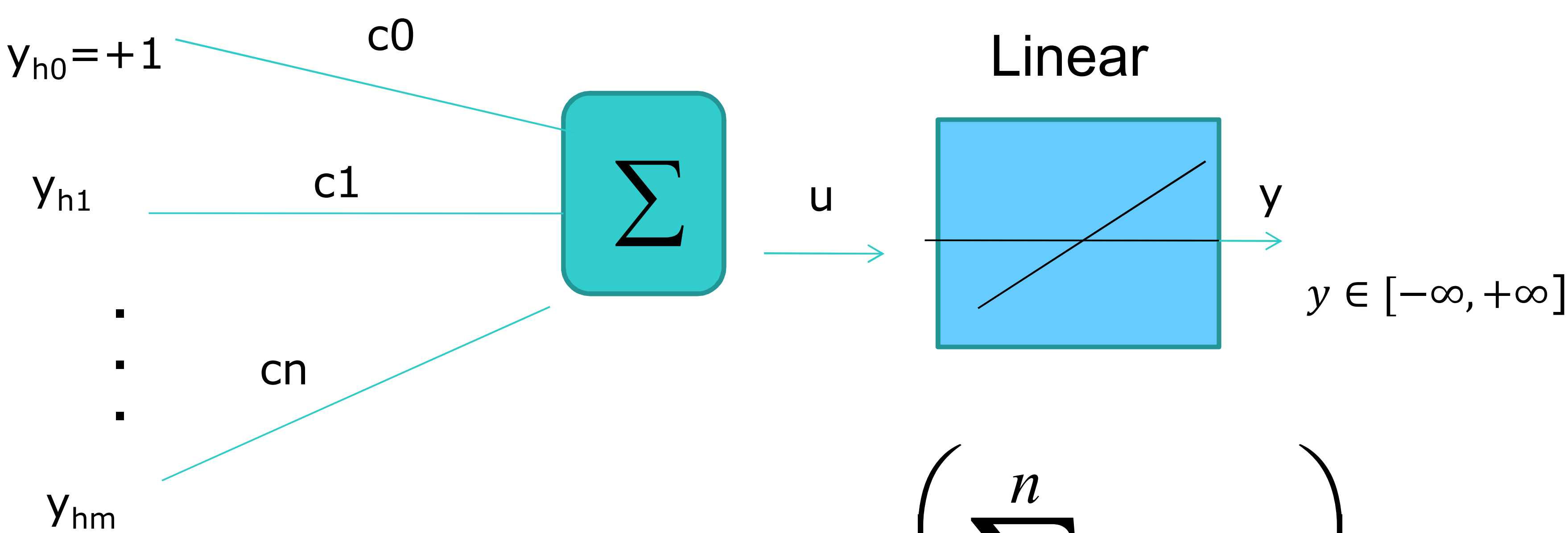
f = Tangente Hiperbólica (hidden)



$$y_h = \frac{e^{(u)} - e^{(-u)}}{e^{(u)} + e^{(-u)}}, \quad \text{onde} \quad u = a + \left(\sum_{i=1}^{n} b_i x_i \right)$$

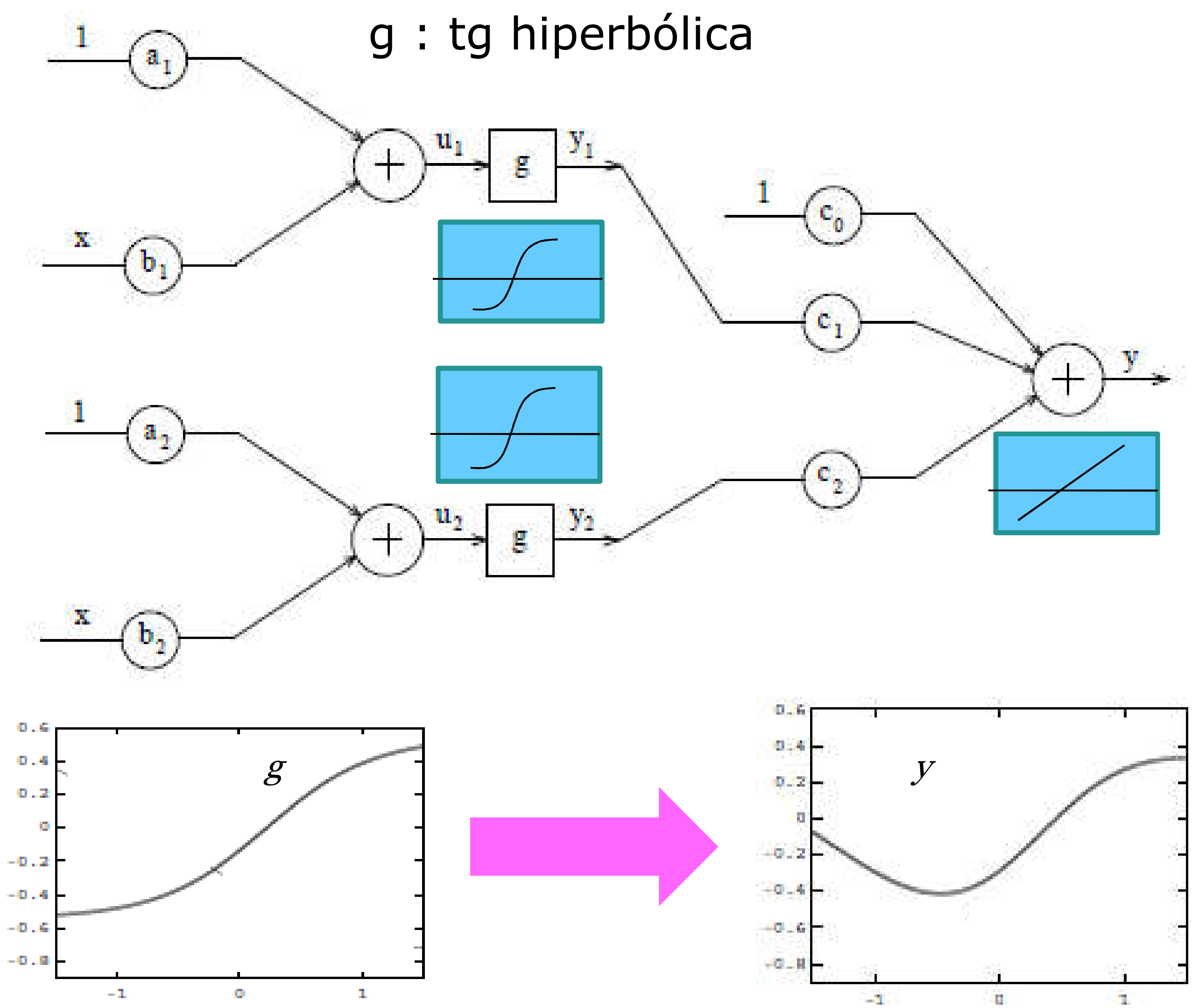
MLP híbrida: cam Linear(N_s)

f = função Linear (saída)



$$y = \left(\sum_{i=0}^{n} c_i y_{hi} \right) = u$$

MLP Híbrida (Perceptron’ + Adaline)



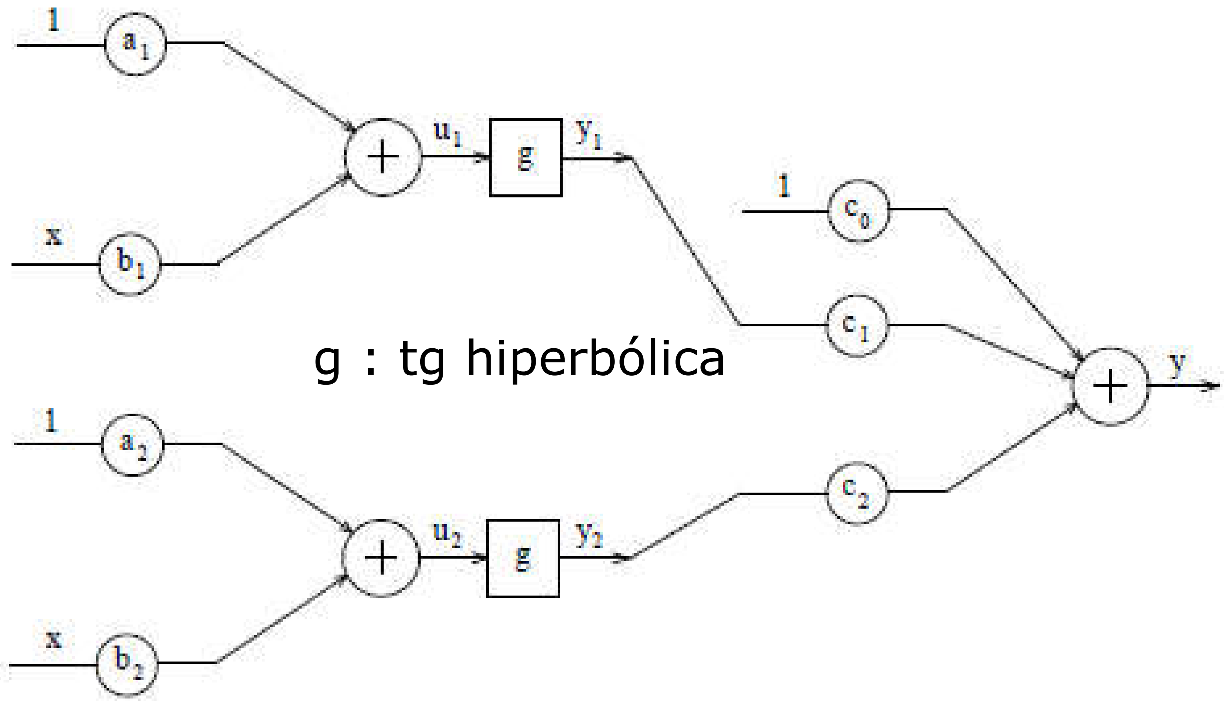
Aprendizado Conexionista (MLP híbrida)

O papel dos pesos

Aprendizado Conexionista (MLP híbrida)

O papel dos pesos

$$y = c_0 + \sum_{n=1}^{p} c_n g(b_n x + a_n)$$



g : tg hiperbólica

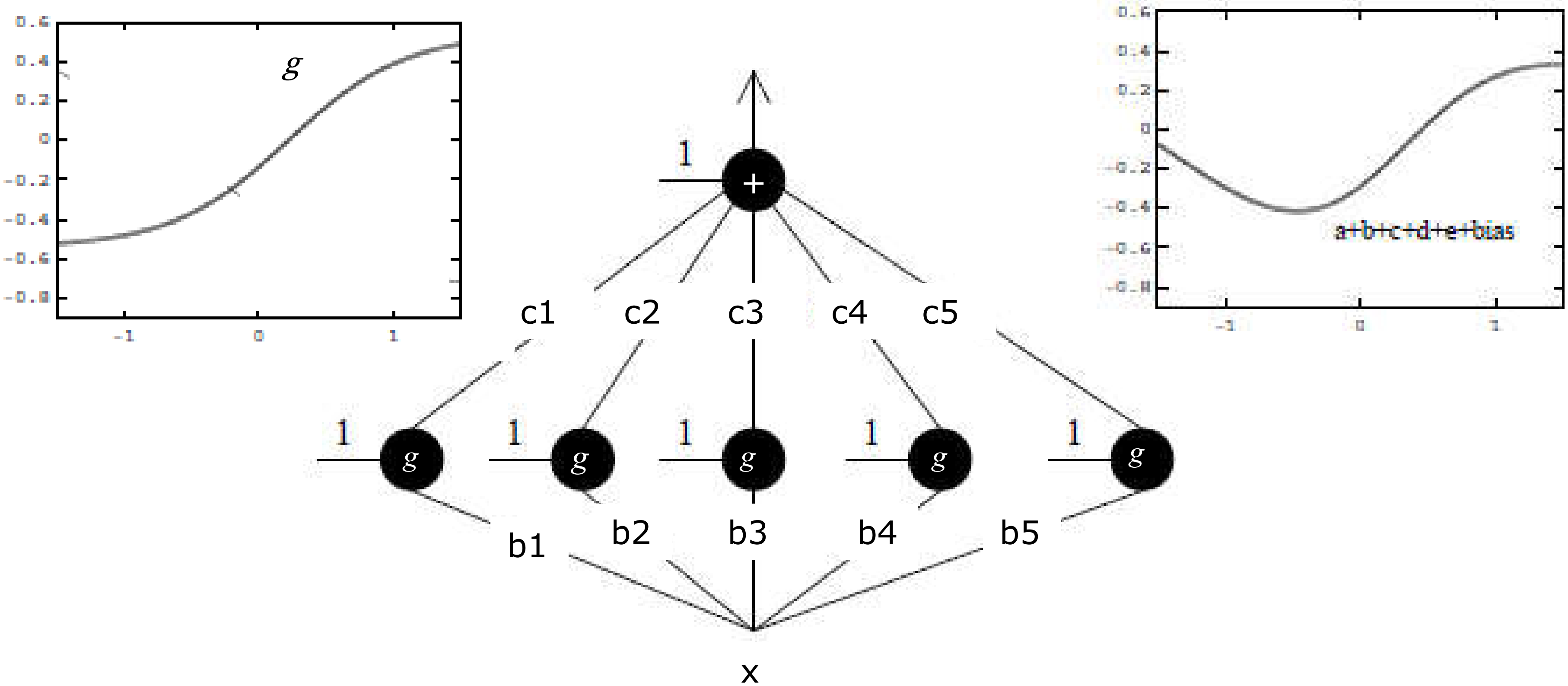
$$y = c_0 + c_1 g(b_1 x + a_1) + c_2 g(b_2 x + a_2)$$

Aprendizado Conexionista: pesos

Exemplo: Forma "construtiva" de aproximação de um mapeamento não-linear

Aprendizado Conexionista: pesos

<u>Exemplo</u>: Forma "construtiva" de aproximação de um mapeamento não-linear



$$f(\mathrm{w}) = \underbrace{c_1 g(b_1 x + a_1)}_{a} + \underbrace{c_2 g(b_2 x + a_2)}_{b} + \underbrace{c_3 g(b_3 x + a_3)}_{c} + \underbrace{c_4 g(b_4 x + a_4)}_{d} + \underbrace{c_5 g(b_5 x + a_5)}_{e} + \underbrace{c_0}_{bias}$$

Adaptado de *4072 – Prof. Fernando J. Von Zuben DCA/FEEC/Unicamp*

Aprendizado Conexionista: pesos

$$f(\mathrm{w}) = \underbrace{c_1 g(b_1 x + a_1)}_{a} + \underbrace{c_2 g(b_2 x + a_2)}_{b} +$$

$$+ \underbrace{c_5 g(b_5 x + a_5)}_{e}$$

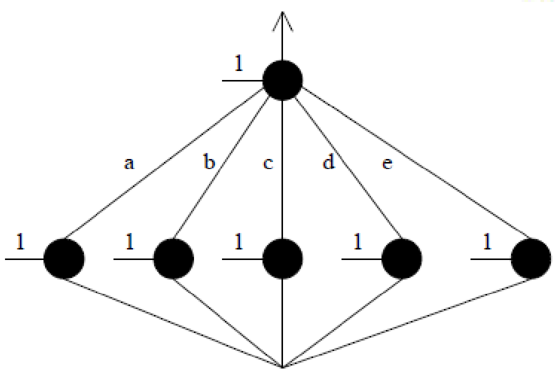
$$+ \underbrace{c_3 g(b_3 x + a_3)}_{c}$$

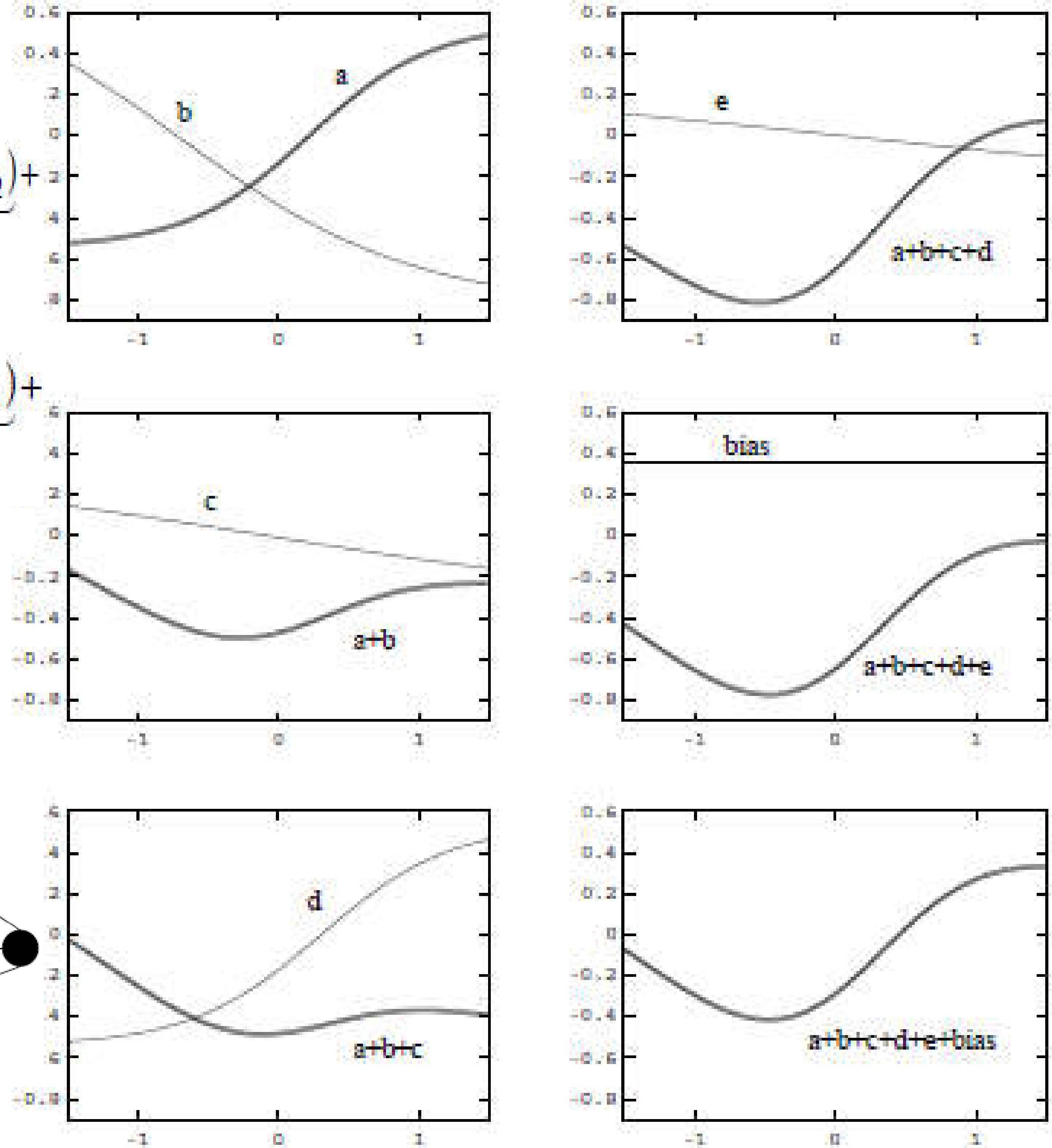
$$+ \underbrace{c_0}_{bias}$$

$$+ \underbrace{c_4 g(b_4 x + a_4)}_{d}$$

Aprendizado Conexionista: pesos

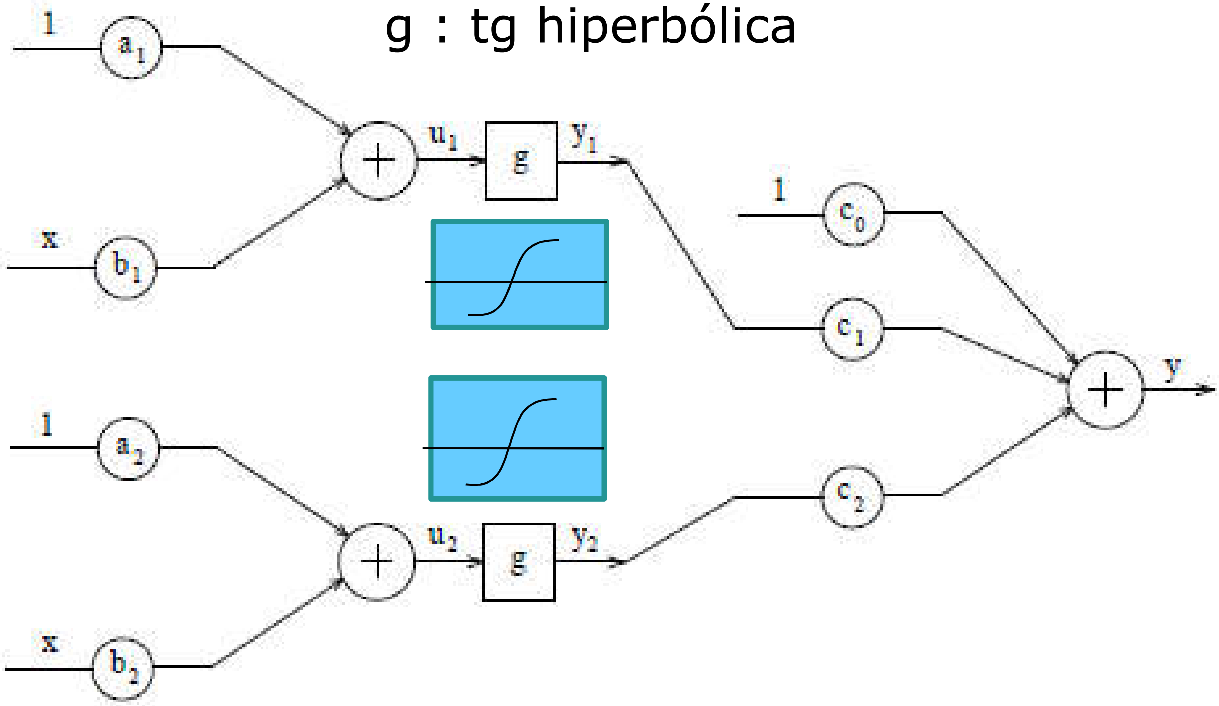
$$f(\mathrm{w}) = \underbrace{c_1 g(b_1 x + a_1)}_{a} + \underbrace{c_2 g(b_2 x + a_2)}_{b} + \underbrace{c_3 g(b_3 x + a_3)}_{c} + \underbrace{c_4 g(b_4 x + a_4)}_{d} + \underbrace{c_5 g(b_5 x + a_5)}_{e} + \underbrace{c_0}_{bias}$$

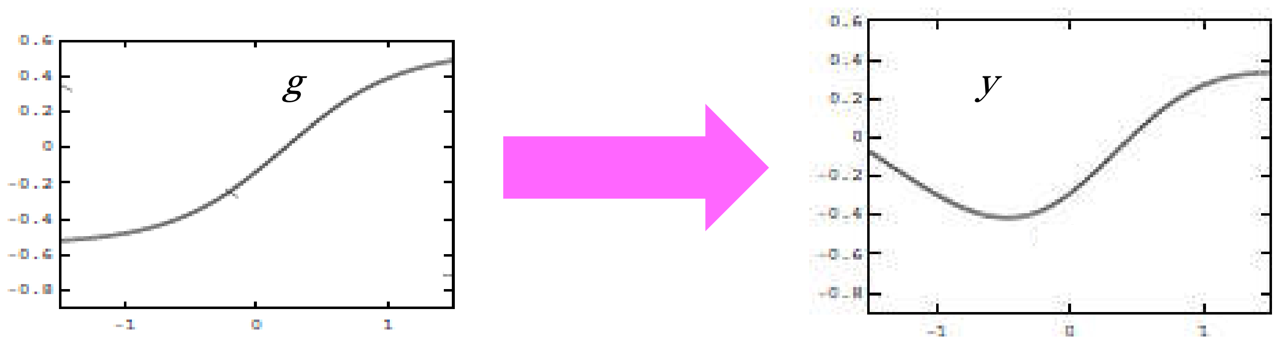




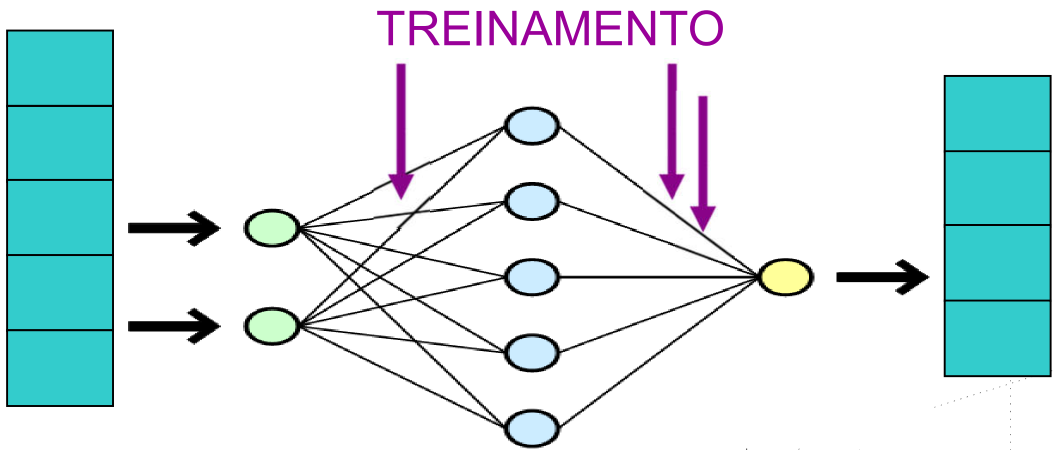
MLP Híbrida (Perceptron' + Adaline)

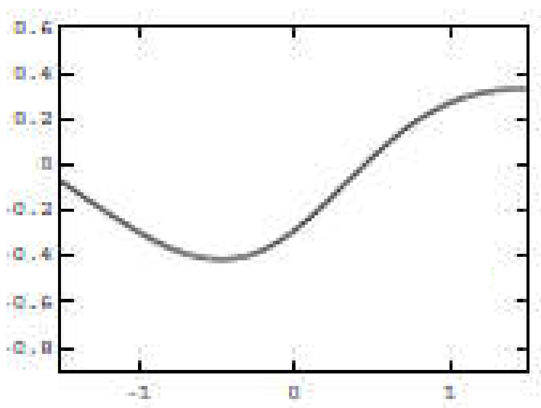
g : tg hiperbólica



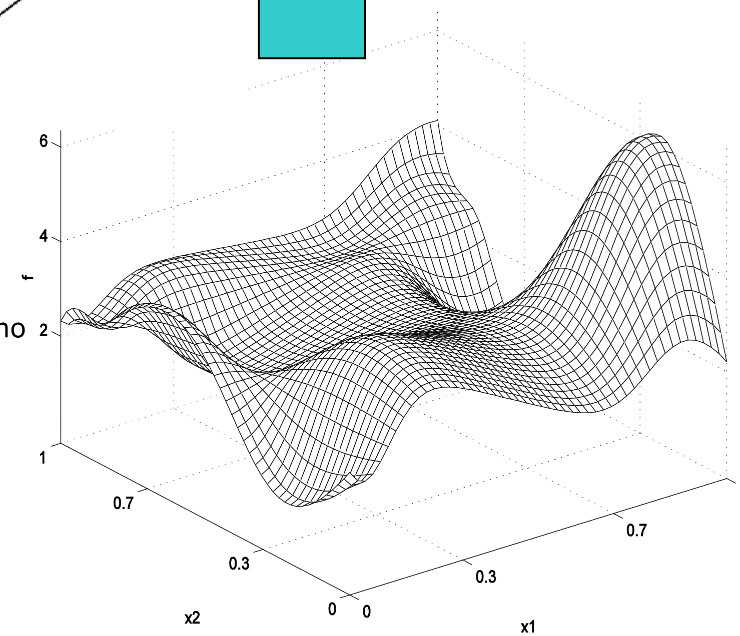


MLP Híbrida (Perceptron' + Adaline)





Mas esta ideia pode ser estendida para modelos multidimensionais (2 entradas 1 saída como no exemplo ao lado)



O Treinamento permite o ajuste dos parâmetros (pesos e bias) o qual modela a curva do mapeamento entrada – saída (1 entrada e 1 saída no exemplo acima)

Aproximação de Funções