Universidade Tecnológica Federal do Paraná - Campus Curitiba

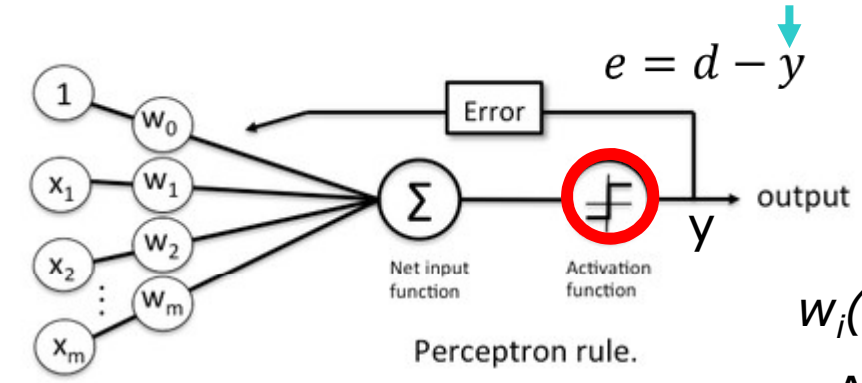
IA- Redes Neurias

Rede Multi-Layer Perceptron (MLP)

Prof. Myriam Delgado

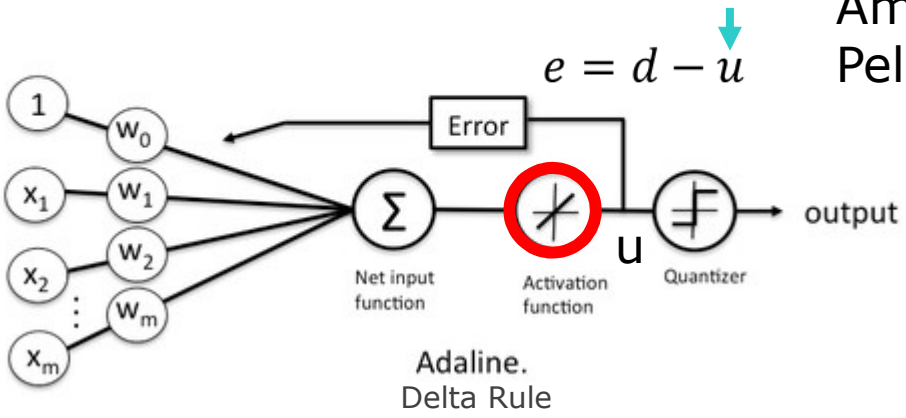
RNAs: Arquitetura (revisão)

Perceptron e Adaline



$w_i(t+1) = w_i(t) + \eta\ e\ x_i$

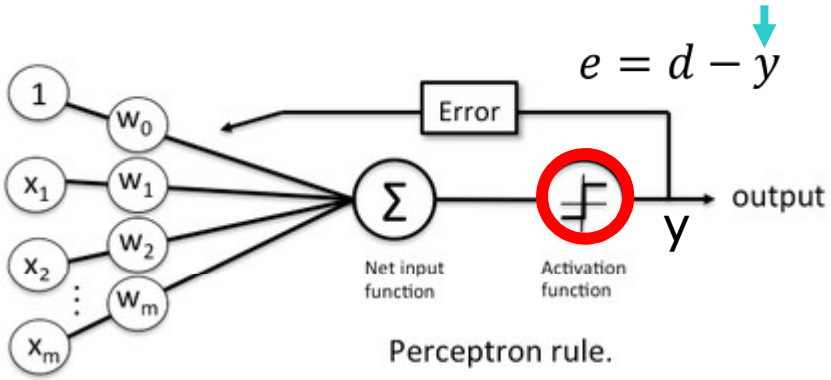
Ambos treinados
Pela regra delta



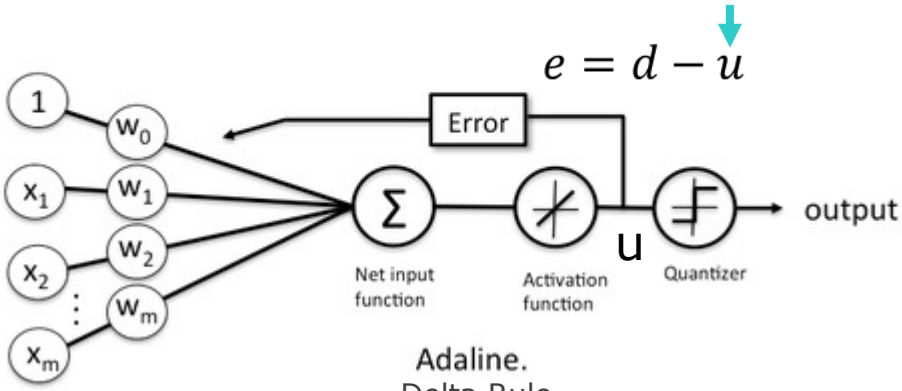
RNAs: Arquitetura (revisão)

Perceptron e Adaline

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



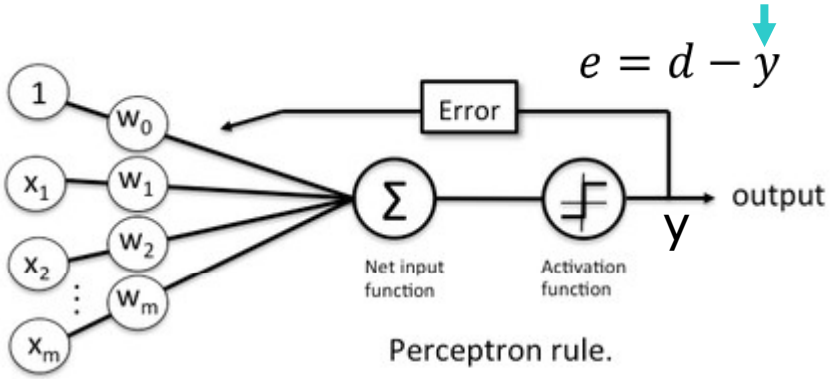
Perceptron rule.



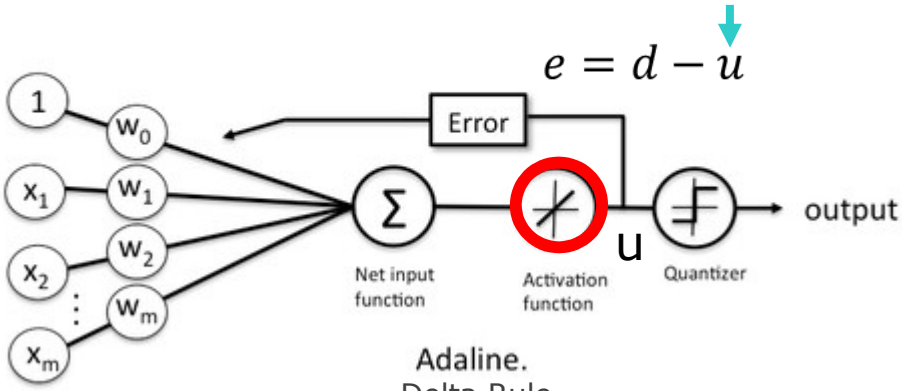
Adaline.
Delta Rule

RNAs: Arquitetura (revisão)

Perceptron e Adaline



Perceptron rule.

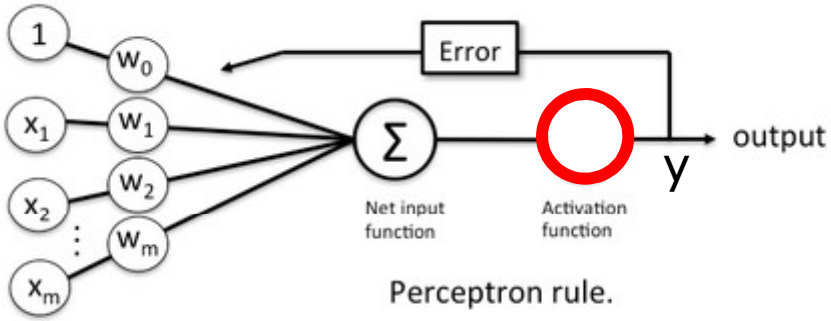


Adaline.
Delta Rule

Y∈ {−∝, +∝}
Aplicação em Aproximação De Funções (combinação Linear de Funções não Lineares)

RNAs: Arquitetura (revisão)

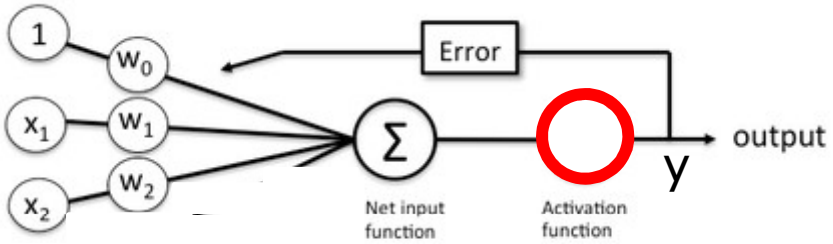
Perceptron Atual



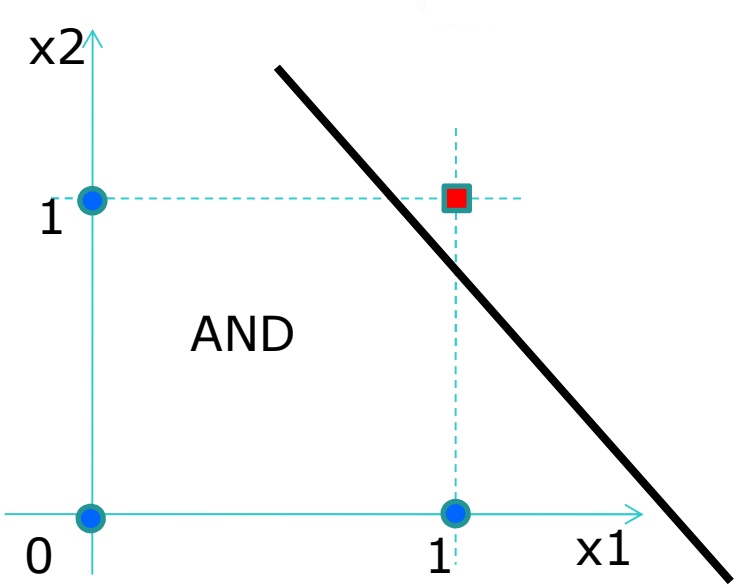
Perceptron Atual: MLP de camada única
(aceita diferentes funções de ativação)

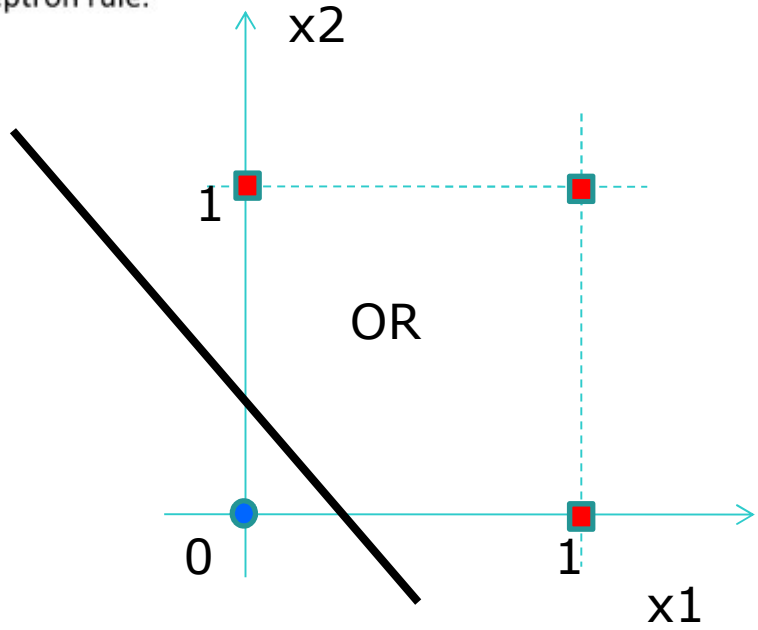
RNAs: Arquitetura (revisão)

Perceptron Atual



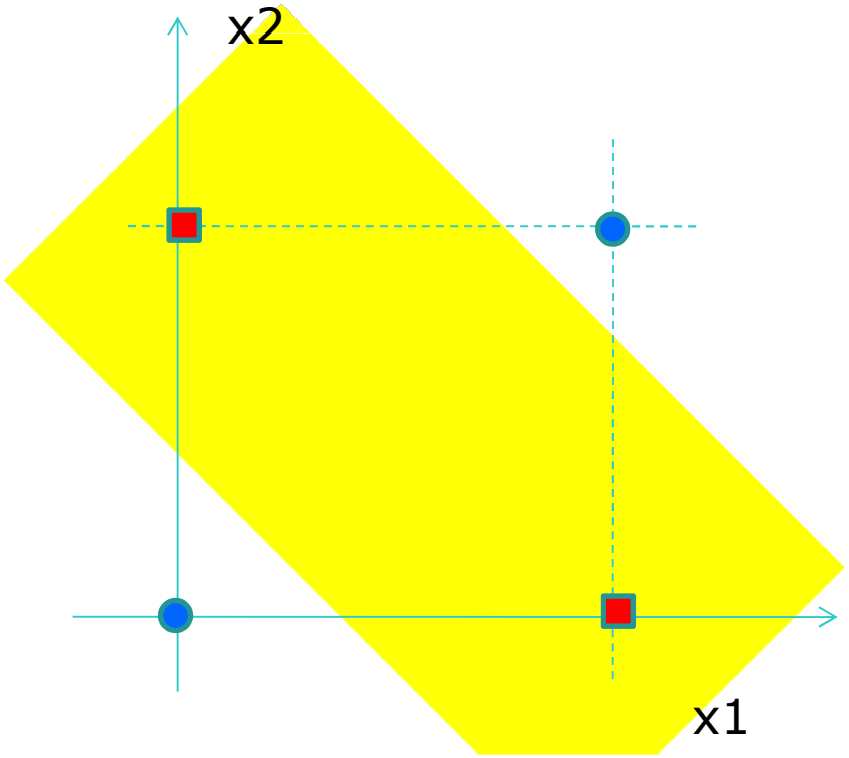
Perceptron rule.





RNAs: Perceptron (resolução de problemas)

XOR lógico: não linearmente separável
Perceptron não resolve

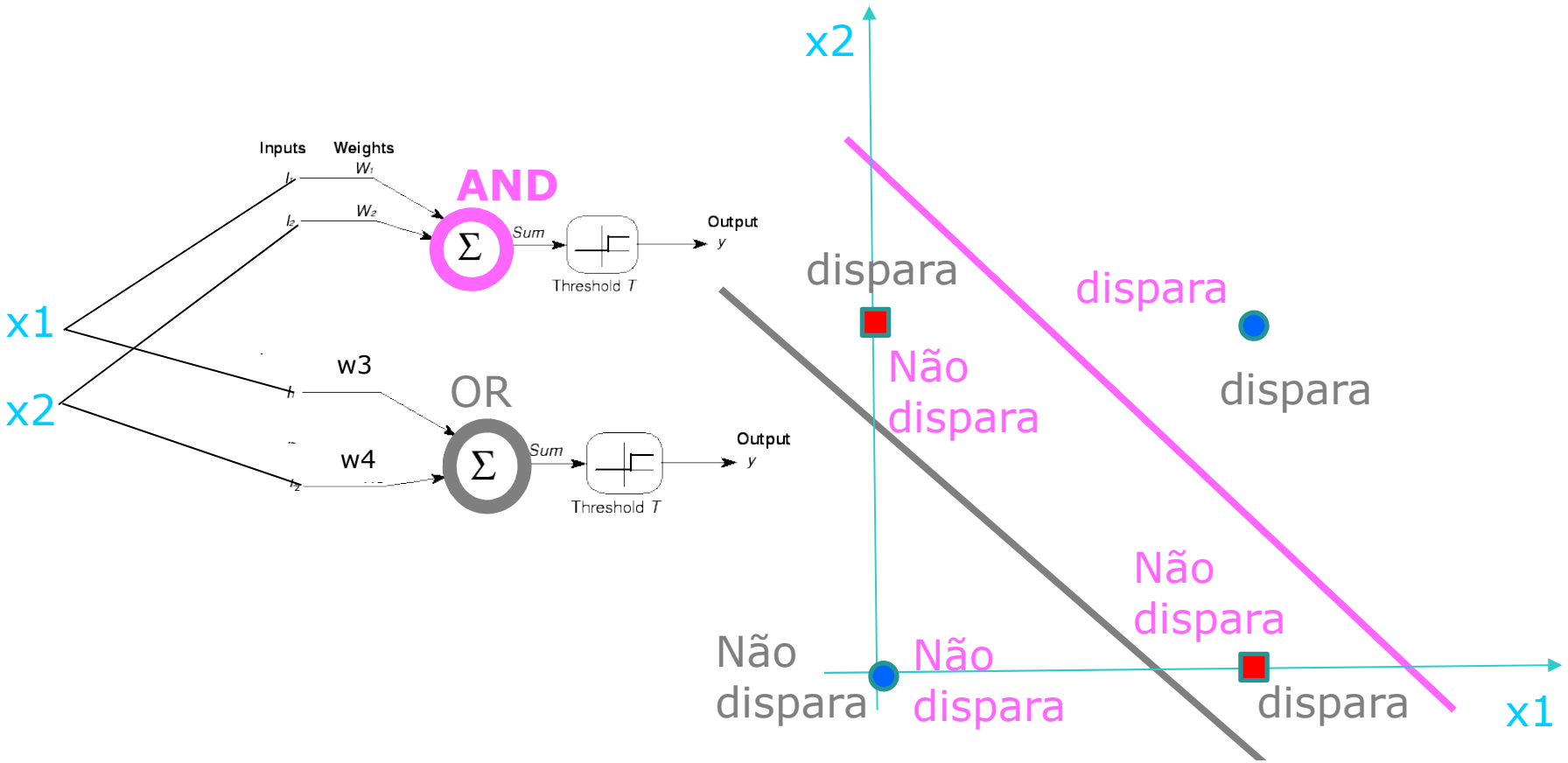


XOR Lógico

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

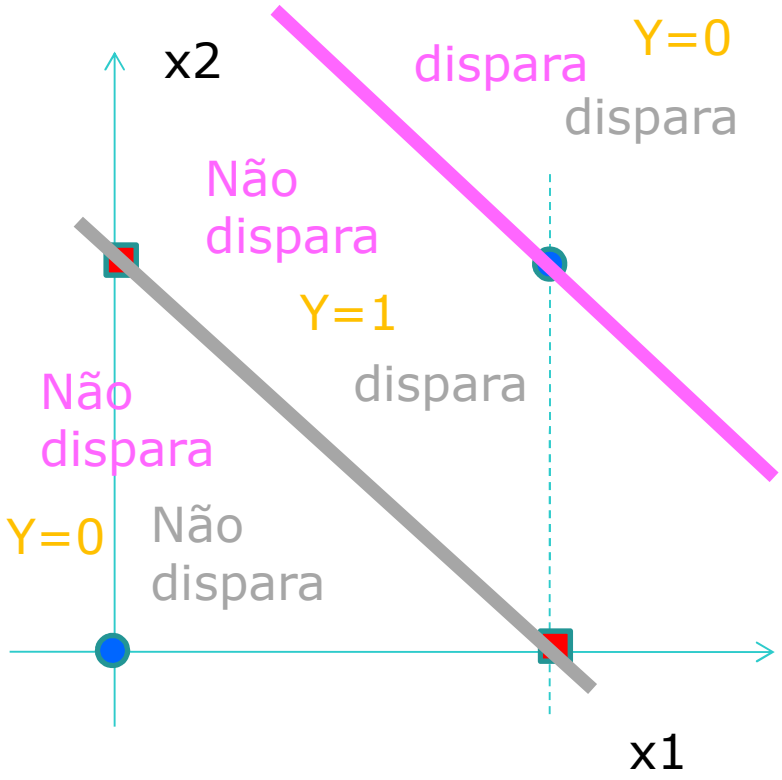
Perceptron → MLP para resolver XOR

Rede Neural (1 camada): partições independentes no espaço



RNAs: MLP (resolução de problemas)

XOR lógico: Mais camadas combinando AND e OR



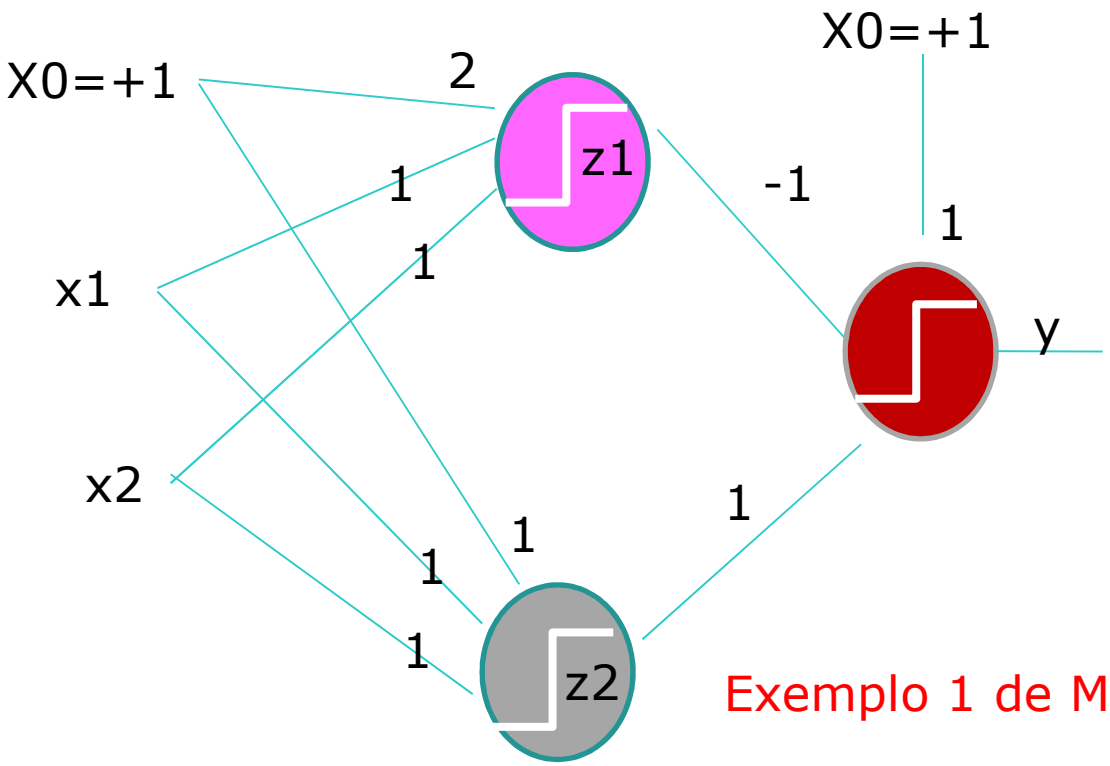
XOR Lógico

x1	x2	Z1 AND	Z2 OR	Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

RNAs: MLP (resolução de problemas)

XOR lógico: PerceptronB + Mais uma camada

Y=x1 XOR x2 ↔ not (x1 AND x2) AND! (x1 OR x2)



XOR Lógico

x1	x2	Z1 AND	Z2 OR	Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Exemplo 1 de MLP para o XOR

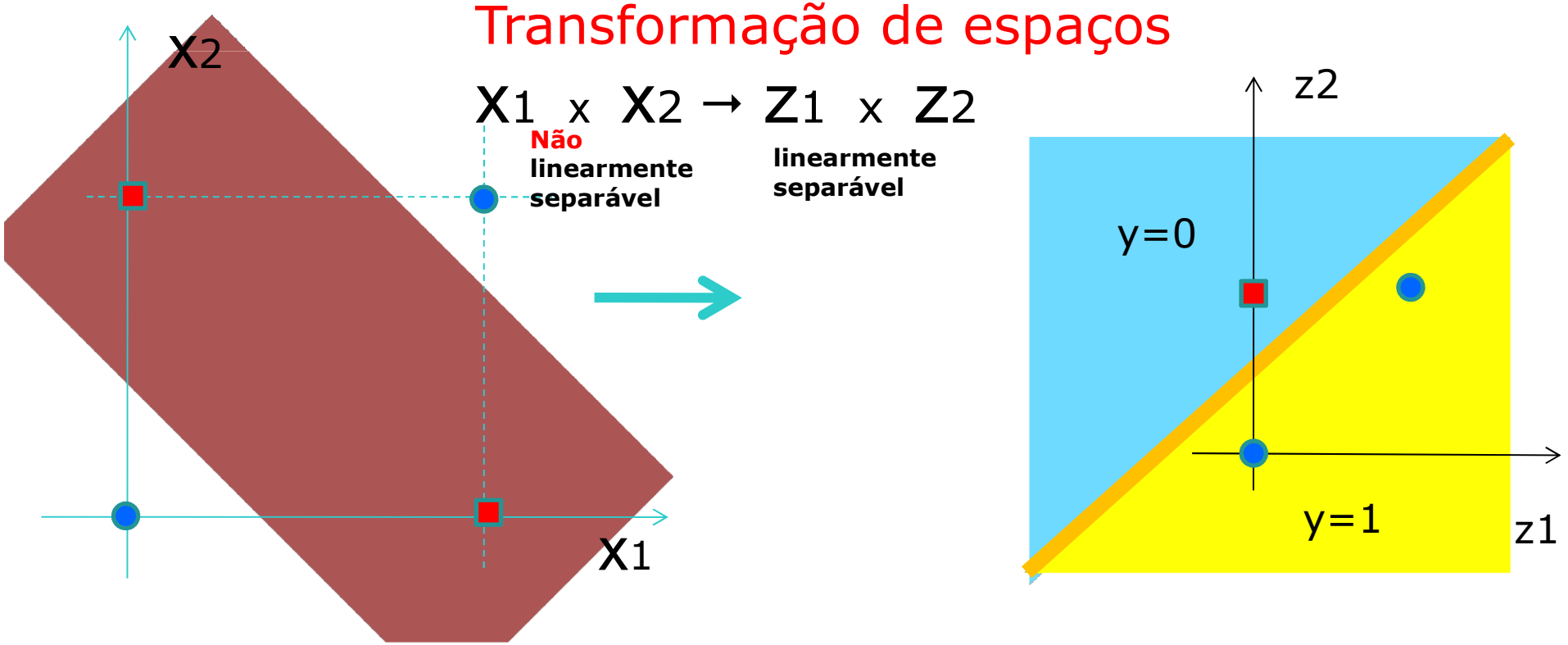
RNAs: Neurônio MCP (resolução de problemas)

XOR lógico: não linearmente separável

Por que as redes multicamadas são capazes de resolver XOR ?

Transformação de espaços

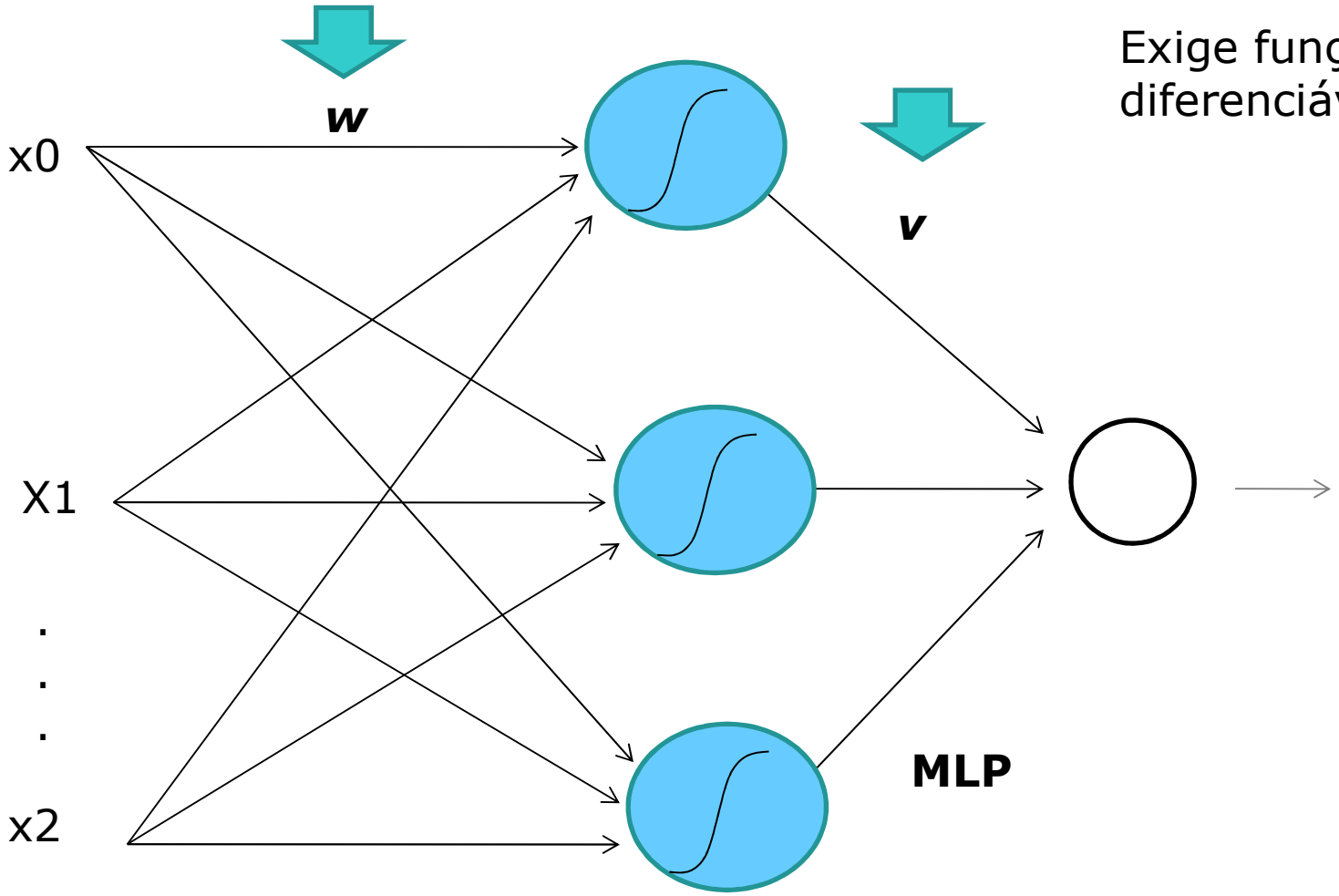
$X_1 \times X_2 \rightarrow Z_1 \times Z_2$



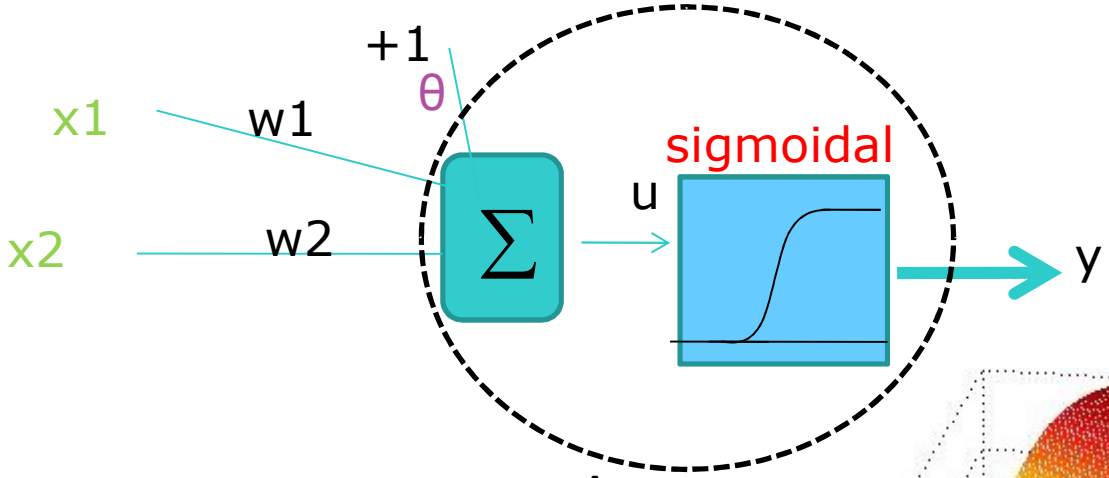
Multi-Layer Perceptron

Treinamento (descida do gradiente)

Exige função diferenciável



Neurônio na MLP: função de ativação Sigmoide

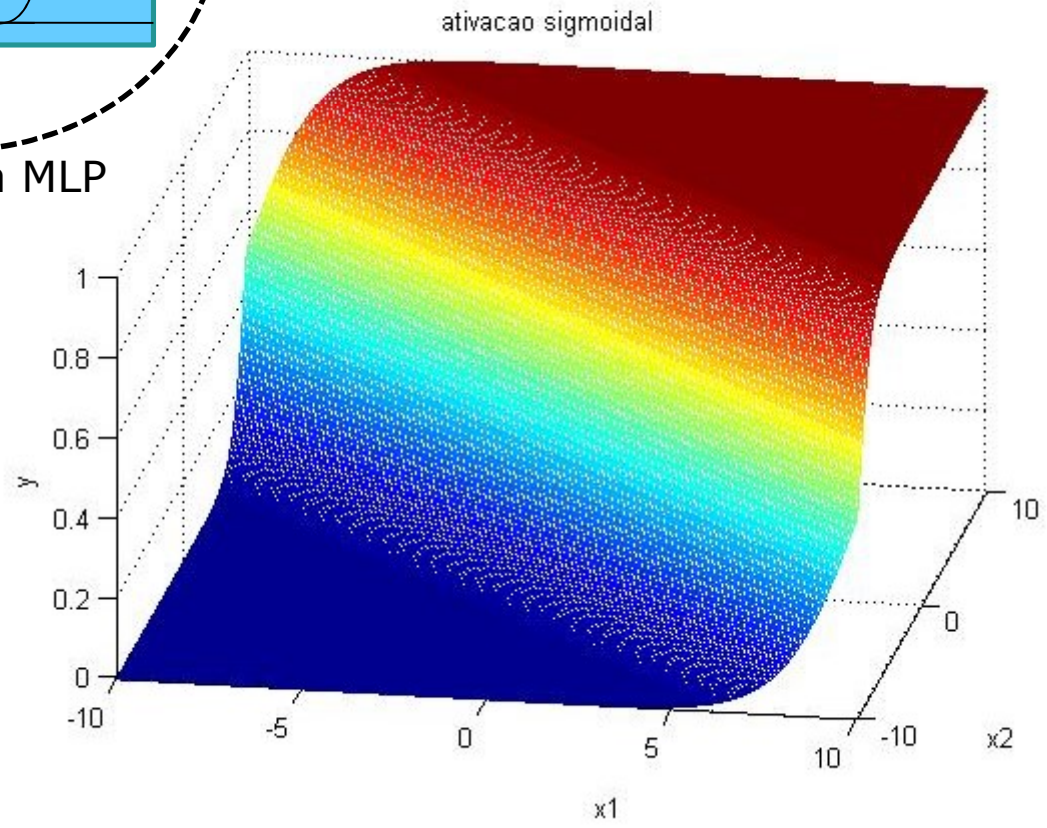


Neurônio na MLP

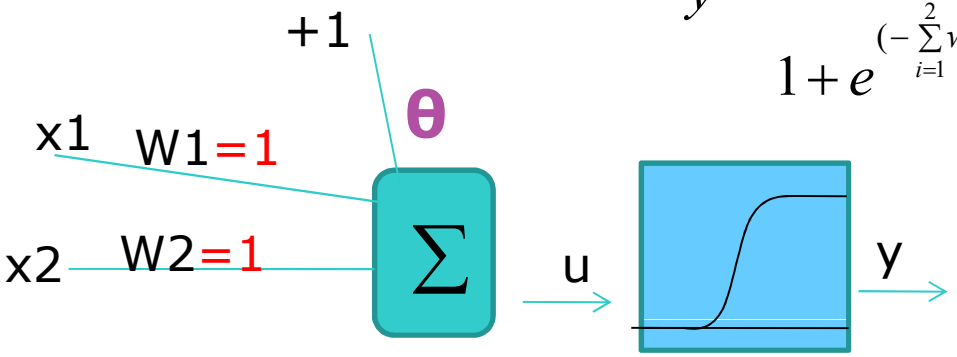
Função Ativação no espaço

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

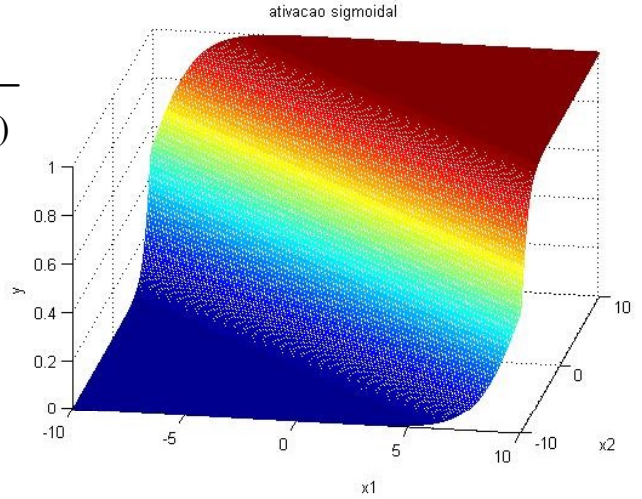
$$\text{onde} \quad u = \left(\sum_{i=1}^{2} w_i x_i + \theta \right)$$



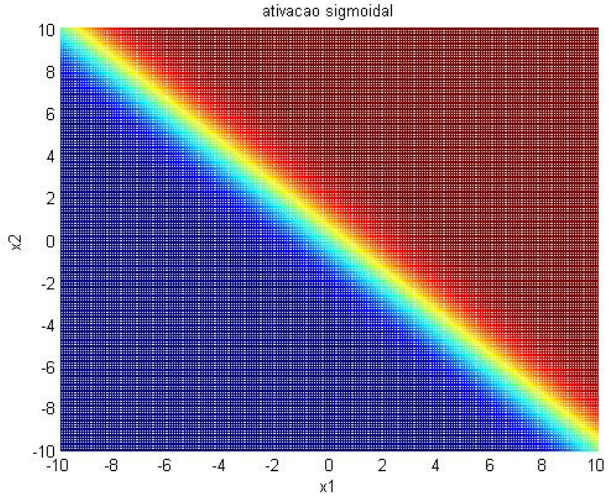
Neurônio na MLP: partição *x1* x *x2*



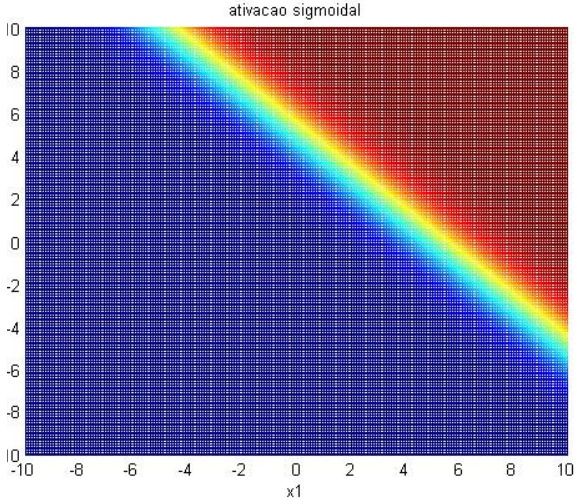
$$y = \frac{1}{1 + e^{(-\sum_{i=1}^{2} w_i x_i + \theta)}}$$



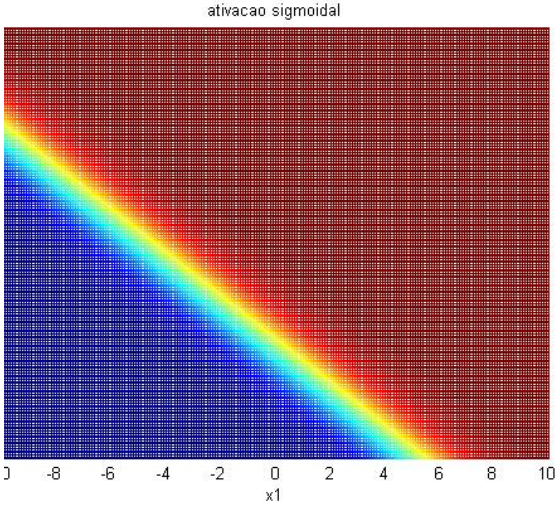
Θ =0



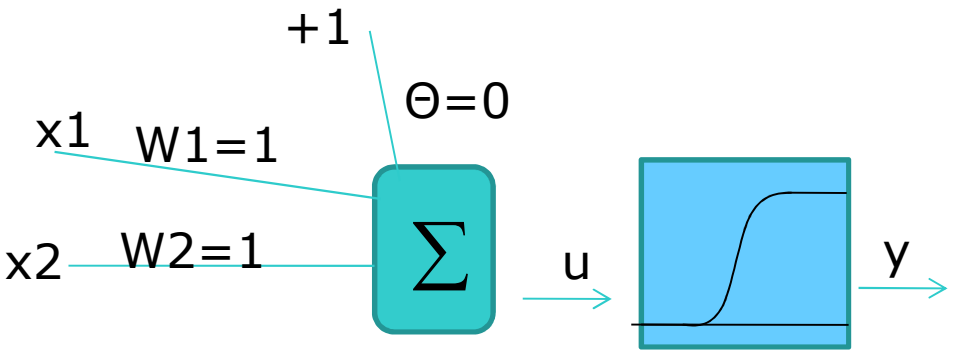
Θ =-5



Θ =+5

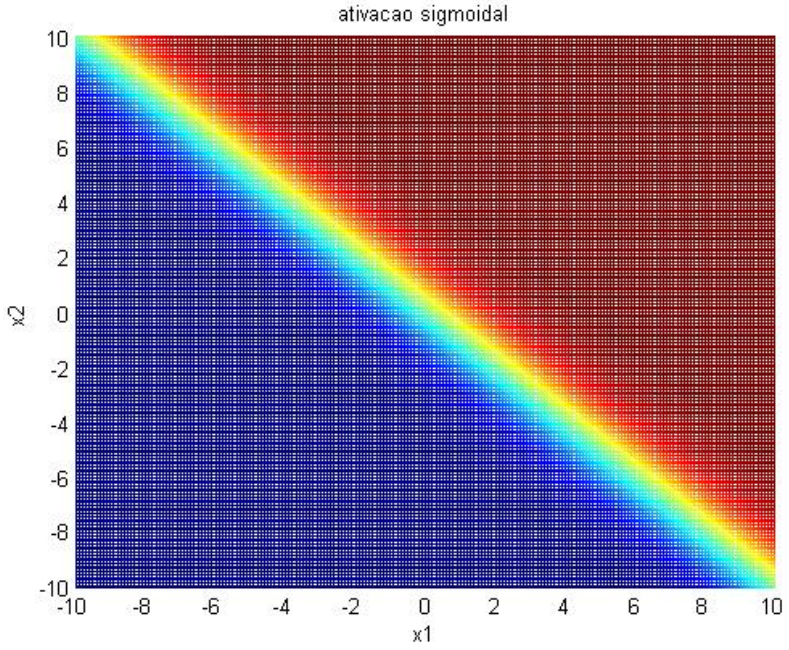


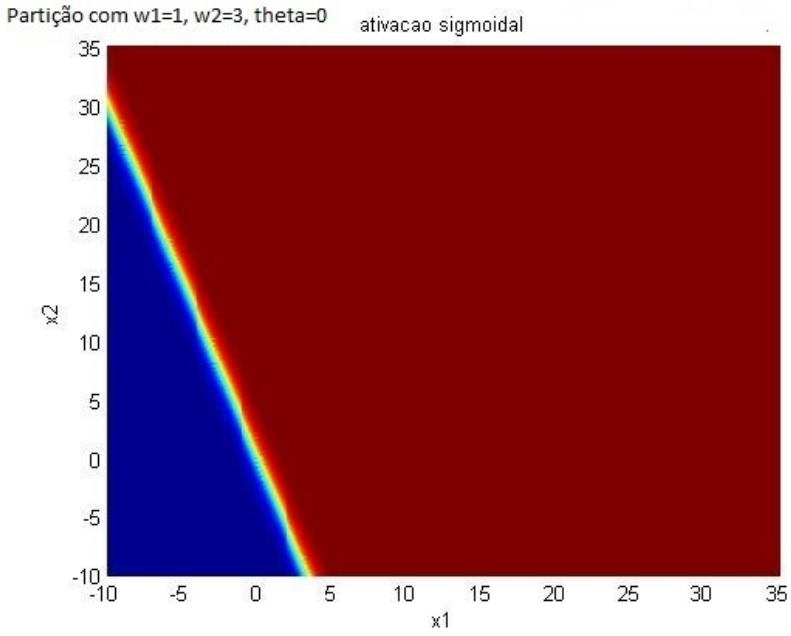
Neurônio na MLP:



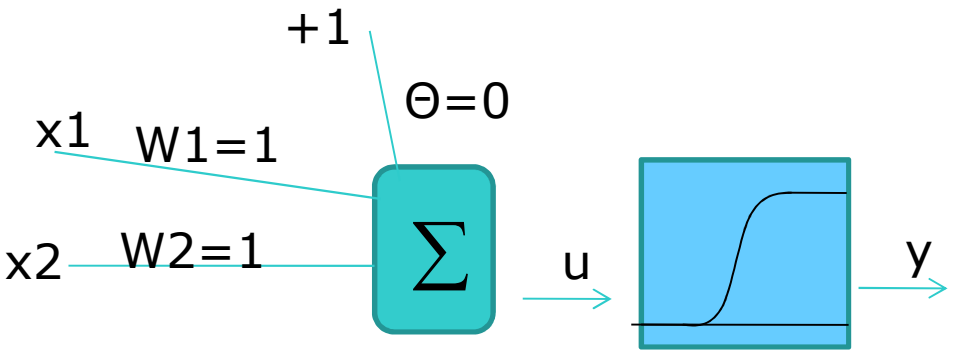
Vimos o que acontece se alterarmos os pesos w1, w2 (w2 > w1)

Θ=0, w1=1, w2=3



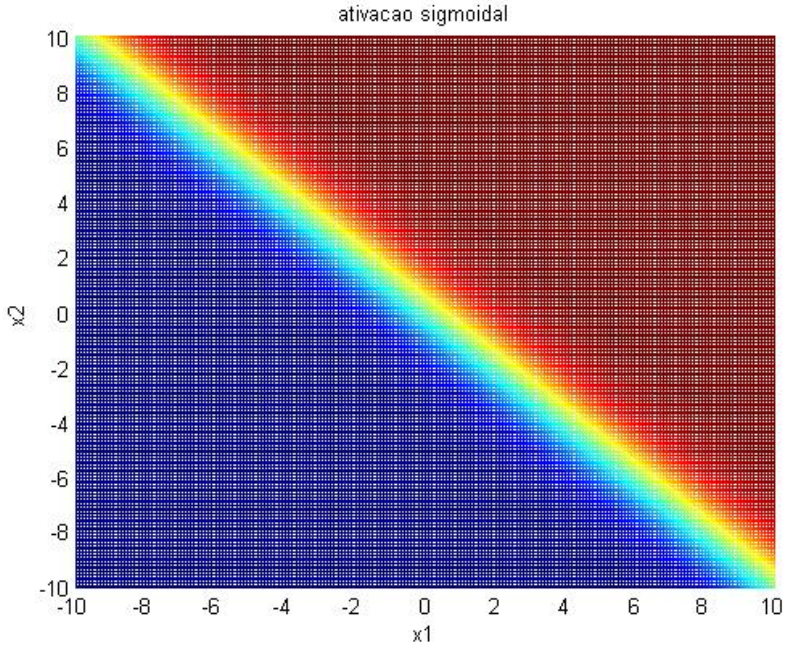


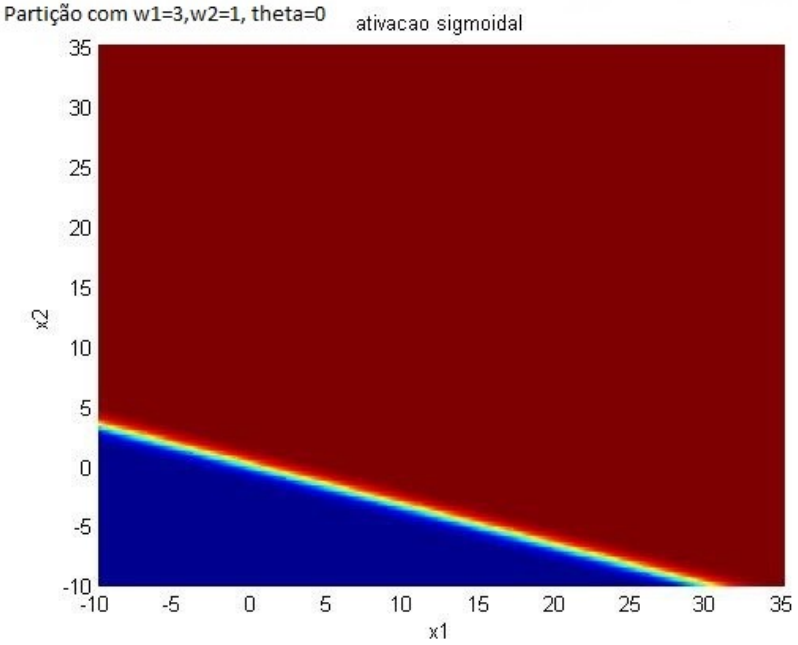
Neurônio na MLP: Abstração na partição



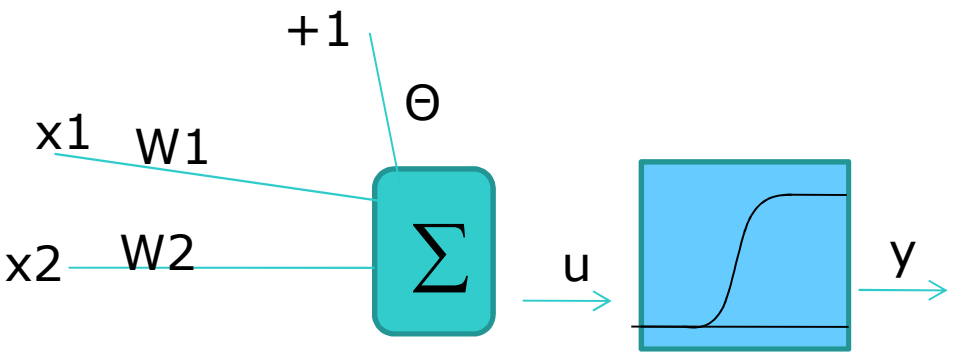
Vimos o que acontece se alterarmos os pesos w1, w2 (w1 > w2)

Θ=0, W1=3, w2=1



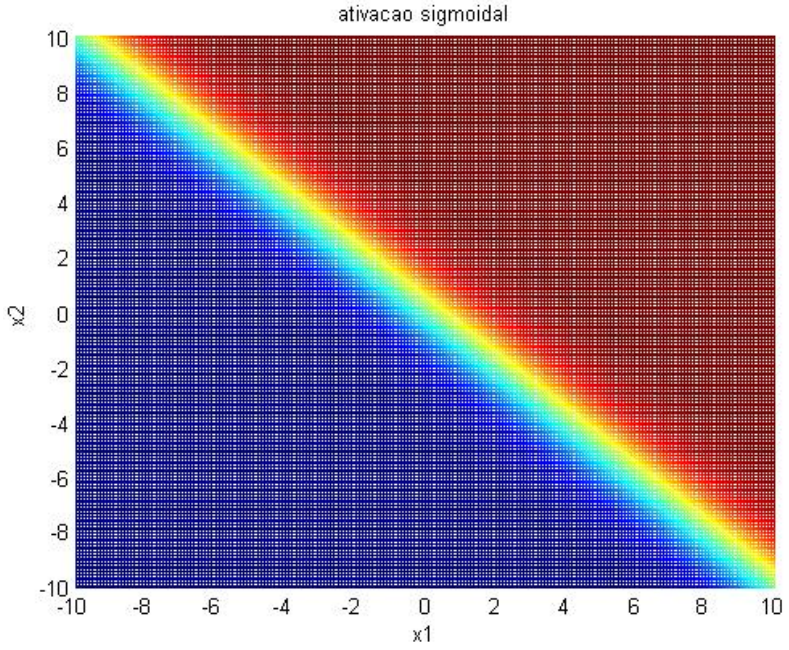


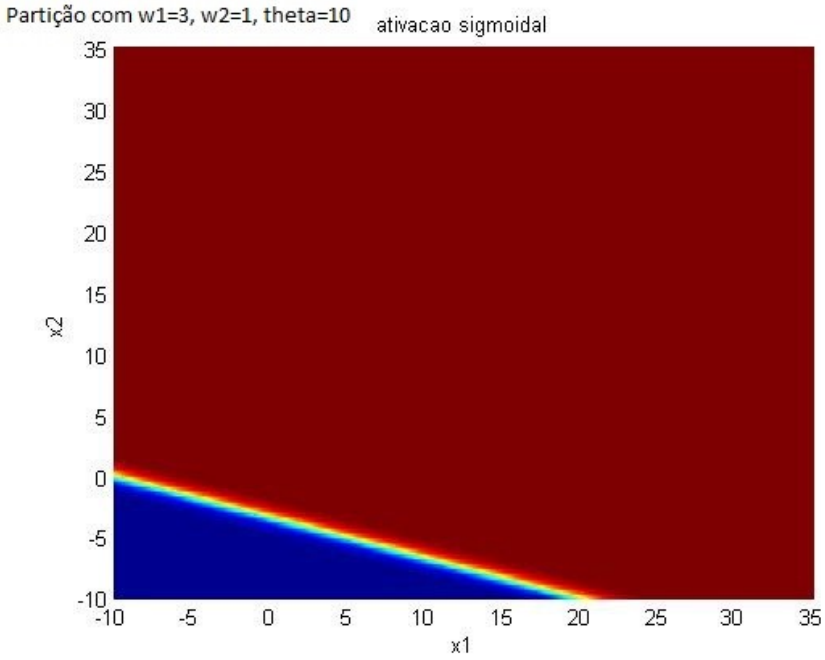
Neurônio na MLP: Abstração na partição



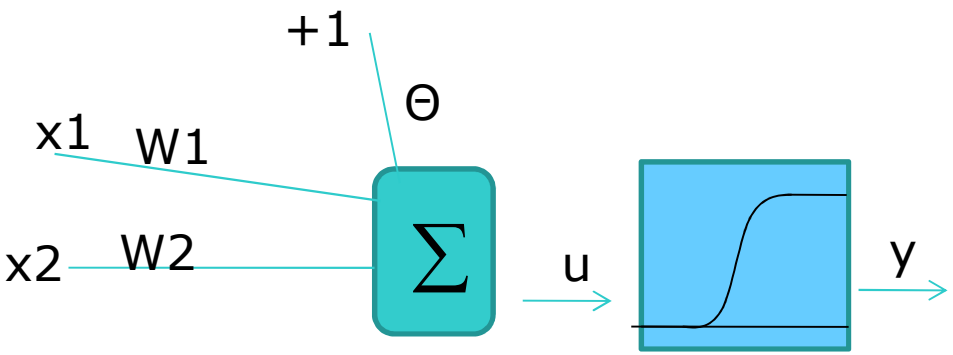
Vimos o que acontece se alterarmos o limiar Θ, e os pesos w1, w2

Θ=10, W1=3, w2=1



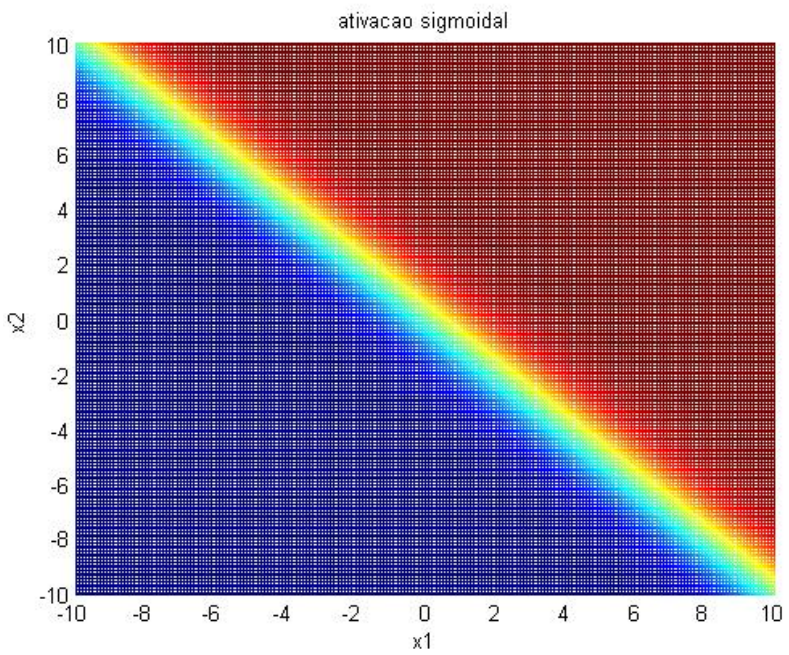


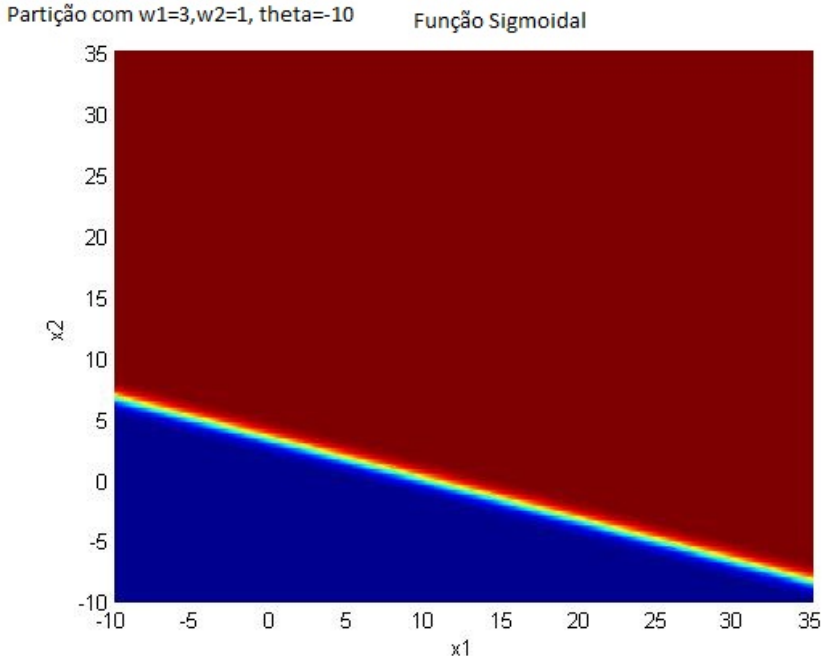
Neurônio na MLP: Abstração na partição



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

Θ=-10, W1=3, w2=1

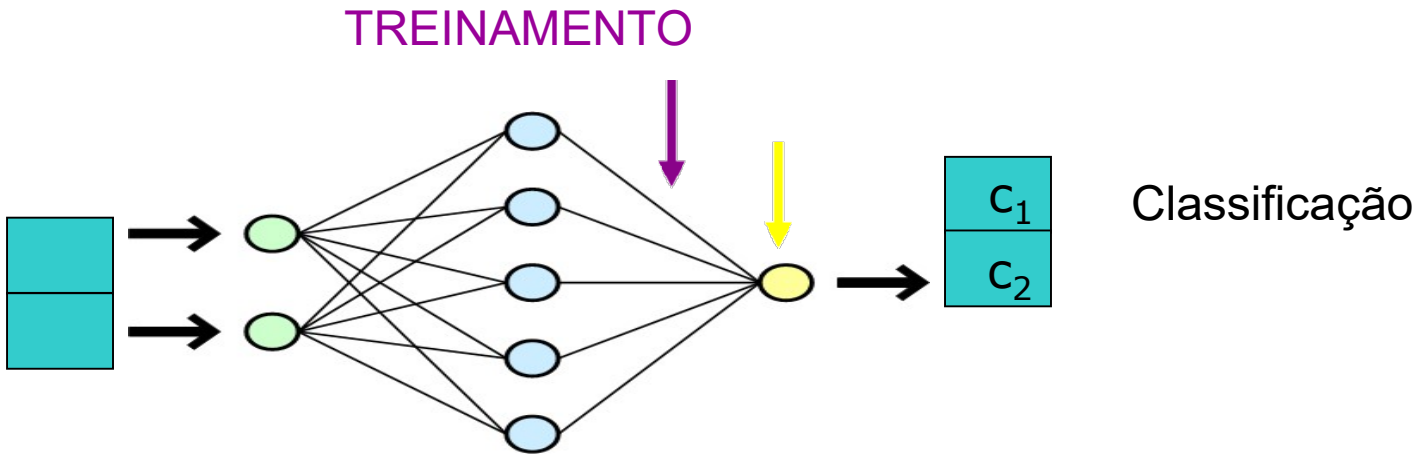


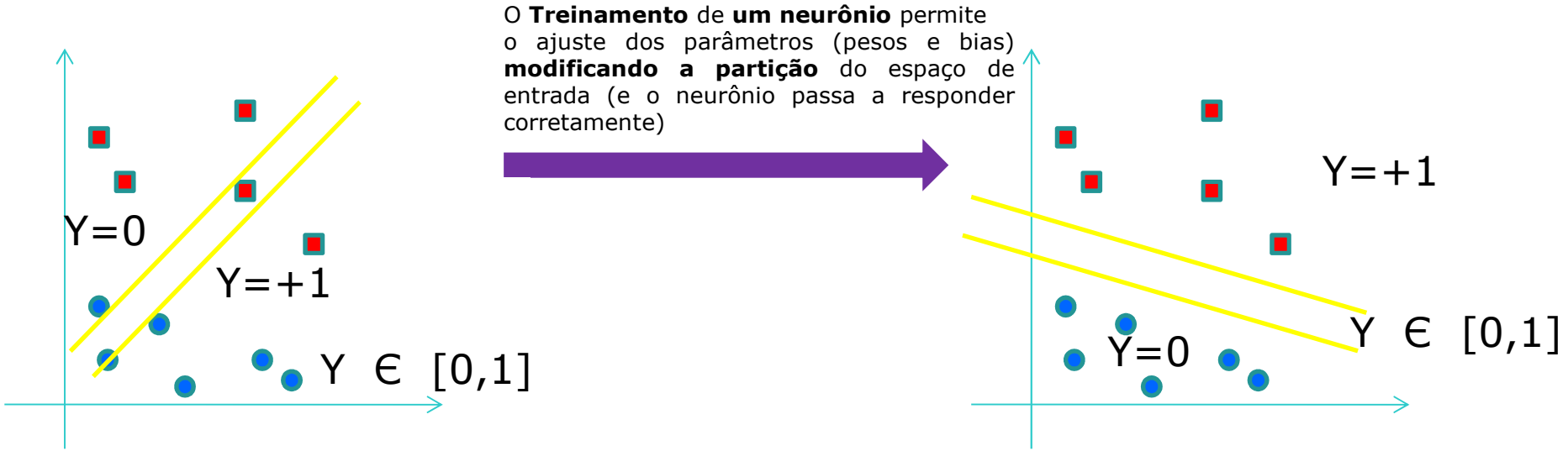


Multi-Layer Perceptron

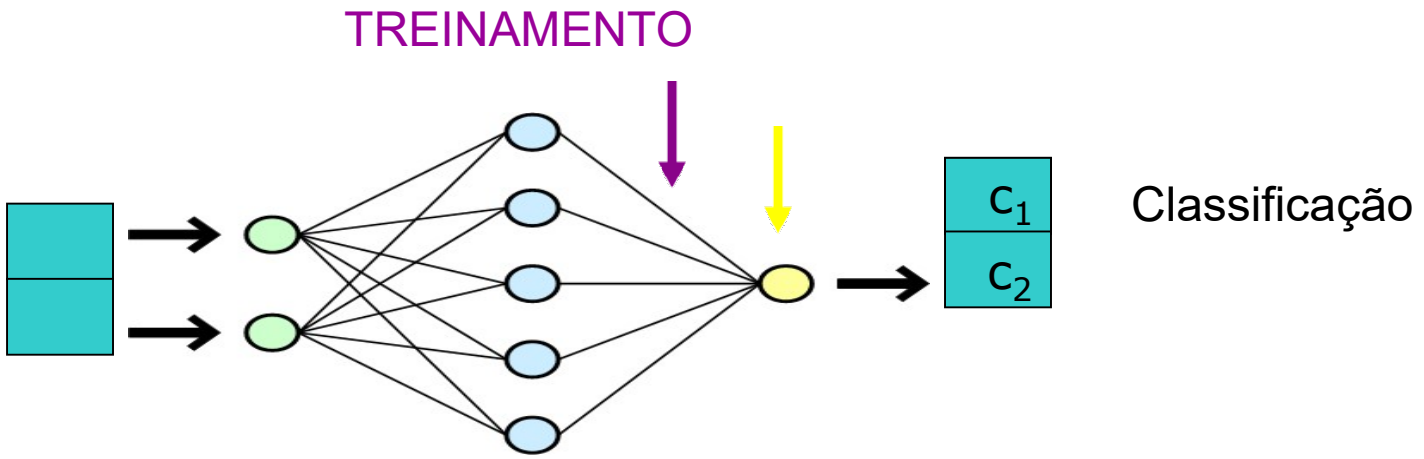
- Arquitetura da rede
- Modelo do neurônio: função de ativação
- **Treinamento x Aplicação**

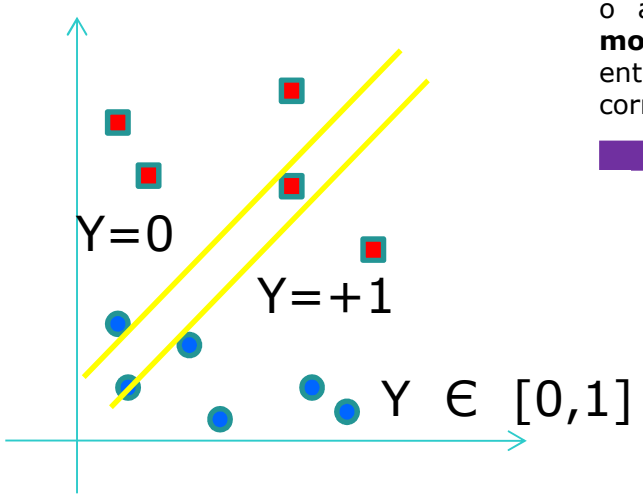
MLP: Treinamento e Classificação



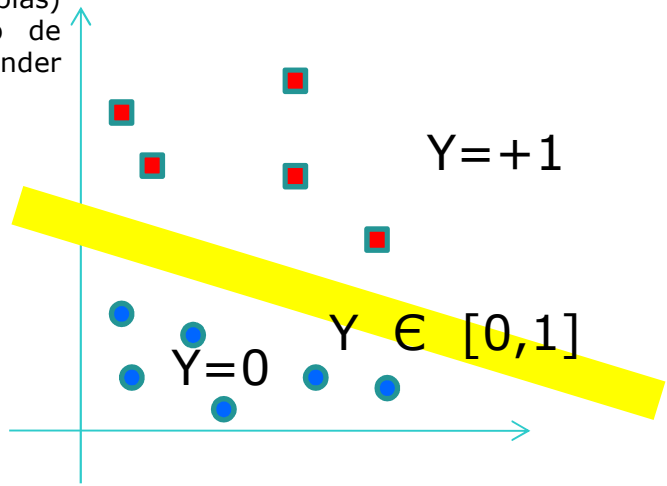


MLP: Treinamento e Classificãção

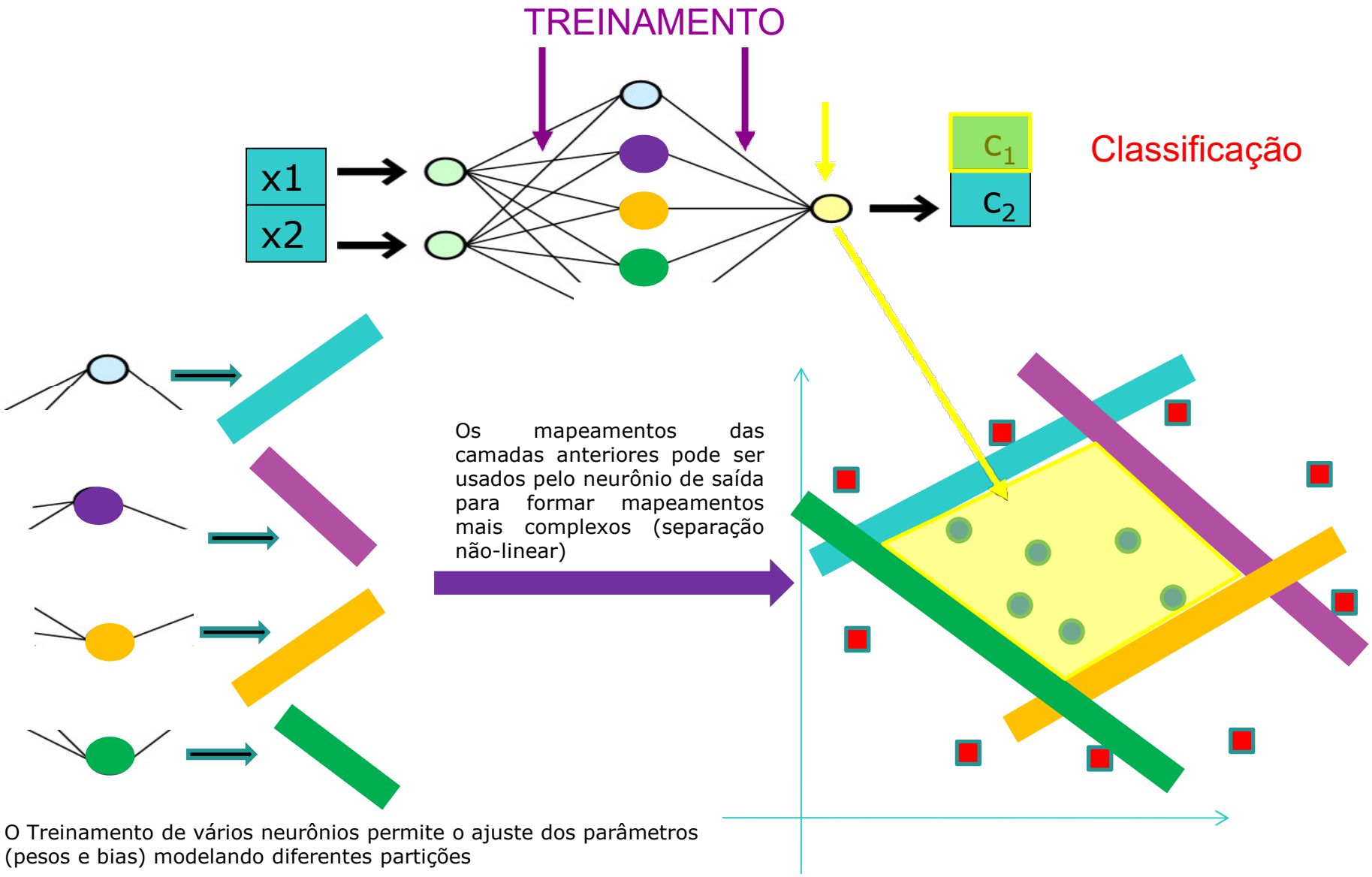




O **Treinamento** de **um neurônio** permite o ajuste dos parâmetros (pesos e bias) **modificando a partição** do espaço de entrada (e o neurônio passa a responder corretamente)

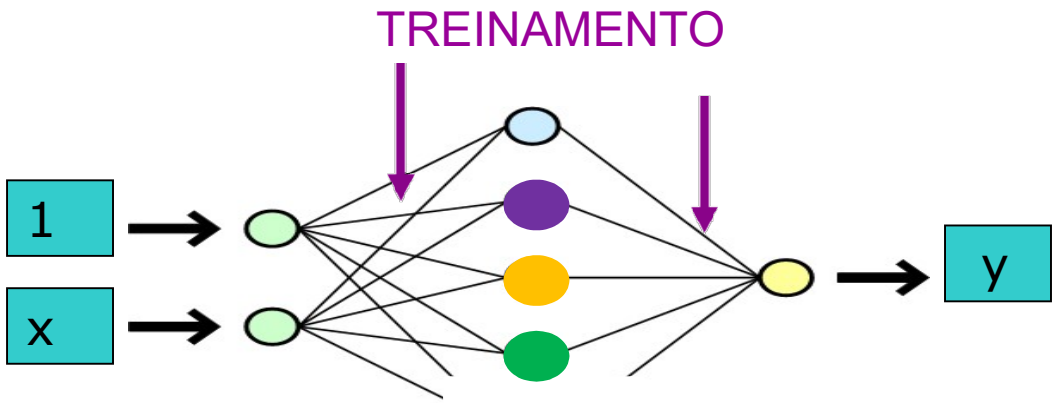


MLP: Treinamento e Classificação



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

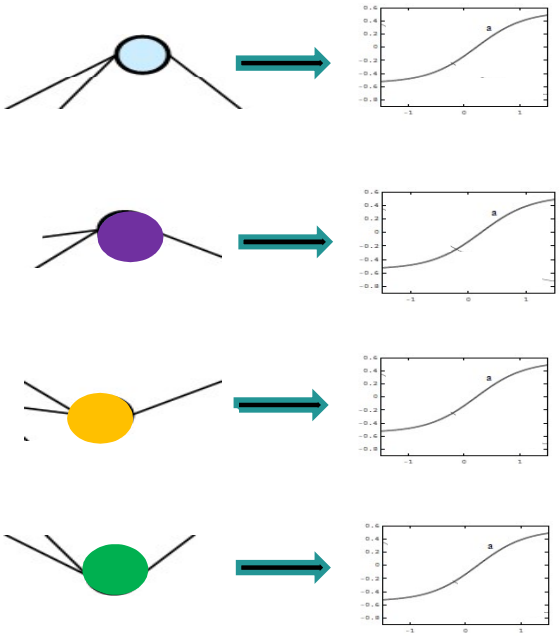
MLP: Treinamento e Regressão



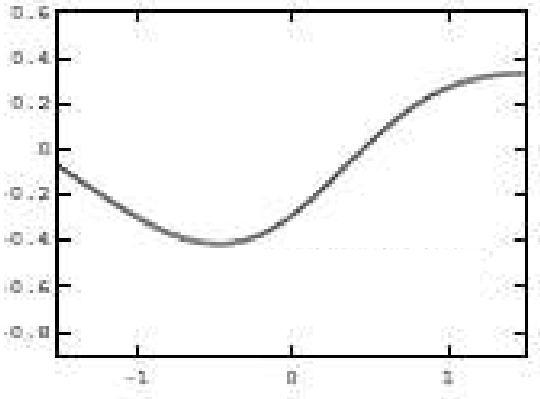
Aproximação

De Funções

(**unidimensionais**)

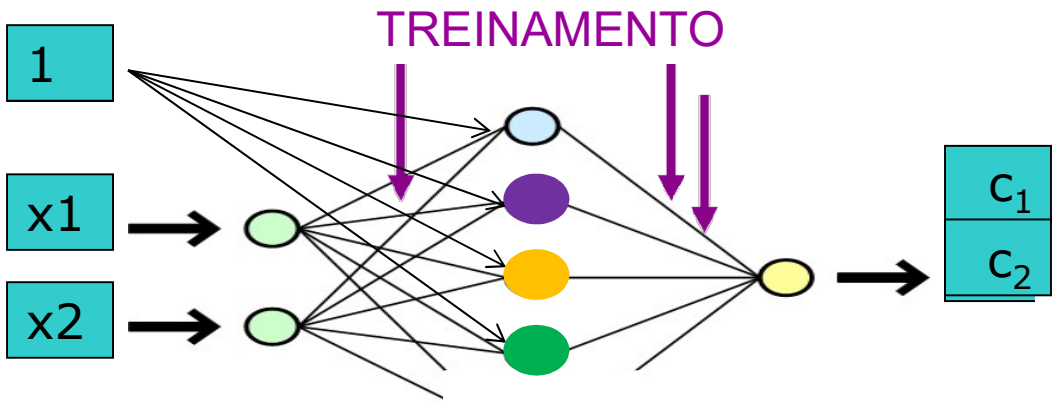


Os mapeamentos das camadas anteriores podem ser usados pelo neurônio de saída para formar **mapeamentos mais complexos** (funções aproximadas complexas)



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

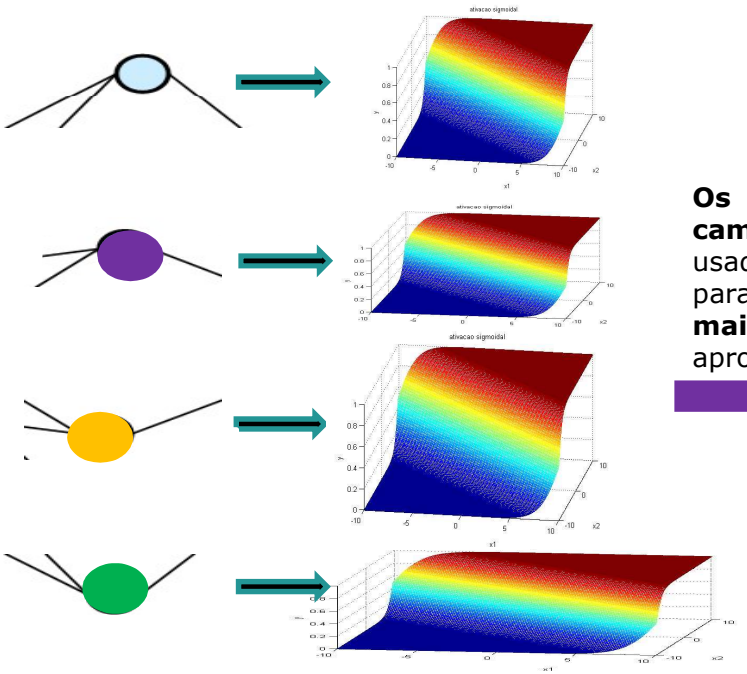
MLP: Treinamento e Regressão



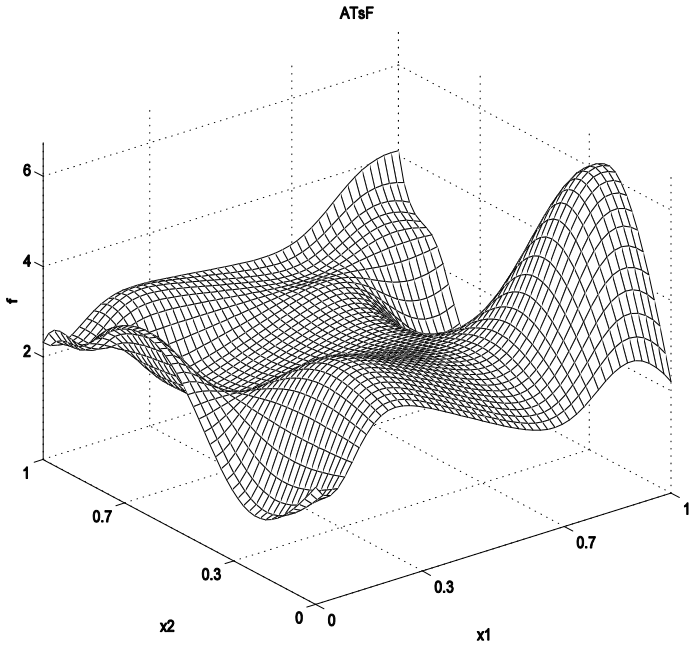
Aproximação

De Funções

(**Multidimensionais**)

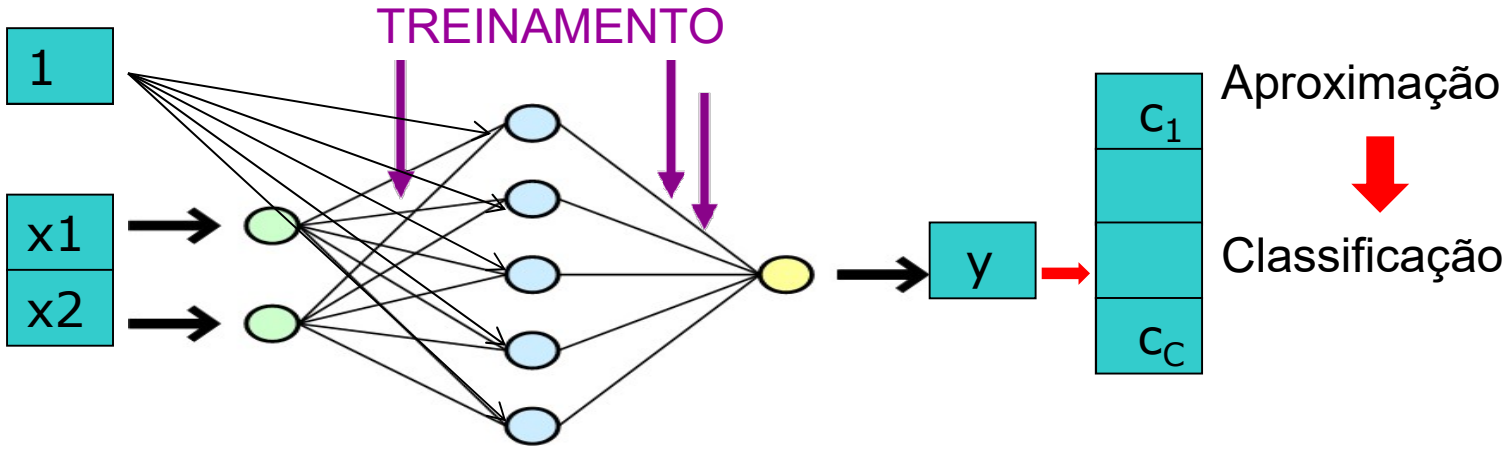


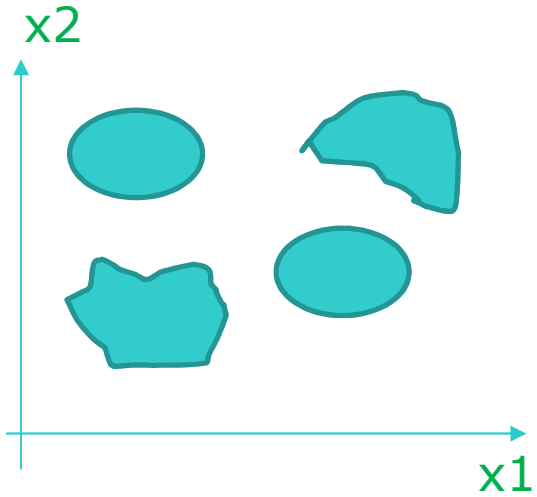
Os mapeamentos das camadas anteriores podem ser usados pelo neurônio de saída para formar **mapeamentos mais complexos** (funções aproximadas complexas)



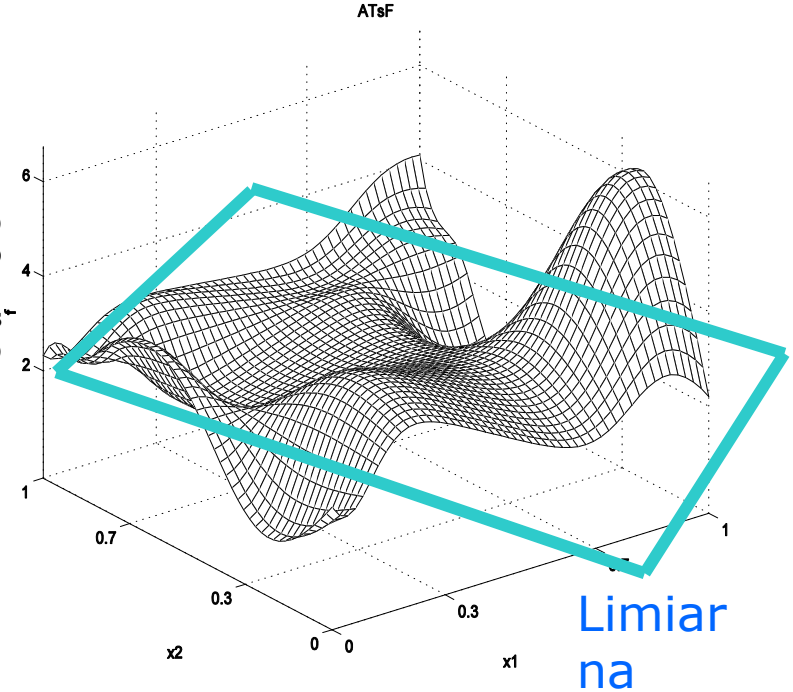
O Treinamento de vários neurônios permite o ajuste dos parâmetros (pesos e bias) modelando diferentes saídas no nível hidden

MLP: Treinamento e Regressao->Classificação



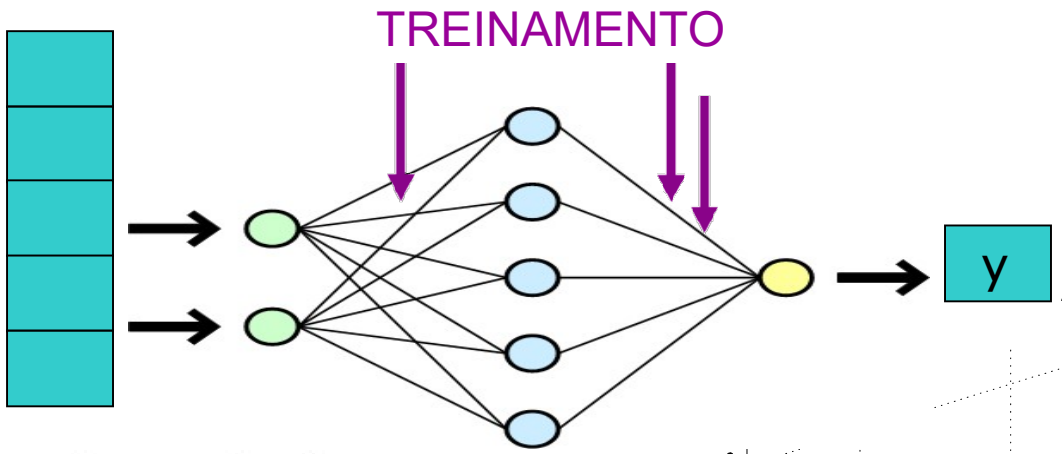


O mapeamento aproximado pode ser **limiarizado** (corte num nivel) de forma a produzir **diferentes grupos** (classes) no espaço de entrada

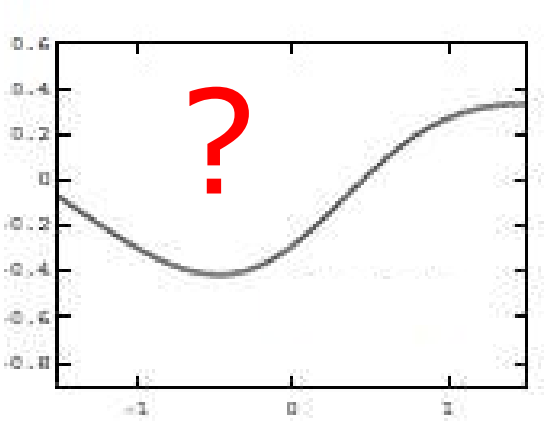


Limiar na saida

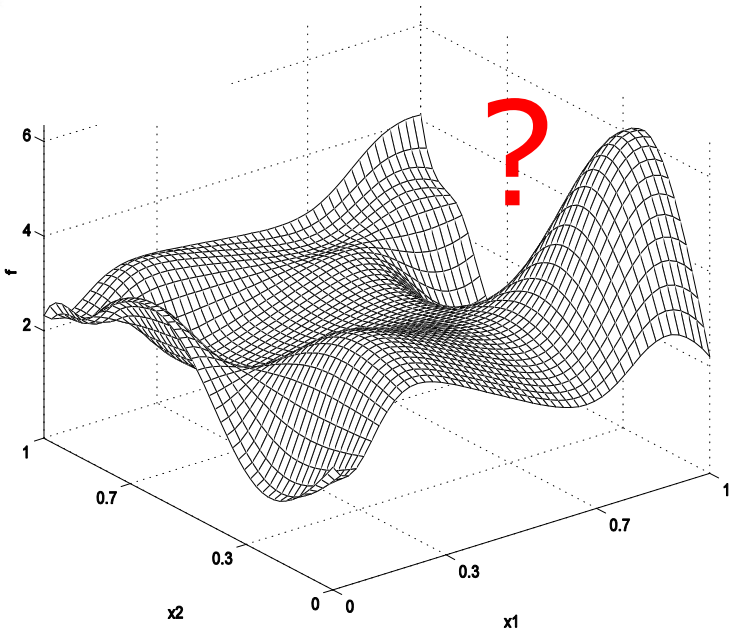
MLP: Treinamento e Aplicação



Aproximação de Funções



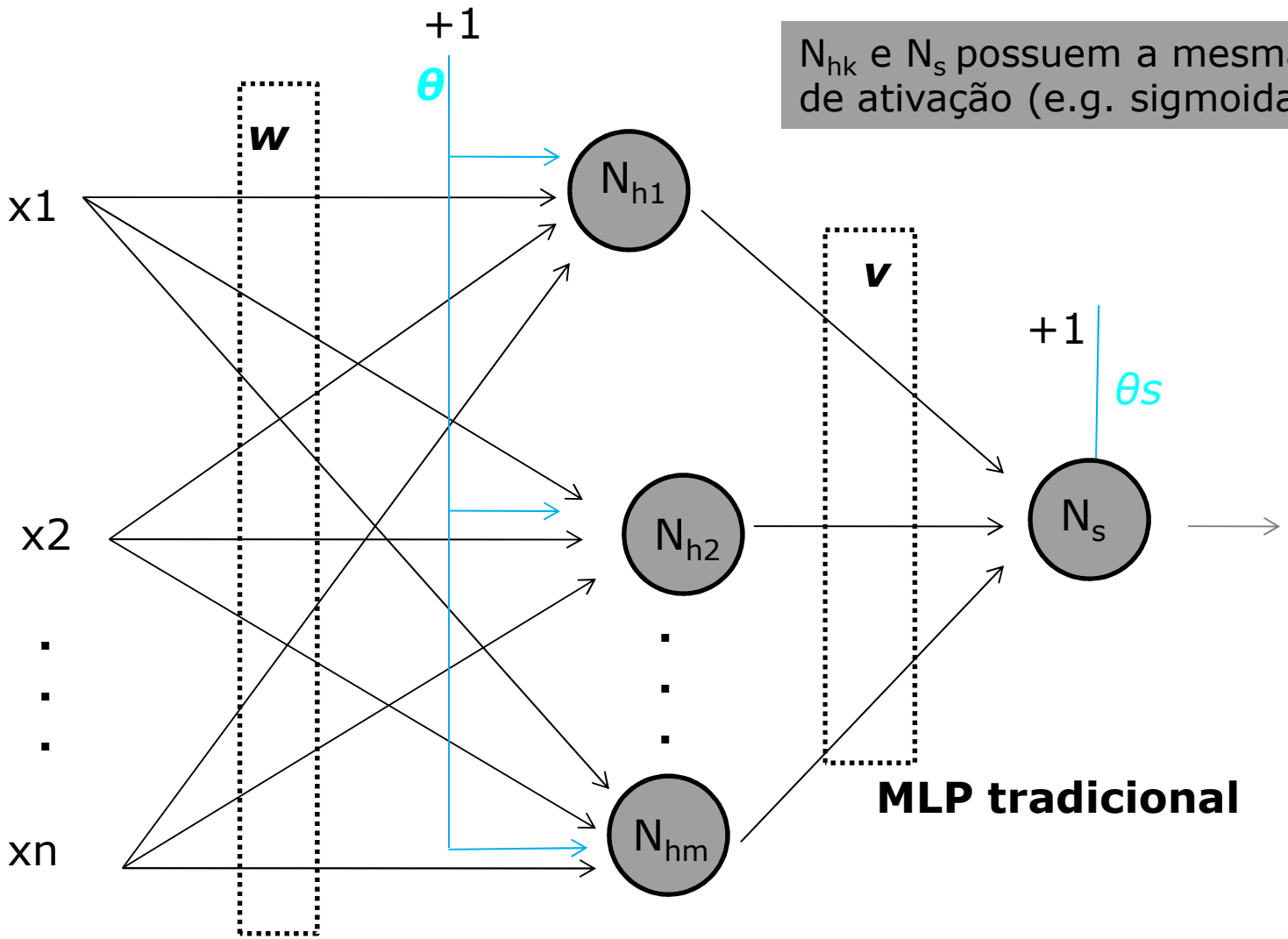
(1 entrada e 1 saída no exemplo acima)



2 entradas 1 saída como no exemplo ao lado

Entendendo a interação entre as camadas

Multi-Layer Perceptron tradicional

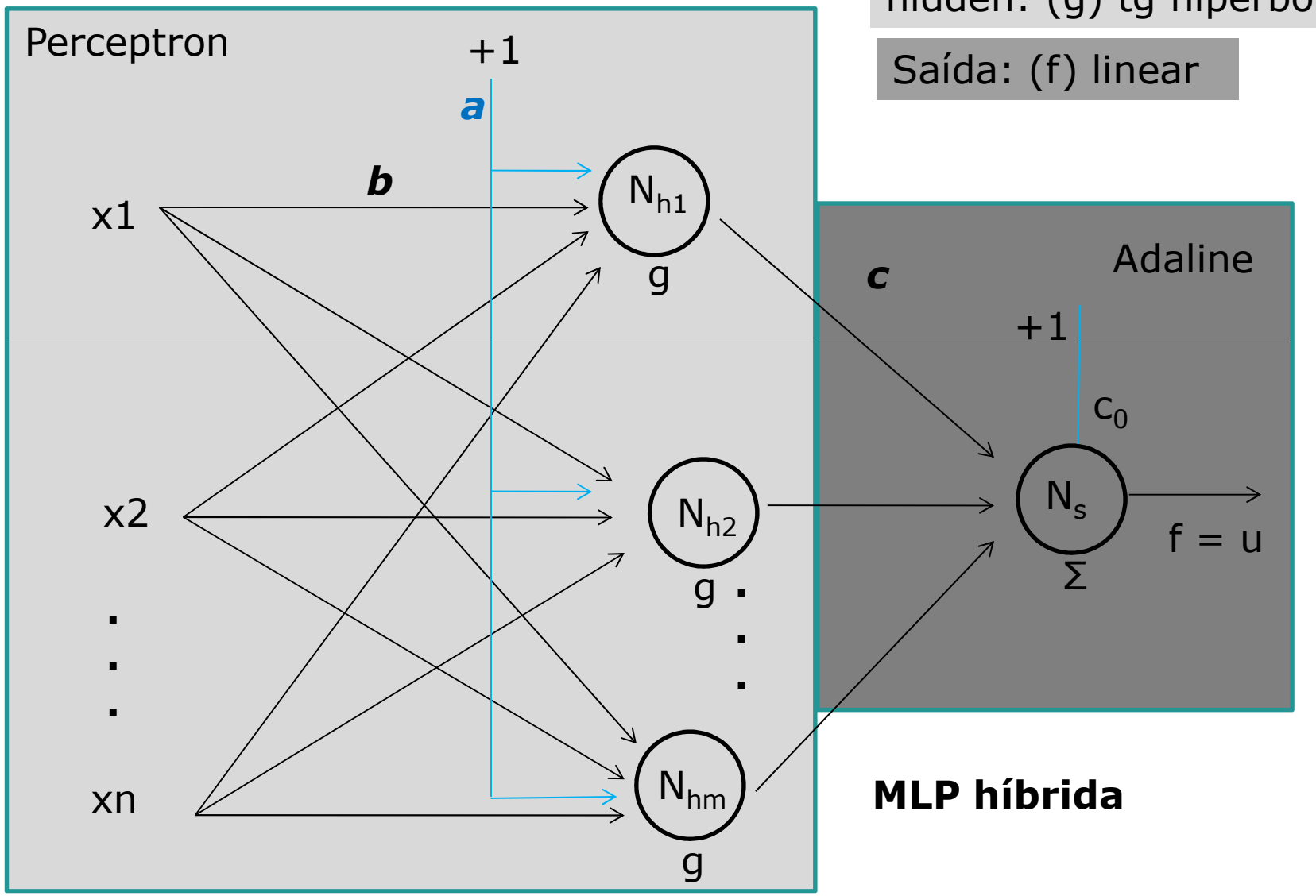


N_{hk} e N_s possuem a mesma função de ativação (e.g. sigmoidal)

MLP híbrida para aproximação de funções

hidden: (g) tg hiperbólica

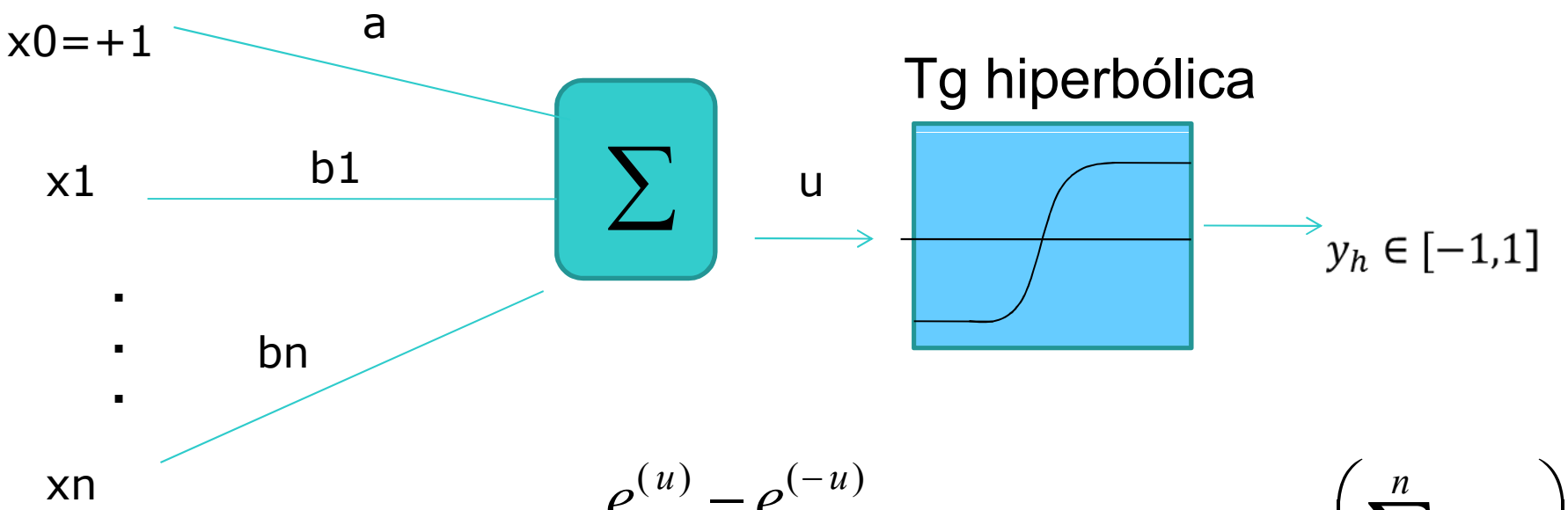
Saída: (f) linear



MLP híbrida

MLP híbrida: Tg Hiperbólica (N_h)

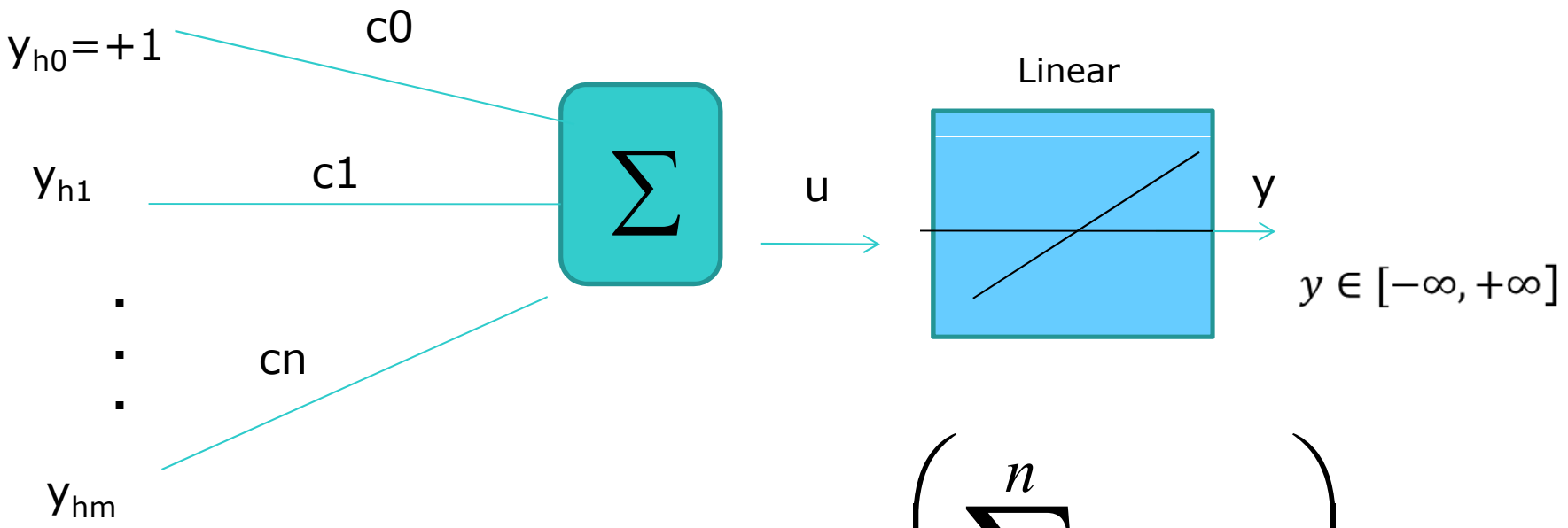
f = Tangente Hiperbólica (hidden)



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

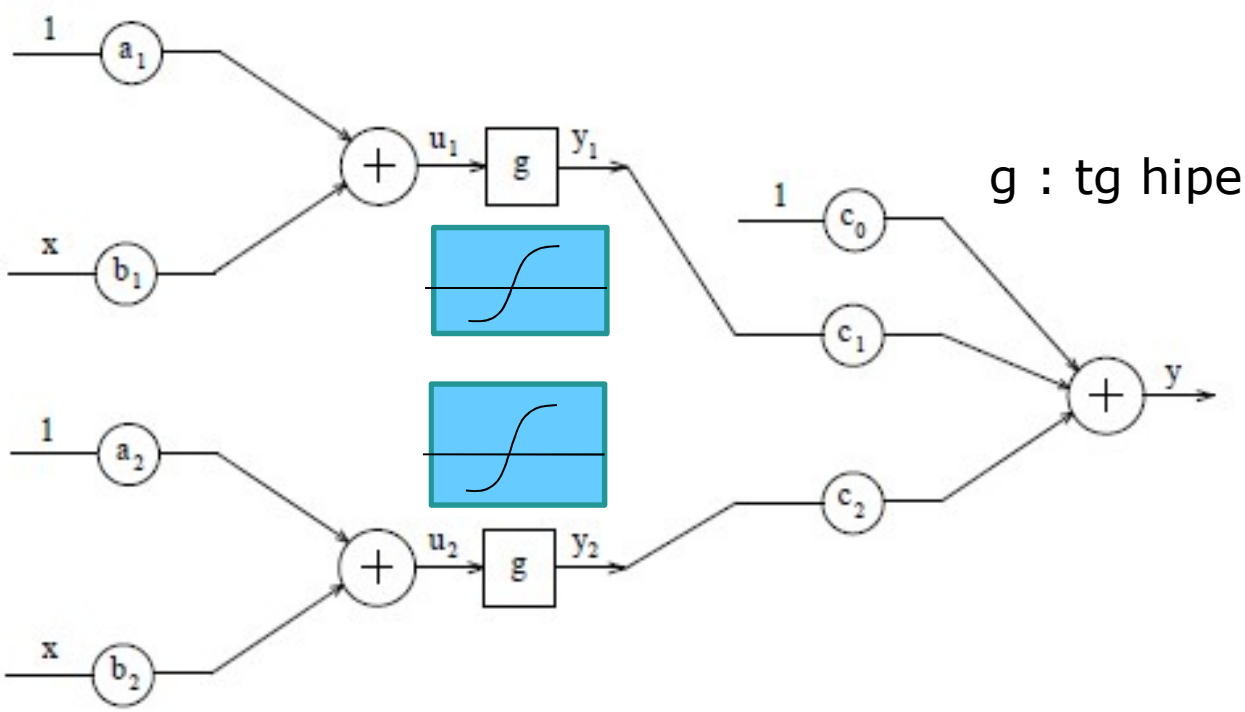
MLP híbrida: Linear(N_s)

f = função Linear (saída)

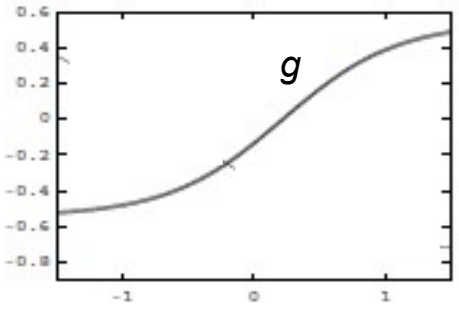


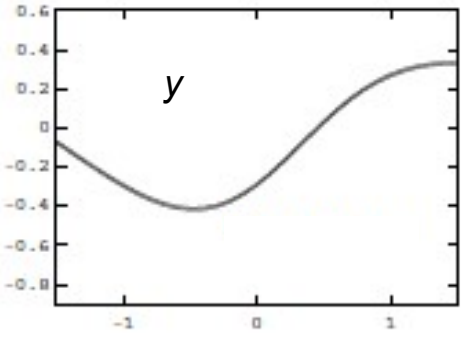
$$y = \left(\sum_{i=0}^{n} c_i y_{hi} \right) = u$$

MLP Híbrida (Perceptron + Adaline)



g : tg hiperbólica





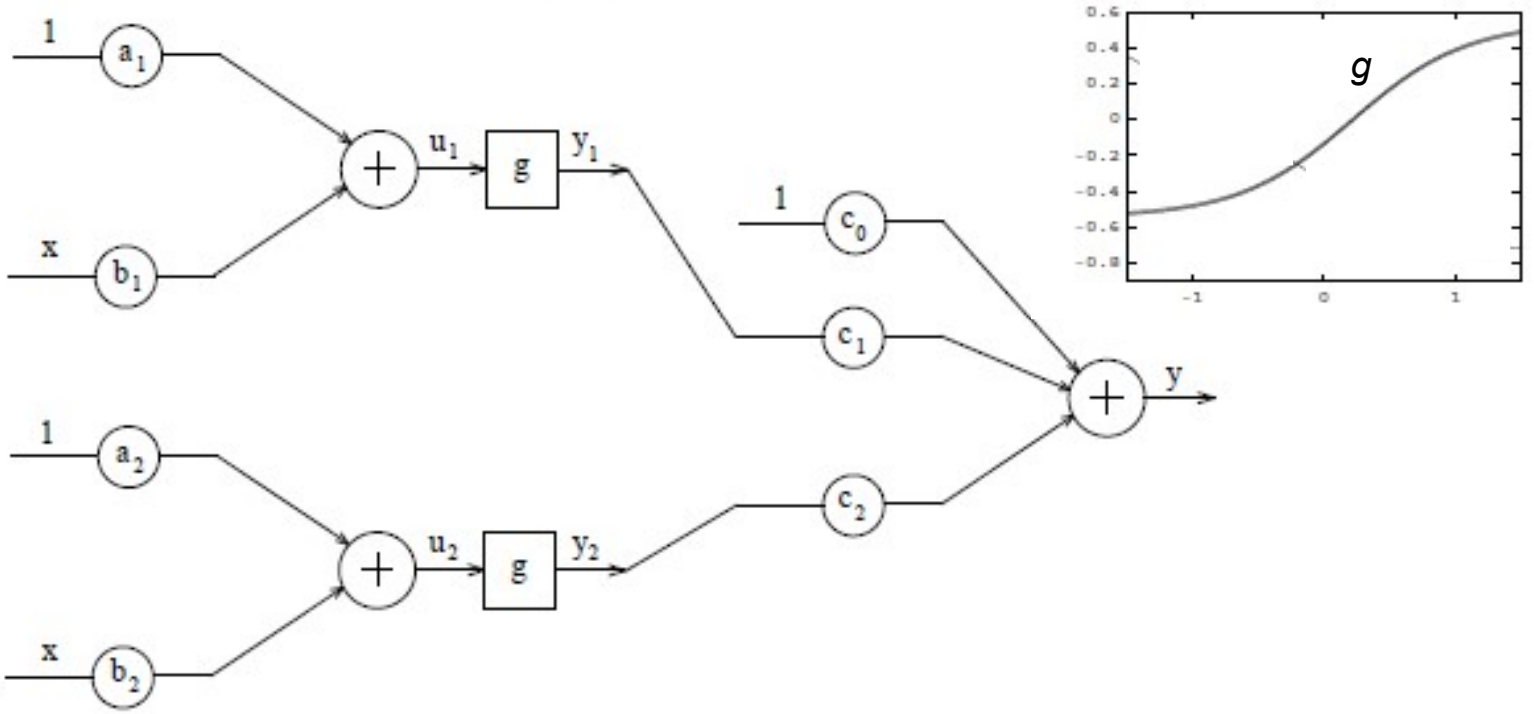
Aprendizado Conexionista (MLP híbrida)

O papel dos pesos

Aprendizado Conexionista (MLP híbrida)

O papel dos pesos

g : tg hiperbólica



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

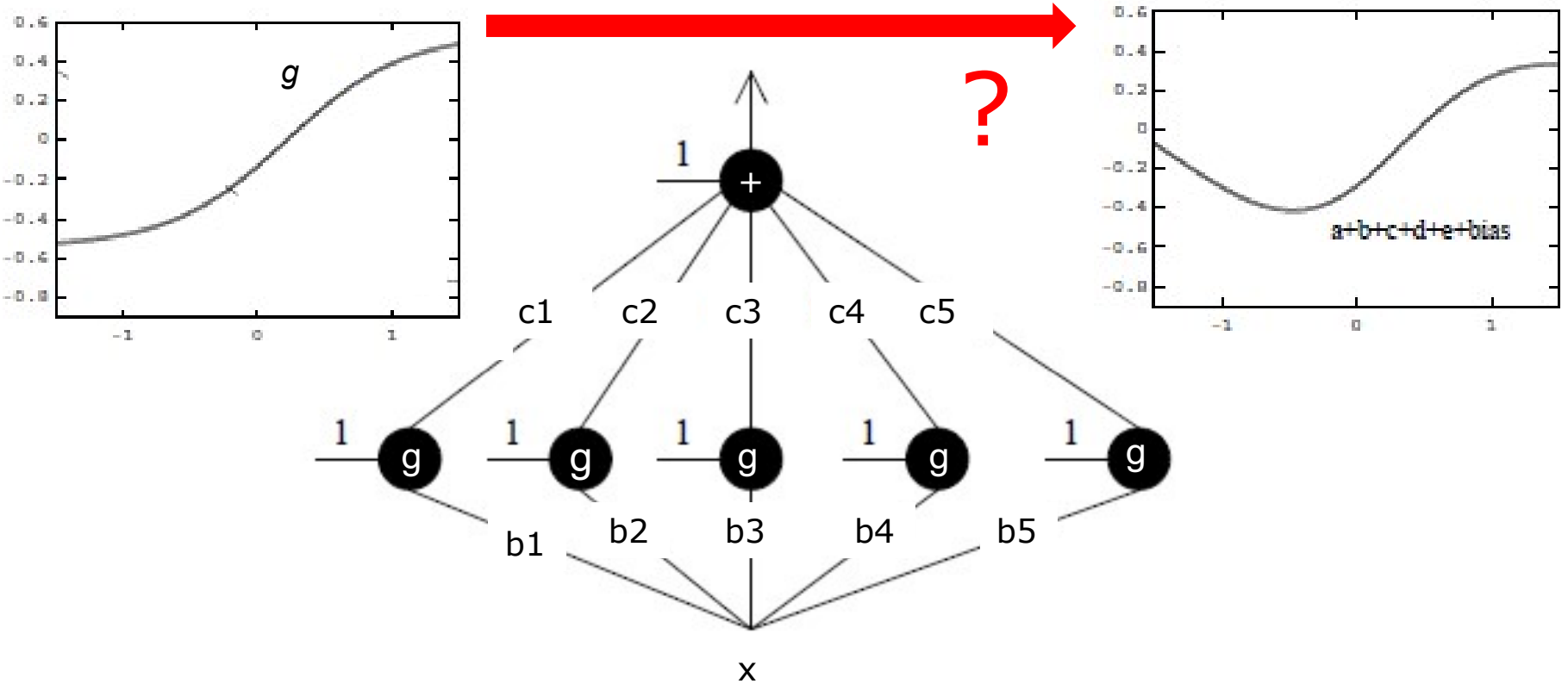
Aprendizado Conexionista: pesos

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

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

Aprendizado Conexionista: pesos

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



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

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

Aprendizado Conexionista: pesos

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

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

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

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

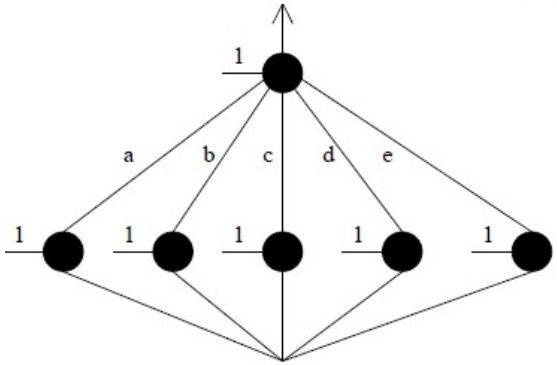
$$+ \underbrace{c_0}_{bias}$$

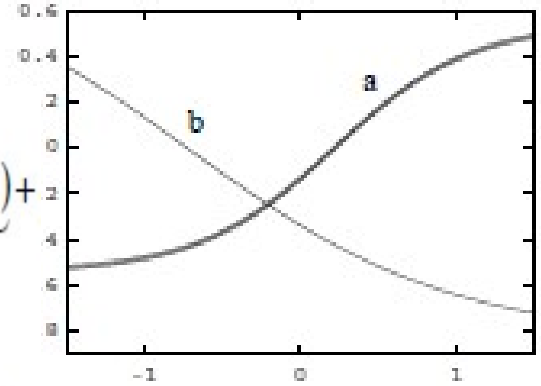
Aprendizado Conexionista: pesos

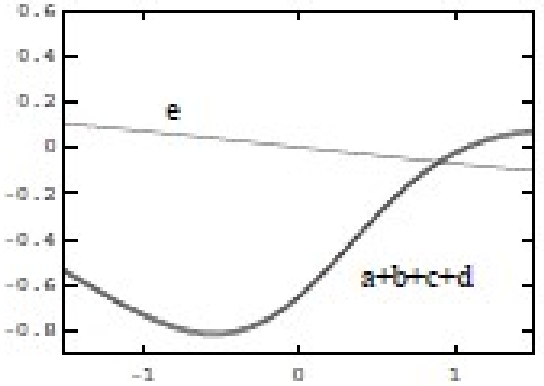
$$f(\mathbf{w}) = \underbrace{c_1 g(b_1 x + a_1)}_{a} + \underbrace{c_2 g(b_2 x + a_2)}_{b} +$$

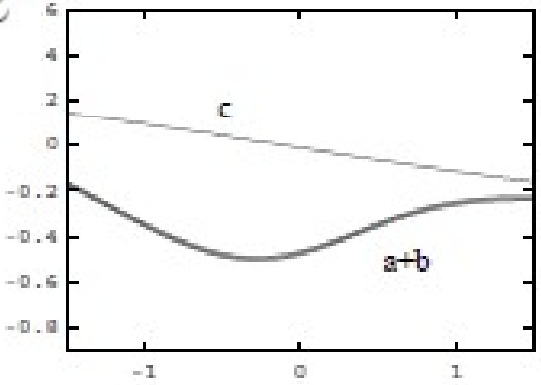
$$+ \underbrace{c_3 g(b_3 x + a_3)}_{c} + \underbrace{c_4 g(b_4 x + a_4)}_{d} +$$

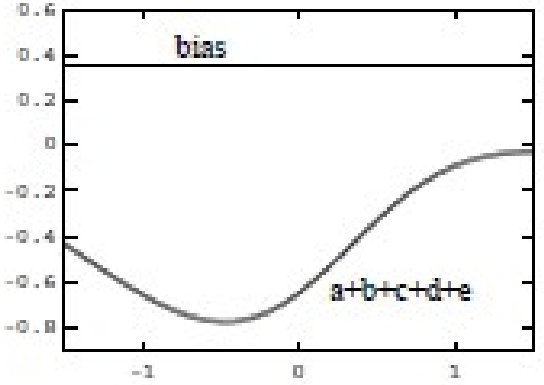
$$+ \underbrace{c_5 g(b_5 x + a_5)}_{e} + \underbrace{c_0}_{bias}$$

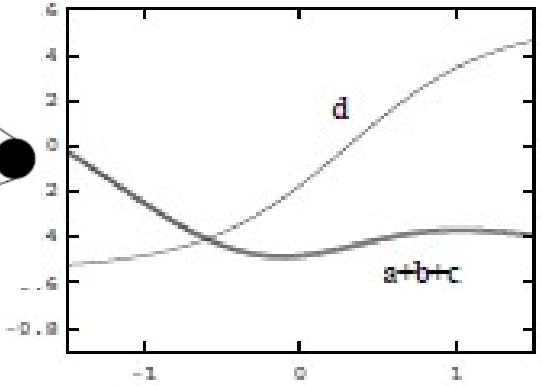


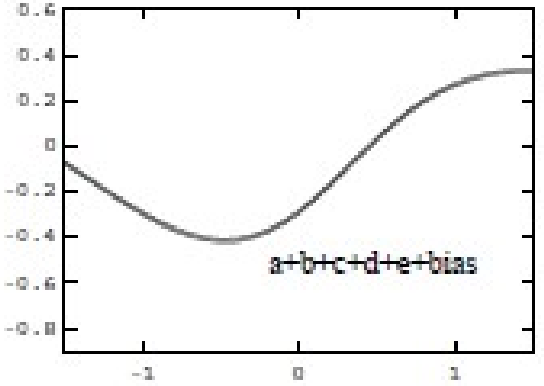




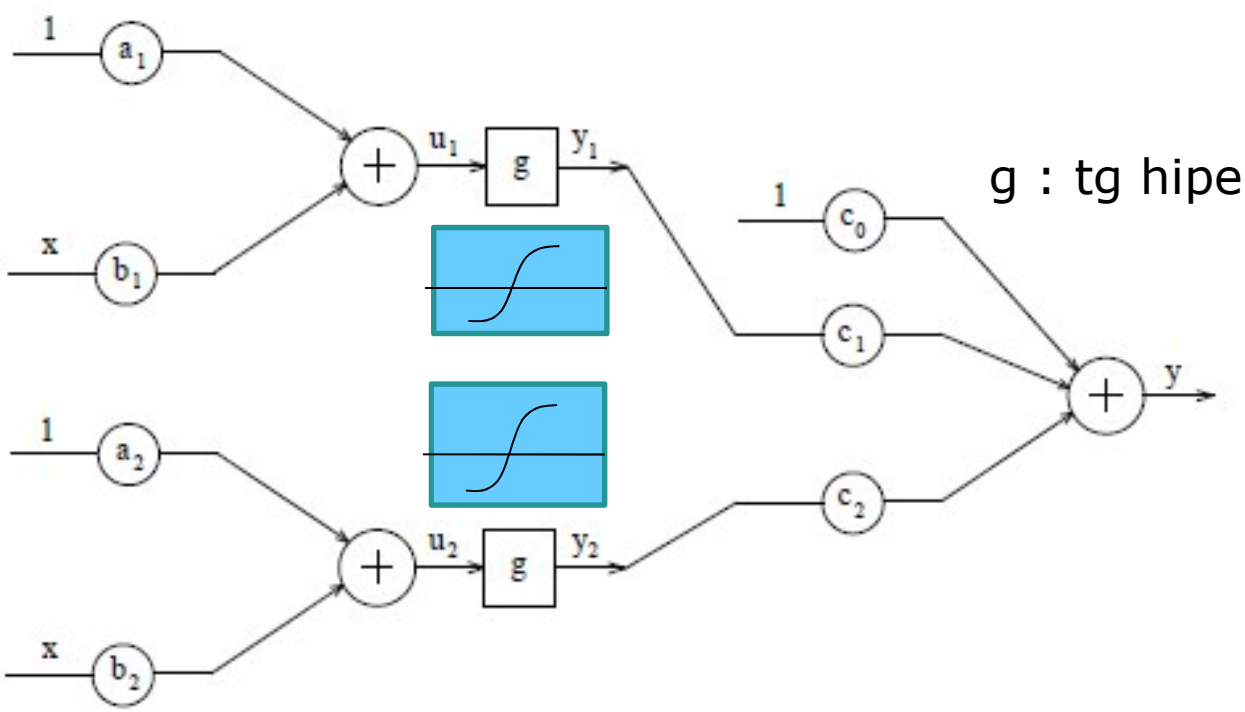




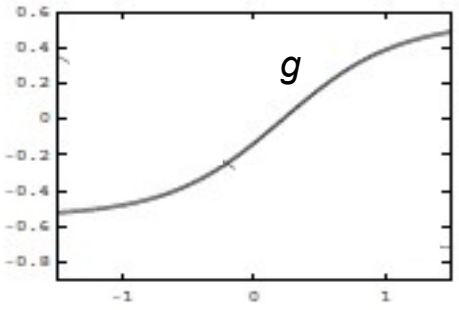


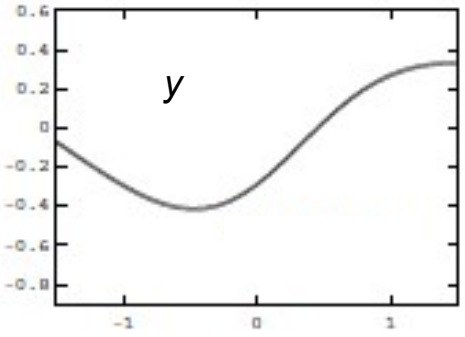


MLP Híbrida (Perceptron + Adaline)

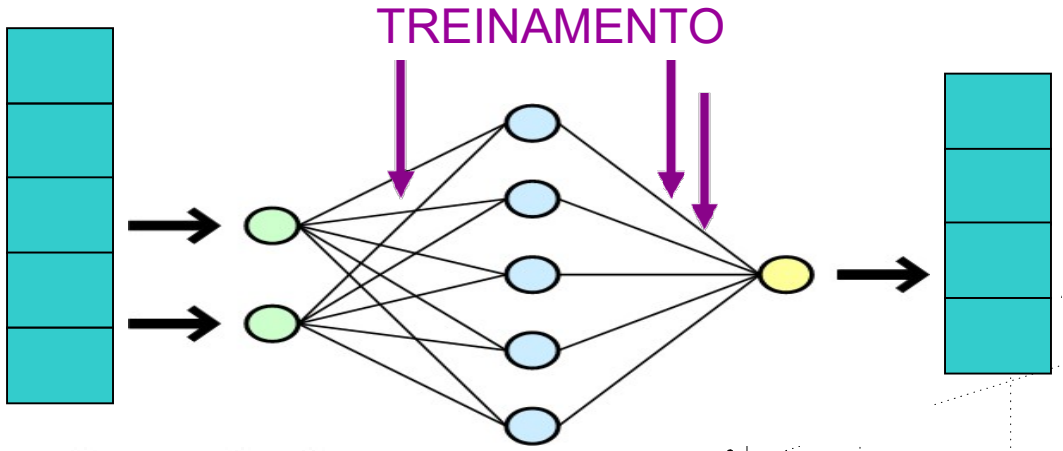


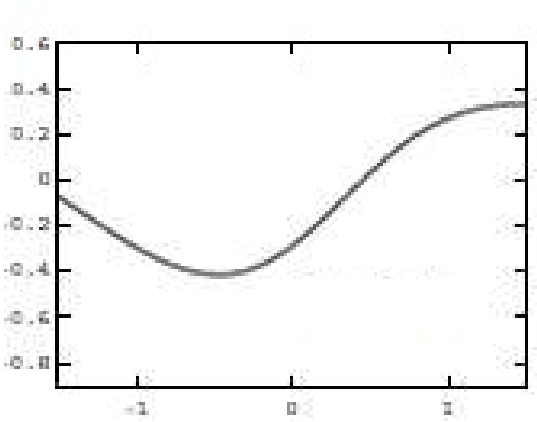
g : tg hiperbólica



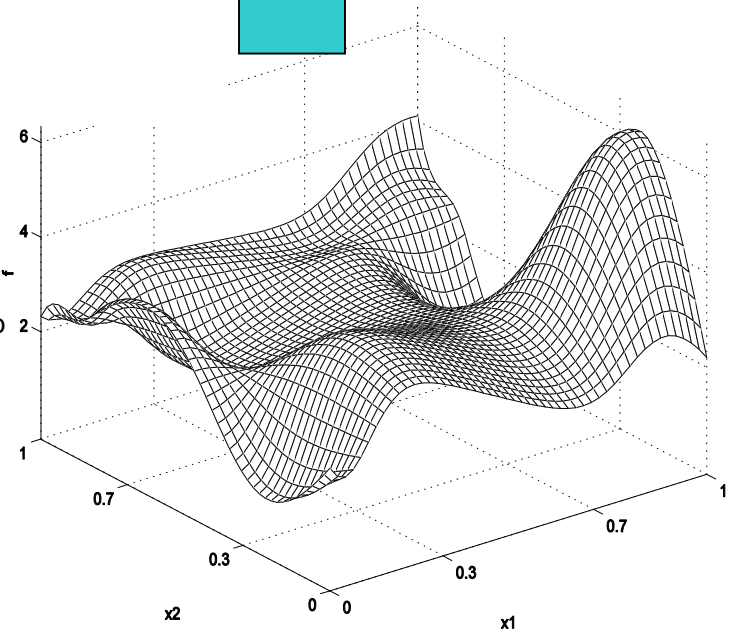


MLP Híbrida (Perceptron + Adaline)





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