



Rede neural MultiLayer Perceptron (MLP)

Silvia Moraes

Redes Neurais Artificiais são sistemas computacionais distribuídos compostos de unidades simples (neurônios), densamente interconectados.

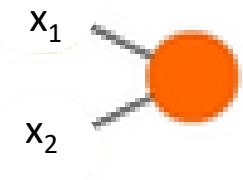


Redes Neurais Artificiais podem aplicadas em várias tarefas, em diferentes paradigmas de aprendizagem a depender da sua arquitetura:

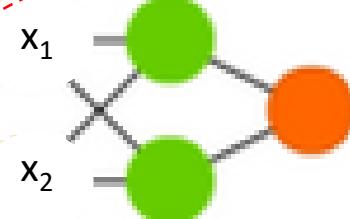
- Classificação de dados, de texto, de imagem, ...
- Reconhecimento de objetos, faces, palavras (escritas ou faladas)...
- Regressão de dados: preços futuros de taxas de câmbio ou ações, previsão de vendas, previsão de demandas, ...
- Detecção de fraudes e anomalias
- Recomendação de produtos
- Agrupamento de dados, de objetos, de pessoas, ...
- Sumarização de dados
- Similaridade de dados
- ...

Redes Feed Forward

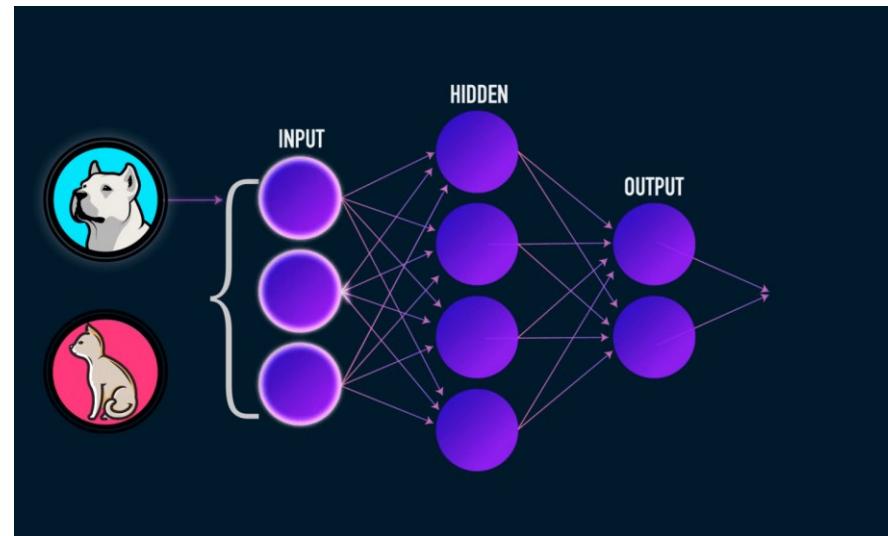
Tarefas Supervisionadas: **classificação** o e **regressão**



Perceptron



MultiLayer Perceptron

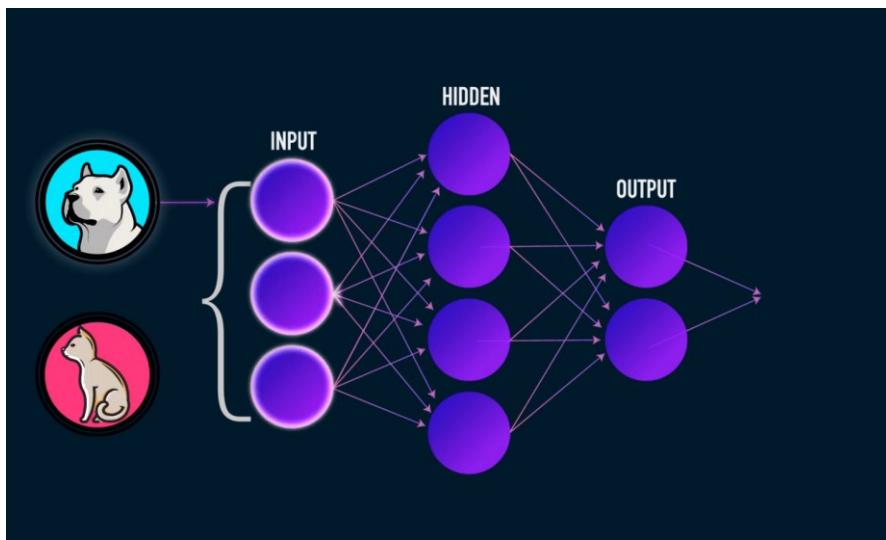


<https://sites.icmc.usp.br/andre/research/neural/>

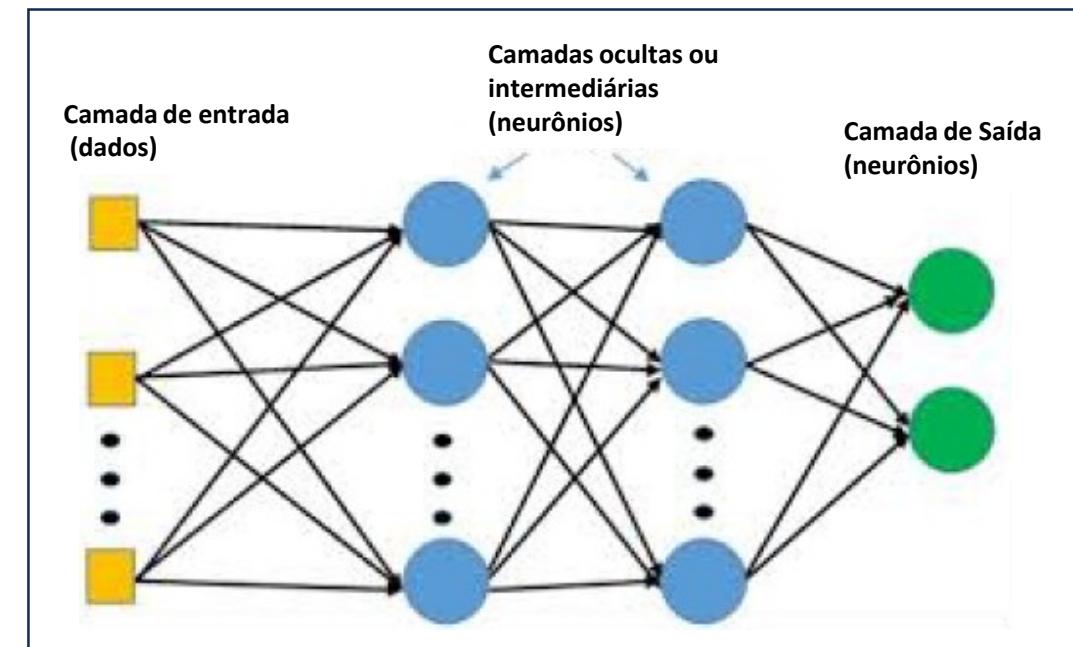
Propagação: fluxo do sinal sempre para frente

Rede MultiLayer Perceptron

- São **redes perceptron de múltiplas camadas**, contendo uma ou mais camadas ocultas.
- Resolve **problemas não linearmente separáveis**.
- É considerada uma **Shallow Learning** (em oposição à **Deep Learning**) especialmente quando possuem apenas uma camada oculta.



<https://sites.icmc.usp.br/andre/research/neural/>



Arquitetura e Topologia

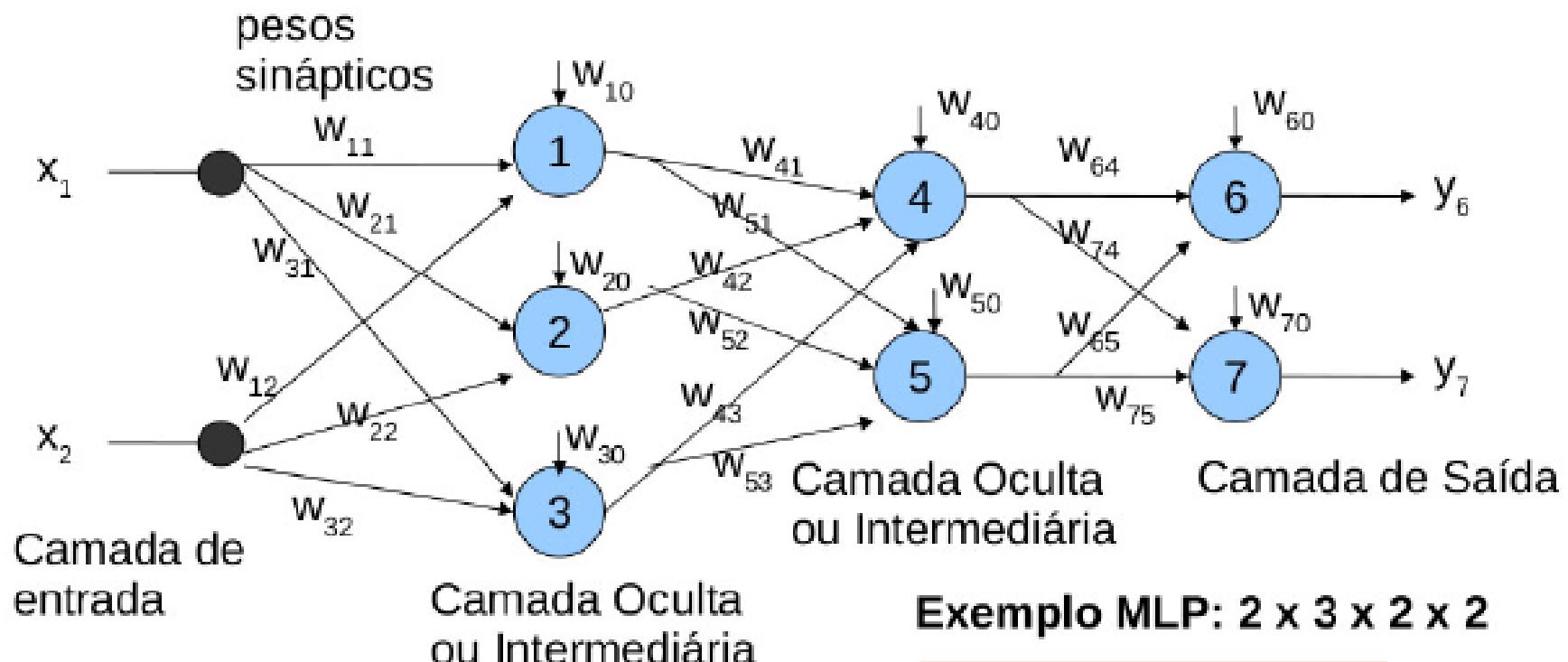


Rede MultiLayer Perceptron

- Arquitetura e Topologia muitas vezes são usados como sinônimos.
- Eles realmente são fortemente relacionados.
- **Arquitetura corresponde ao tipo de rede e sua topologia.**
- **Topologia descreve o “desenho da rede”:** organização dos seus componentes
 - Número de camadas
 - Número de neurônios por camadas
 - Conexões

Rede MultiLayer Perceptron

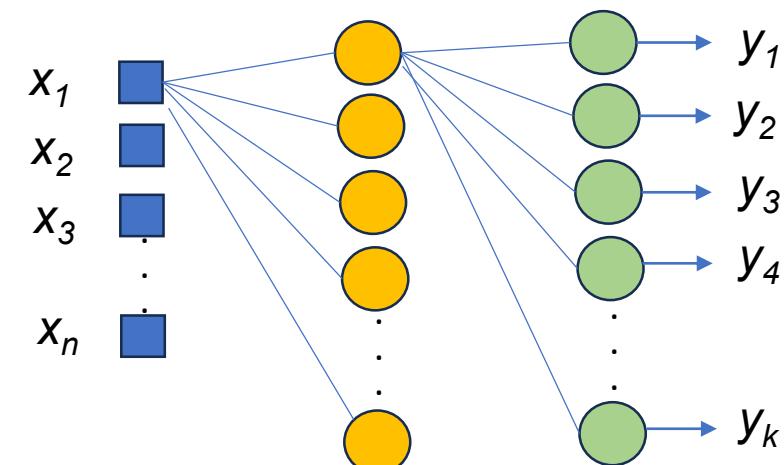
- Exemplo de Topologia :



Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

- Camada de entrada: corresponde à camada de dados
- Camada oculta: de neurônios
- Camada de saída: de neurônios

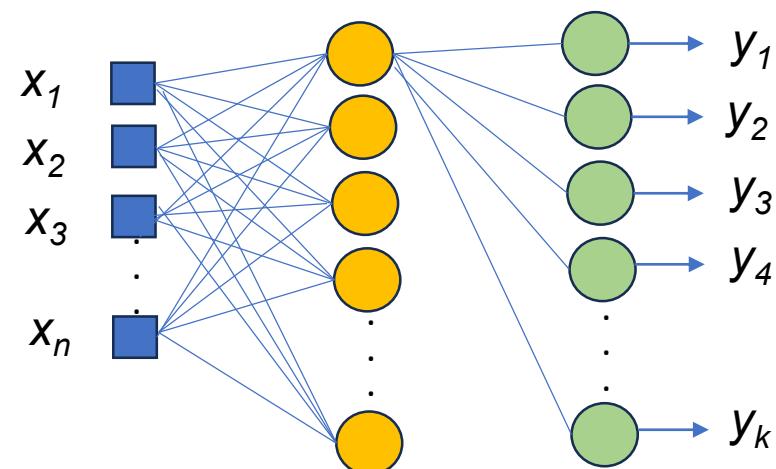


Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

- Camada de entrada: corresponde à camada de dados
- Camada oculta: de neurônios
- Camada de saída: de neurônios

Totalmente Conectada
Parcialmente Conectada



Como definir a topologia de uma rede MLP?

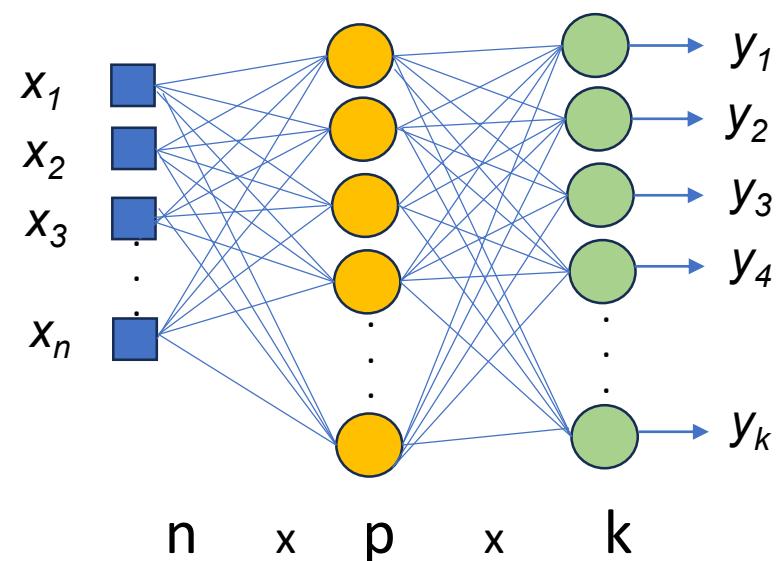
- A topologia mais simples possui 3 camadas:

- Camada de entrada: corresponde à camada de dados
- Camada oculta: de neurônios
- Camada de saída: de neurônios

Totalmente Conectada

- n dados de entrada
- p neurônios na camada oculta
- k neurônios na camada de saída

Parcialmente Conectada



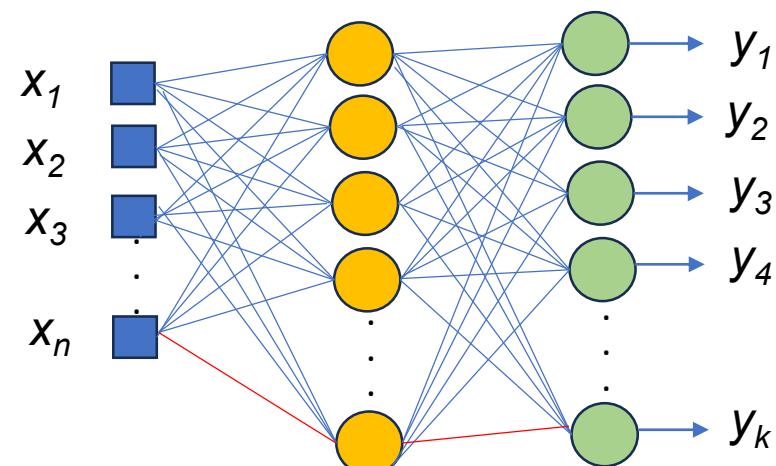
Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

- Camada de entrada: corresponde à camada de dados
- Camada oculta: de neurônios
- Camada de saída: de neurônios



Totalmente Conectada
Parcialmente Conectada
Eliminação de conexões



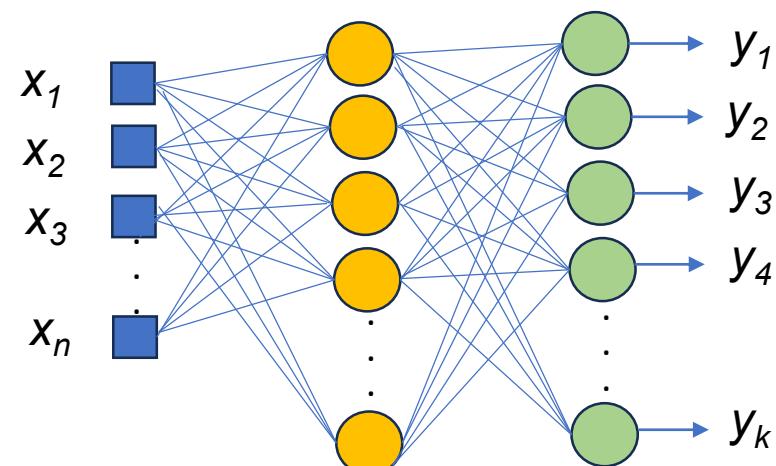
Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

- Camada de entrada: corresponde à camada de dados
- Camada oculta: de neurônios
- Camada de saída: de neurônios



Totalmente Conectada
Parcialmente Conectada
Eliminação de conexões



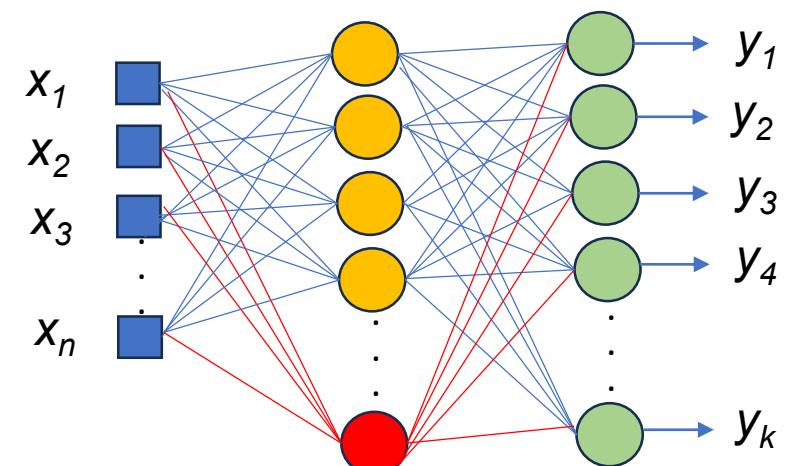
Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

- Camada de entrada: corresponde à camada de dados
- Camada oculta: de neurônios
- Camada de saída: de neurônios



Totalmente Conectada
Parcialmente Conectada
Eliminação de conexões
Eliminação de neurônios



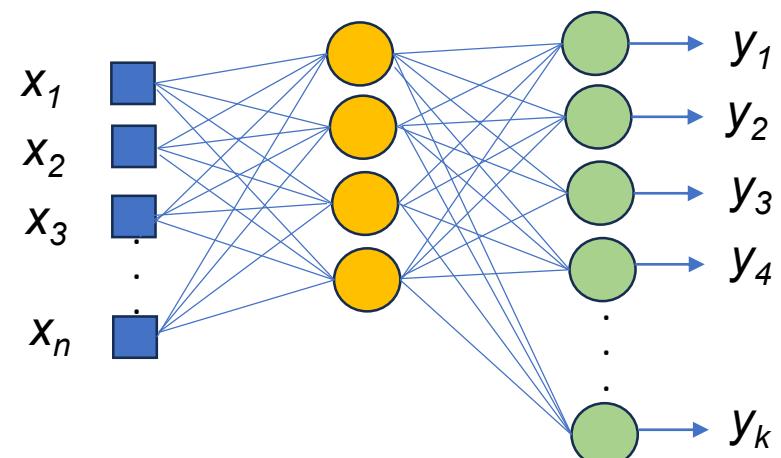
Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

- Camada de entrada: corresponde à camada de dados
- Camada oculta: de neurônios
- Camada de saída: de neurônios



Totalmente Conectada
Parcialmente Conectada
Eliminação de conexões
Eliminação de neurônios

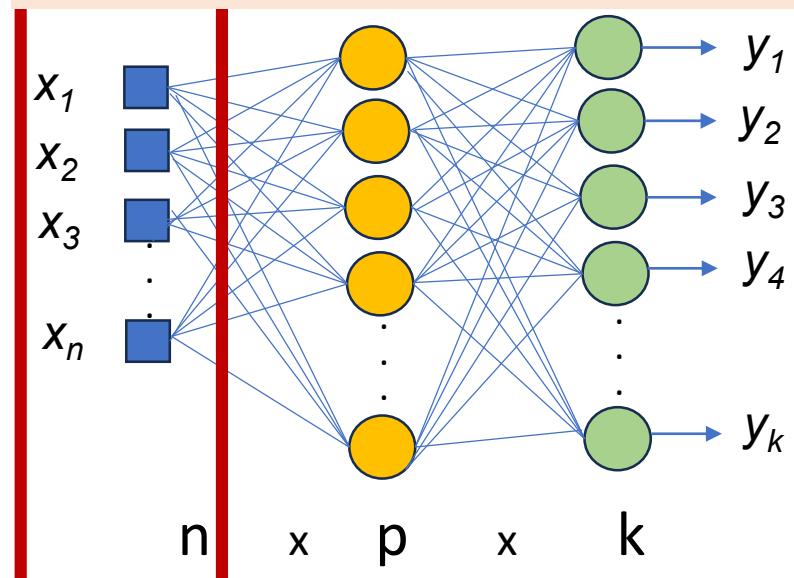


Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

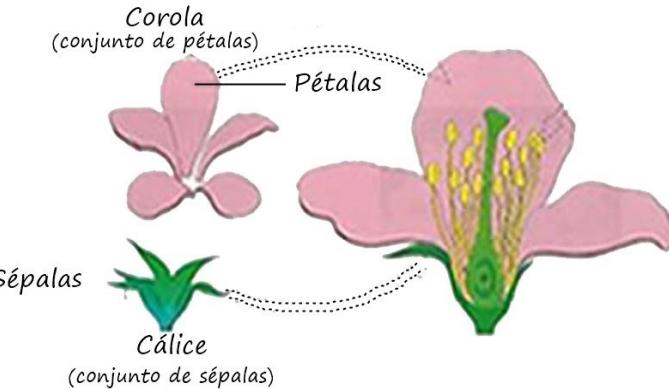
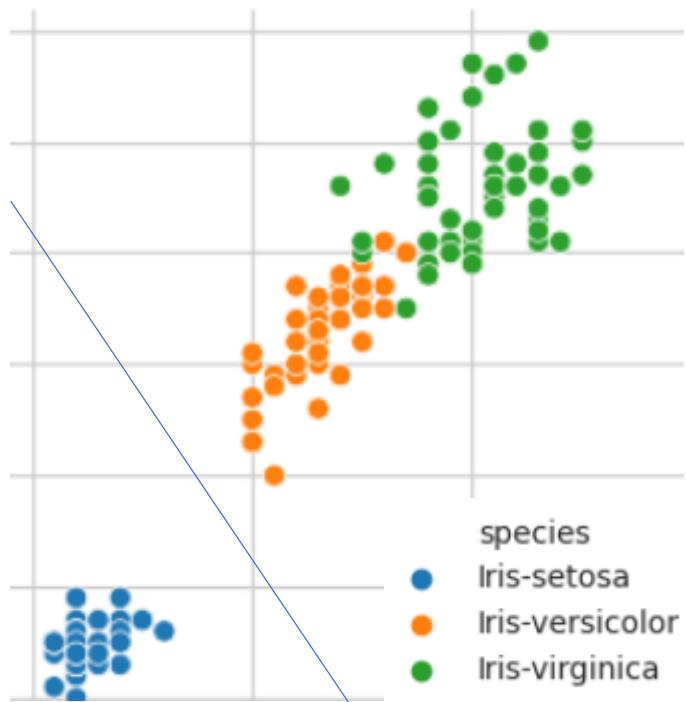
- **Camada de entrada**: corresponde `a camada de dados
- Camada oculta: de neurônios
- Camada de saída: de neurônios

n : quantidade de dados de entrada, número de features/atributos de entrada



Planta Iris

4 atributos de entrada

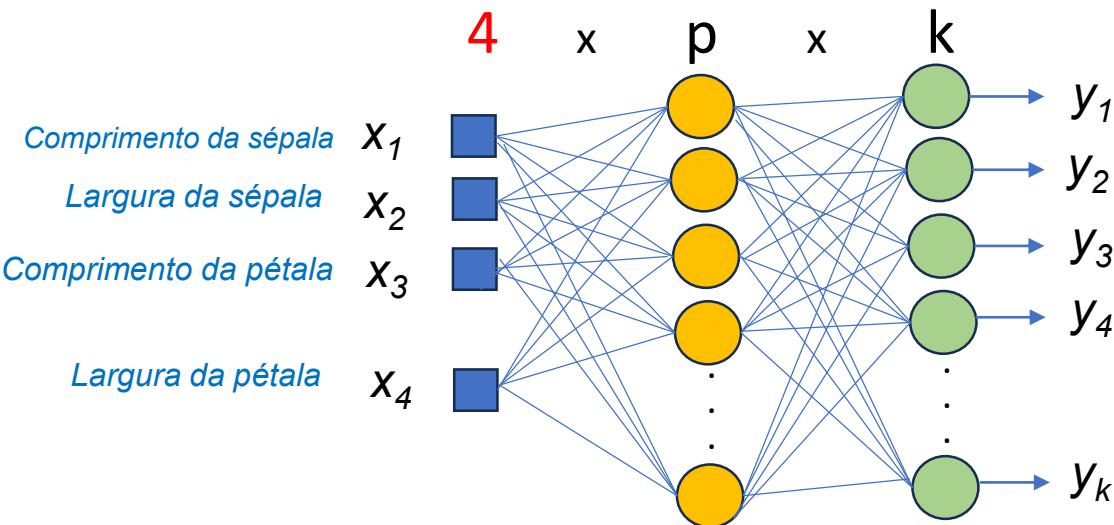


	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

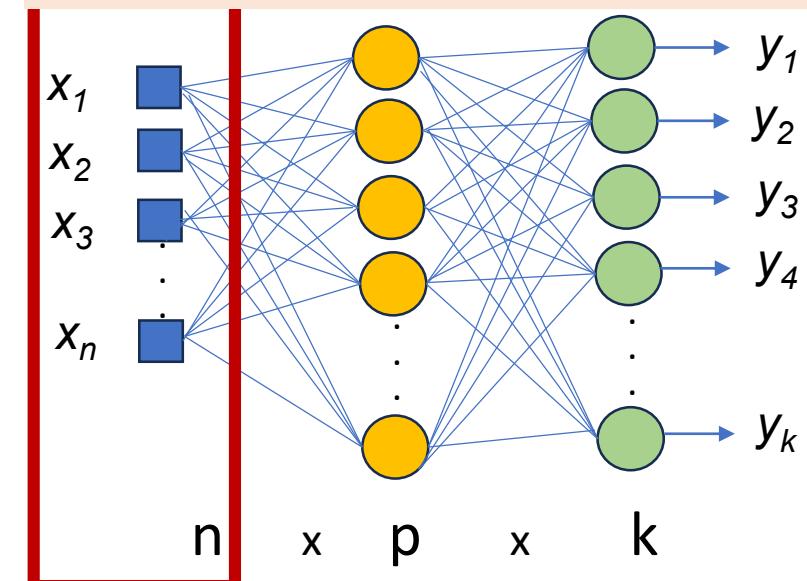
Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

- **Camada de entrada**: corresponde `a camada de dados
- Camada oculta: de neurônios
- Camada de saída: de neurônios



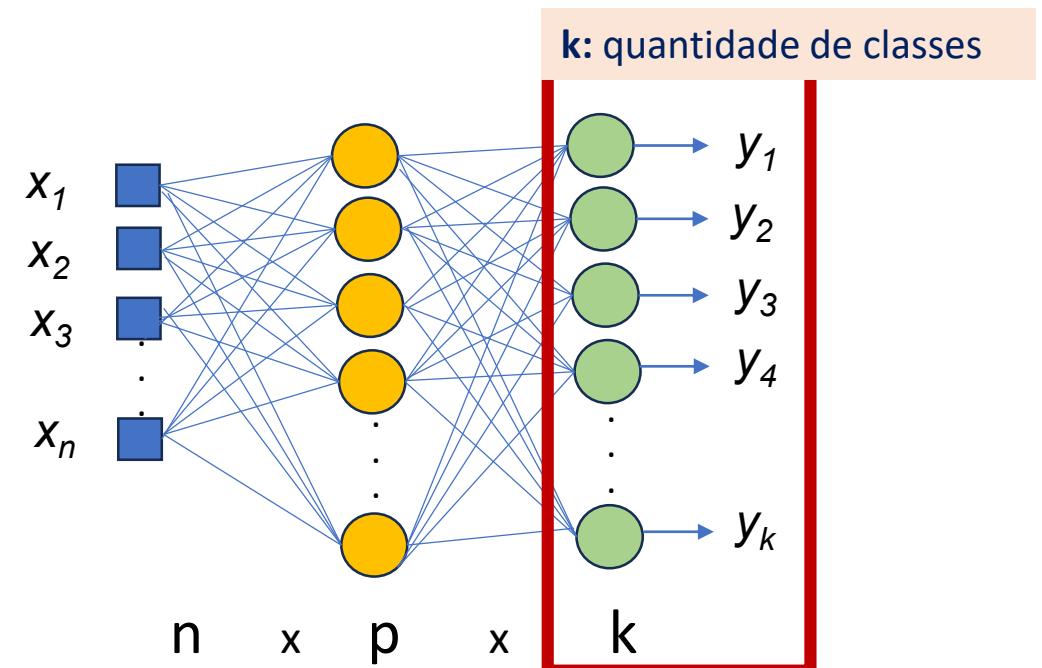
n : quantidade de dados de entrada, número de features/atributos de entrada



Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

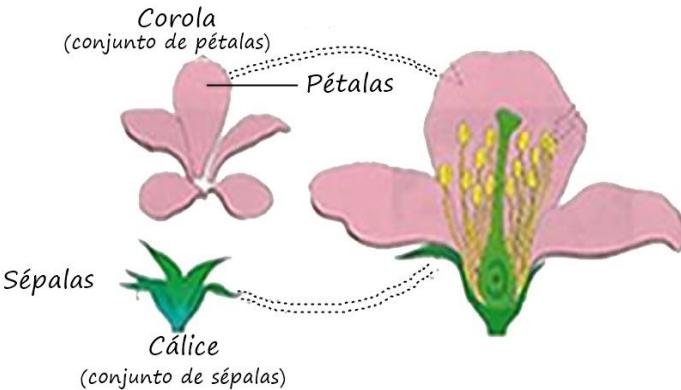
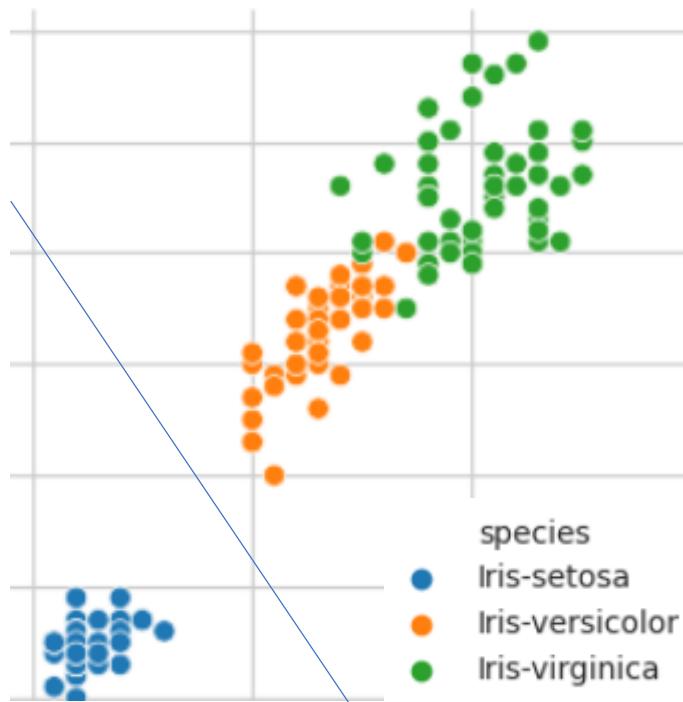
- Camada de entrada: corresponde à camada de dados
- Camada oculta: de neurônios
- **Camada de saída:** de neurônios



Planta Iris

4 atributos de entrada

3 classes: iris-setosa, iris-versicolor e iris-virginica

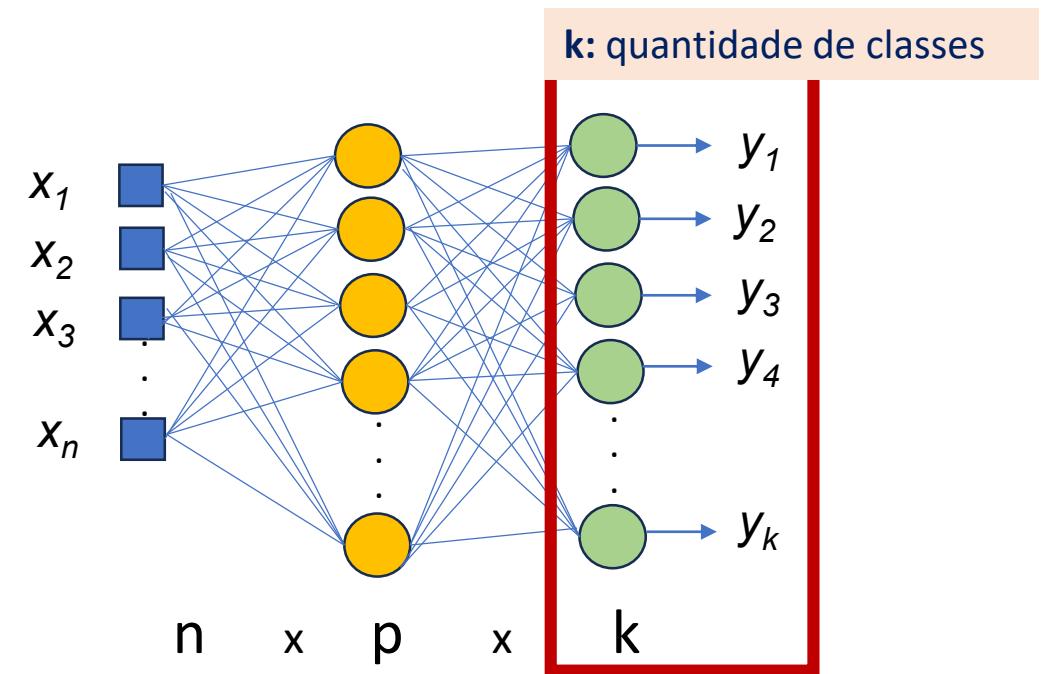
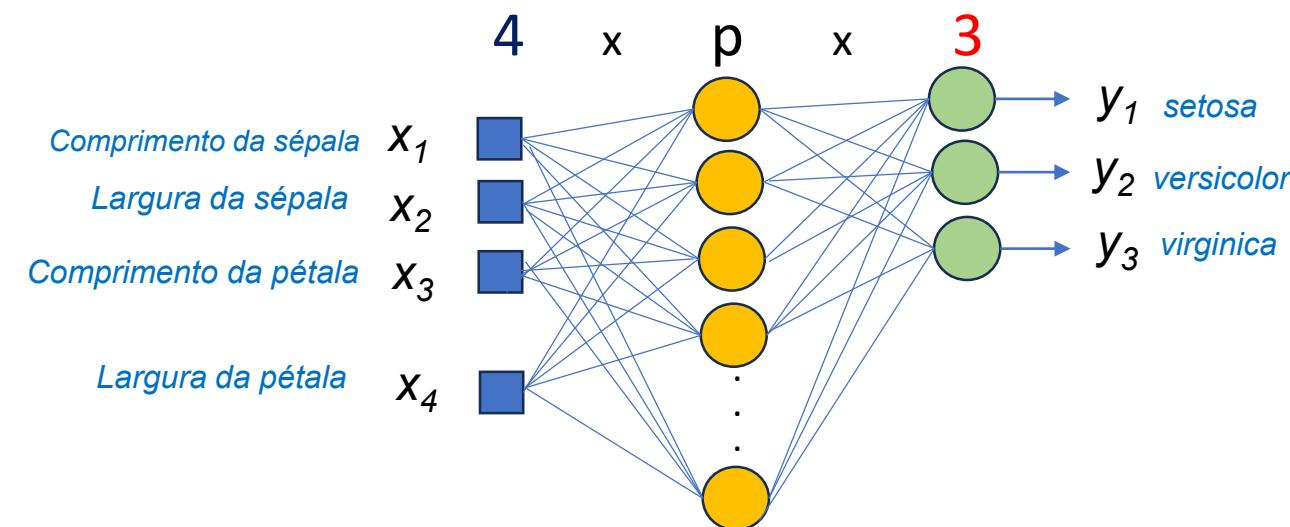


	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

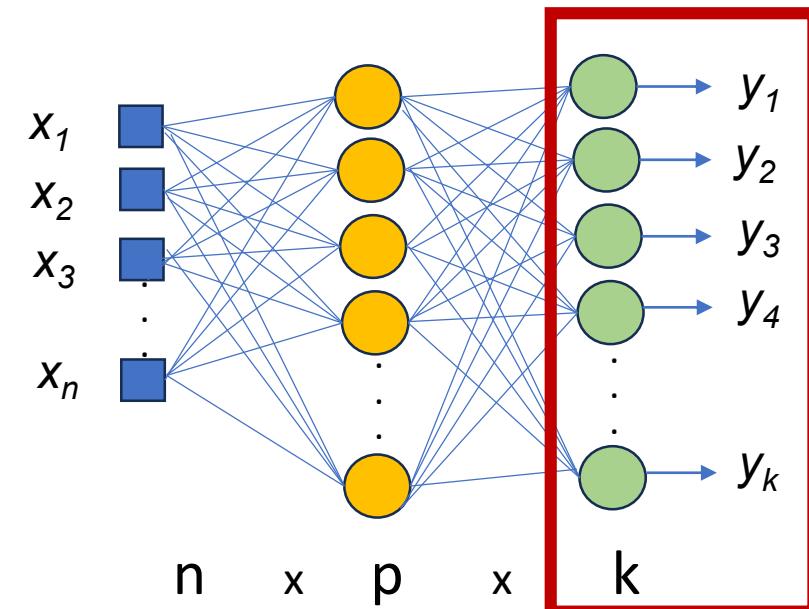
- Camada de entrada: corresponde à camada de dados
- Camada oculta: de neurônios
- **Camada de saída:** de neurônios



Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

- Camada de entrada: corresponde à camada de dados
- **Camada oculta:** de neurônios
- Camada de saída: de neurônios

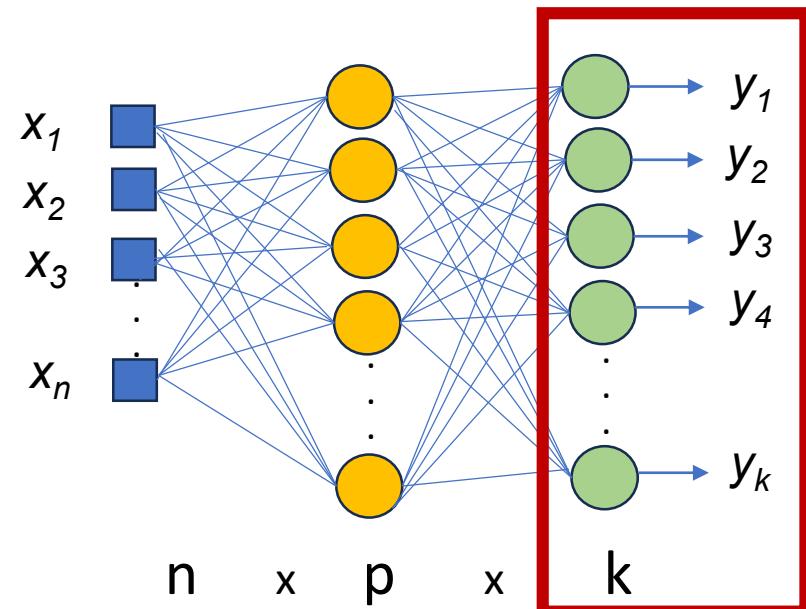


Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

Quantos neurônios usamos na camada oculta?

- Camada de entrada: corresponde à camada de dados
- **Camada oculta:** de neurônios
- Camada de saída: de neurônios

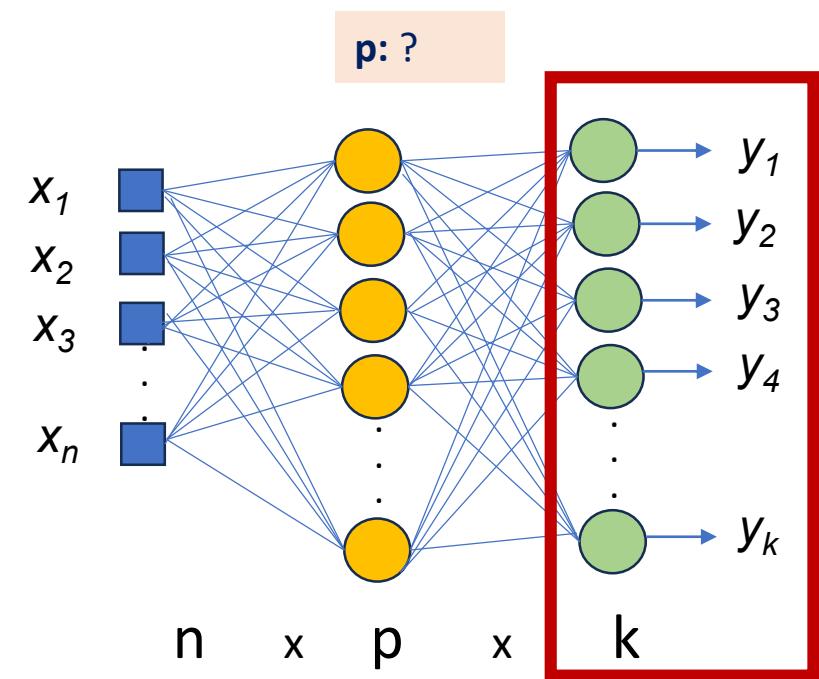
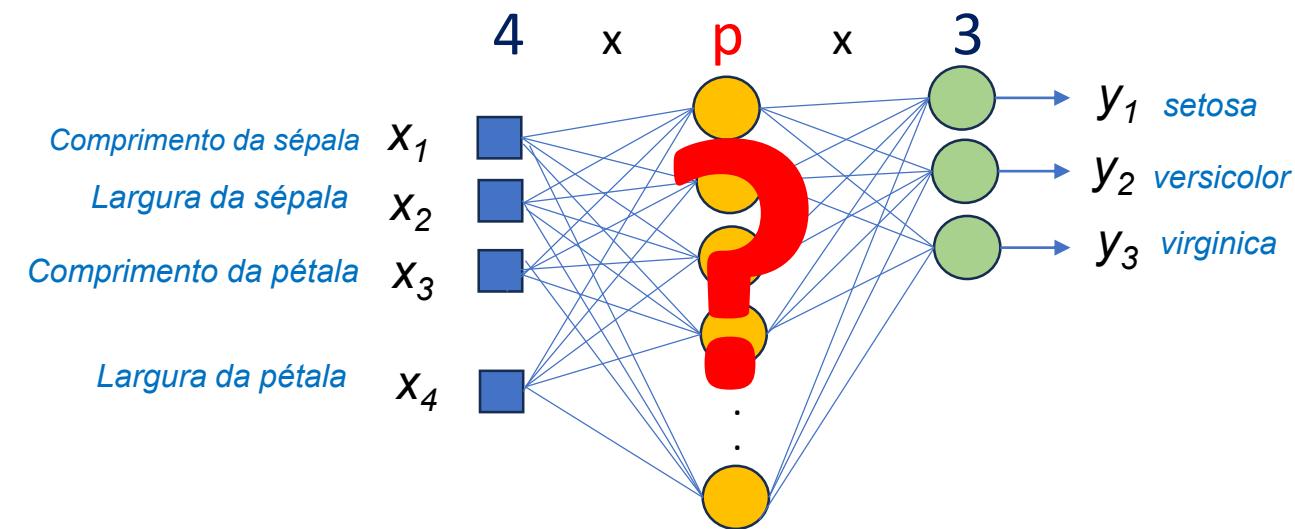


Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

Quantos neurônios usamos na camada oculta?

- Camada de entrada: corresponde à camada de dados
- Camada oculta:** de neurônios
- Camada de saída: de neurônios



Como definir a topologia de uma rede MLP?

Quantos neurônios usamos na camada oculta?

- De forma objetiva **ainda não há como definir**.
- Utiliza-se **experimentação sistemática** para descobrir o que funciona melhor para seu conjunto de dados específico.
- Aplica-se **heurísticas** para estimar o número inicial de neurônios.

Como definir a topologia de uma rede MLP?

E o problema se estende para as camadas...

Quantas camadas ocultas são necessárias?

- Vale a mesma resposta:
 - De forma objetiva **ainda não há como definir**.
 - Utiliza-se **experimentação sistemática** para descobrir o que funciona melhor para seu conjunto de dados específico.
 - Aplica-se **heurísticas** para estimar o número inicial de neurônios.

Como definir a topologia de uma rede MLP?

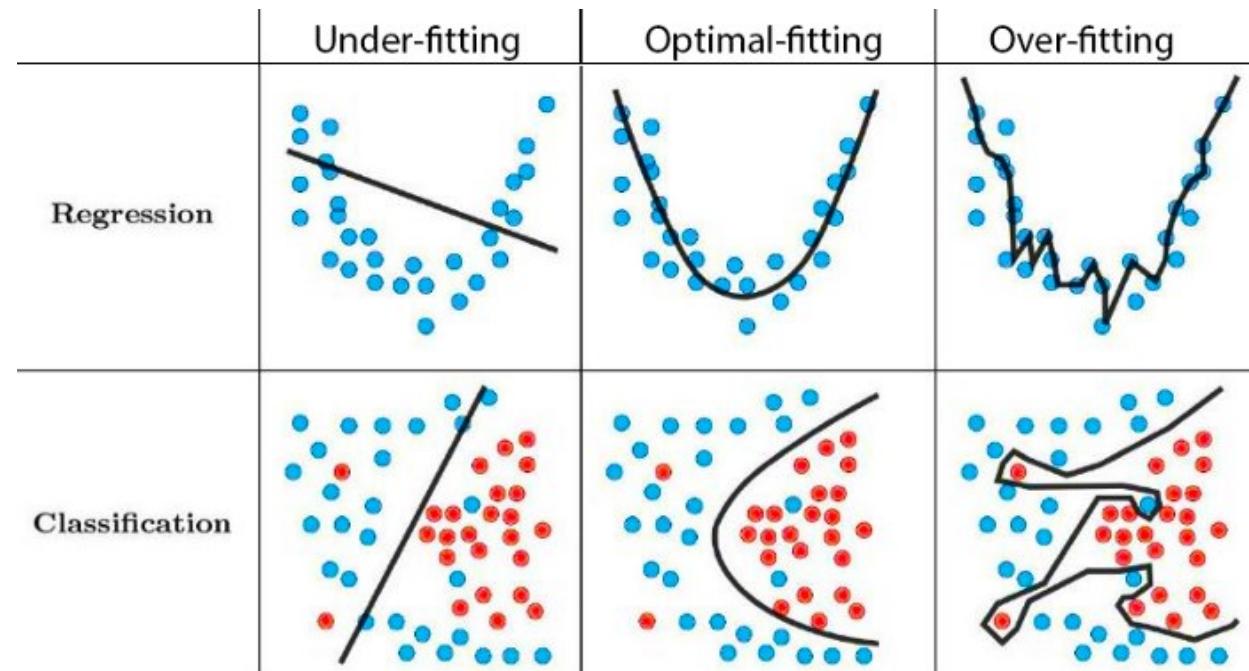
- Algumas estratégias populares:
 - **Aleatório**: experimente configurações aleatórias de camadas e nós (neuronios) por camada.
 - **Grid**: tente uma pesquisa sistemática no número de camadas e nós por camada.
 - **Heurística**: tente uma pesquisa direcionada usando algoritmos que para encontrar as configurações ideais tal como algoritmo genético ou bayesiano.
 - **Exaustivo**: experimente todas as combinações de camadas e o número de nós; pode ser viável para pequenas redes e conjuntos de dados.

Como definir a topologia de uma rede MLP?

- Orientações populares e algumas heurísticas conhecidas:
 - **Uma camada oculta costuma ser suficiente** para a maioria das aplicações;
 - Podem ser necessárias **várias camadas ocultas no caso de domínios com muitas entradas**. Quando precisa-se trabalhar com textos e imagens por exemplo.

Como definir a topologia de uma rede MLP?

- Orientações populares e algumas heurísticas conhecidas:
- **Poucos neurônios** na camada oculta pode gerar **Underfitting**.
- **Muitos neurônios** na camada oculta pode gerar **Overfitting**. Aumenta o tempo de processamento.



<https://towardsdatascience.com/techniques-for-handling-underfitting-and-overfitting-in-machine-learning-348daa2380b9>

Como definir a topologia de uma rede MLP?

- Não existe uma fórmula para selecionar o número ideal de neurônios das camadas ocultas.
- No entanto, algumas dicas e heurísticas estão disponíveis na literatura da área:
 - O número de neurônios ocultos deve estar entre o tamanho da camada de entrada e o tamanho da camada de saída.
 - O número de neurônios ocultos deve ser $2/3$ do tamanho da camada de entrada, mais o tamanho da camada de saída.
 - O número de neurônios ocultos deve ser menor que o dobro do tamanho da camada de entrada.

Como definir a topologia de uma rede MLP?

- Uma aproximação pode ser obtida pela regra da pirâmide geométrica proposta por Masters (1993).
- Para uma rede de três camadas com n neurônios de entrada e k neurônios de saída, a camada oculta poderia ter:

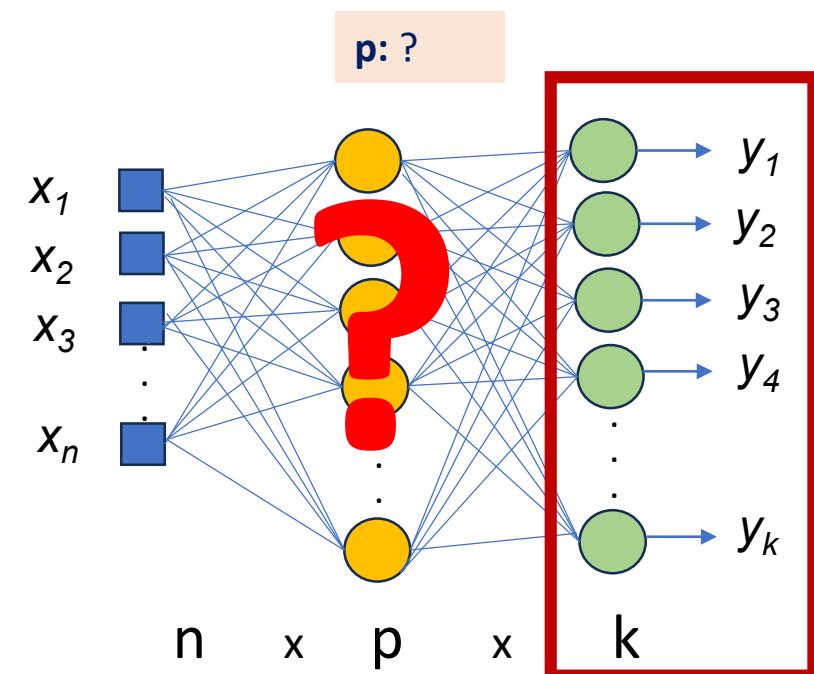
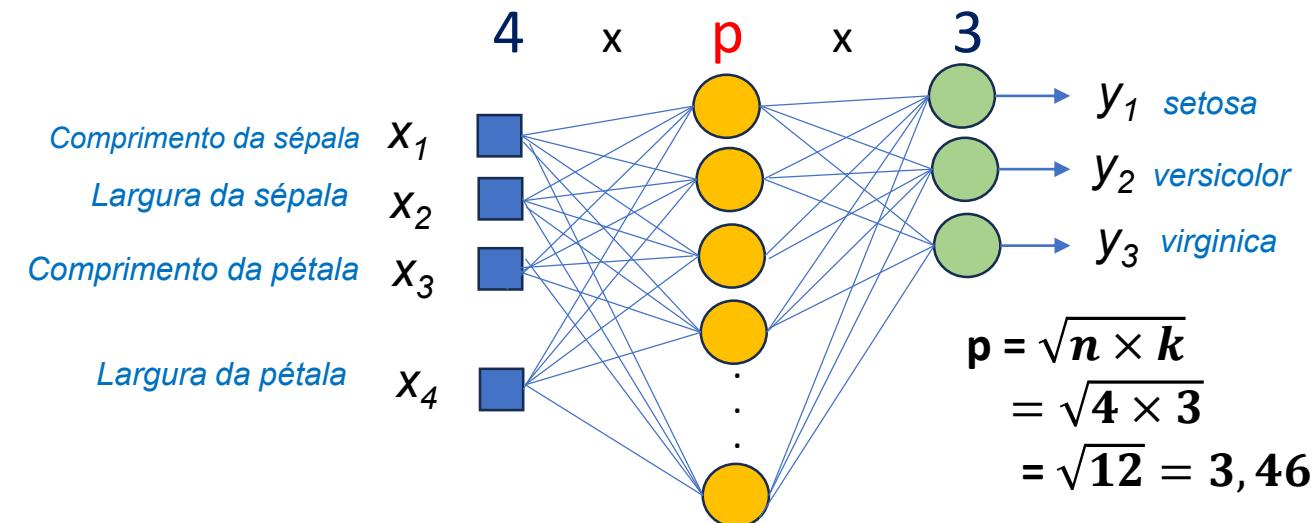
$$\text{Número} = \sqrt{n \times k}$$

Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

Quantos neurônios usamos na camada oculta?

- Camada de entrada: corresponde à camada de dados
- Camada oculta:** de neurônios
- Camada de saída: de neurônios

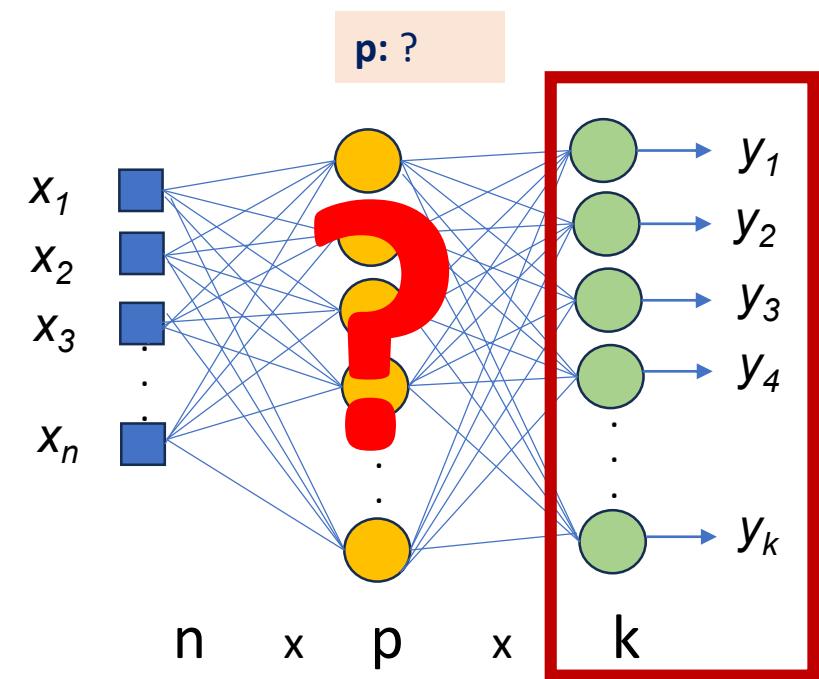
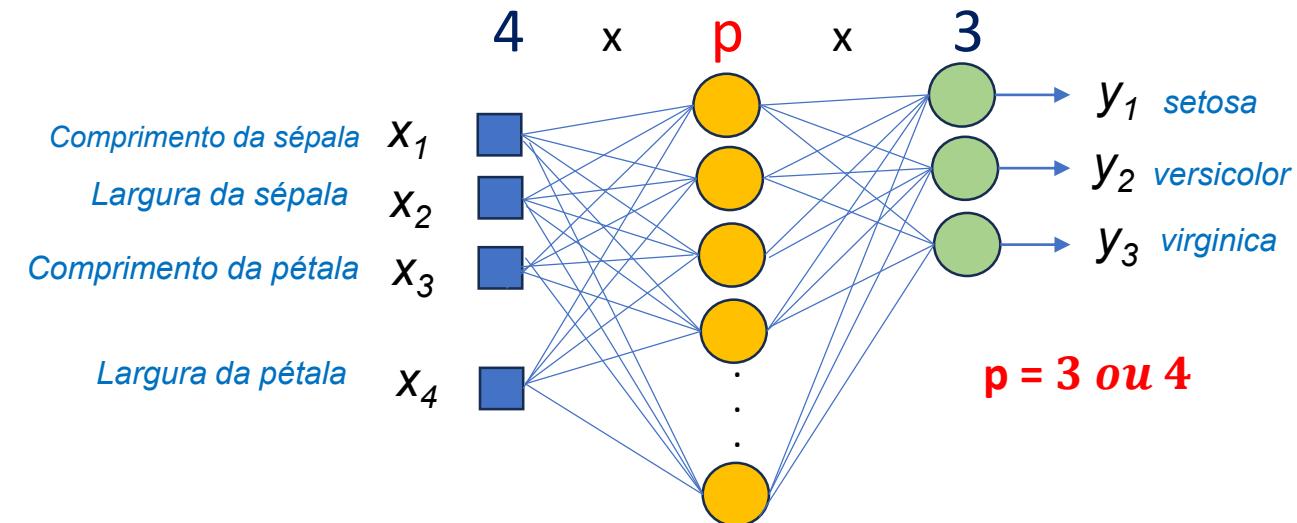


Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

Quantos neurônios usamos na camada oculta?

- Camada de entrada: corresponde à camada de dados
- Camada oculta:** de neurônios
- Camada de saída: de neurônios

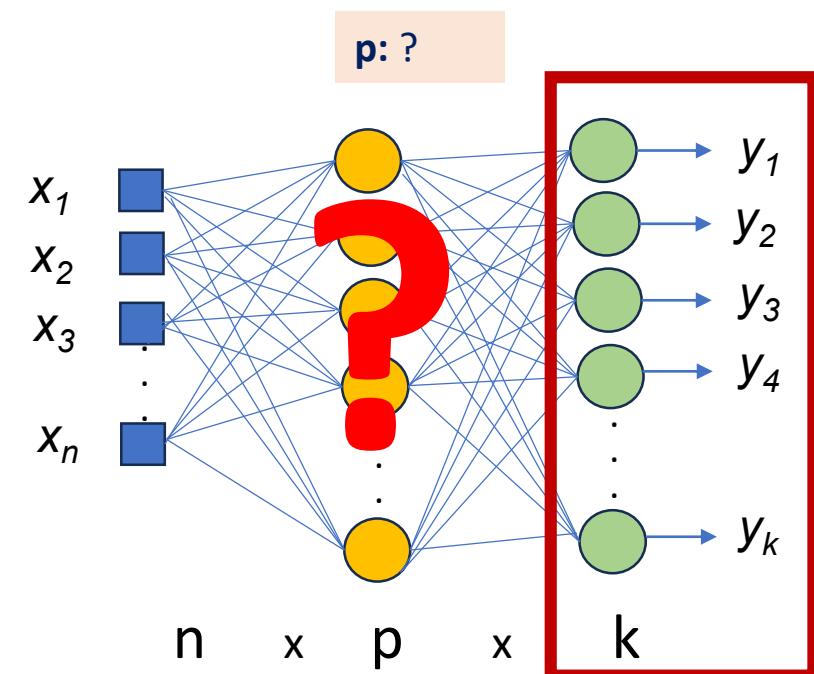
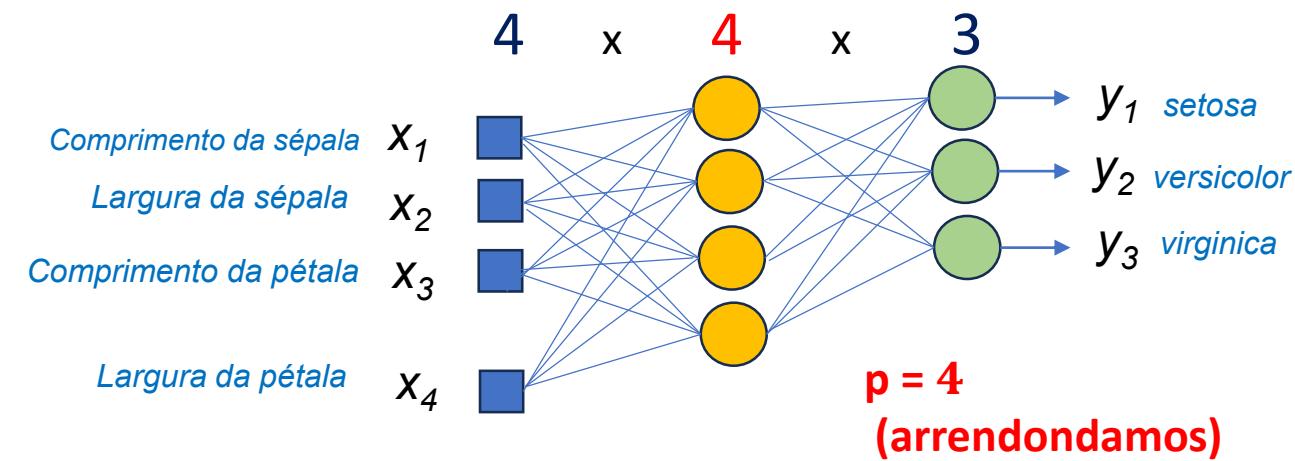


Como definir a topologia de uma rede MLP?

- A topologia mais simples possui 3 camadas:

Quantos neurônios usamos na camada oculta?

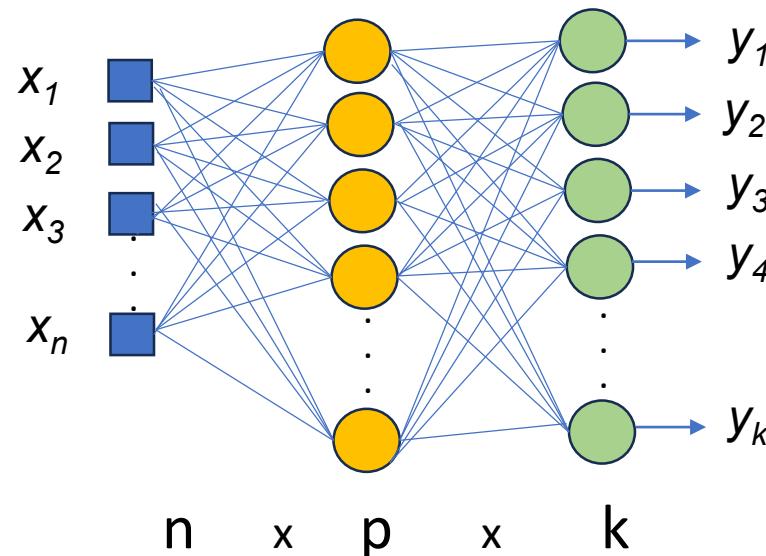
- Camada de entrada: corresponde à camada de dados
- Camada oculta:** de neurônios
- Camada de saída: de neurônios



Como definir a topologia de uma rede MLP?

Mais um exemplo: Tabela XOR

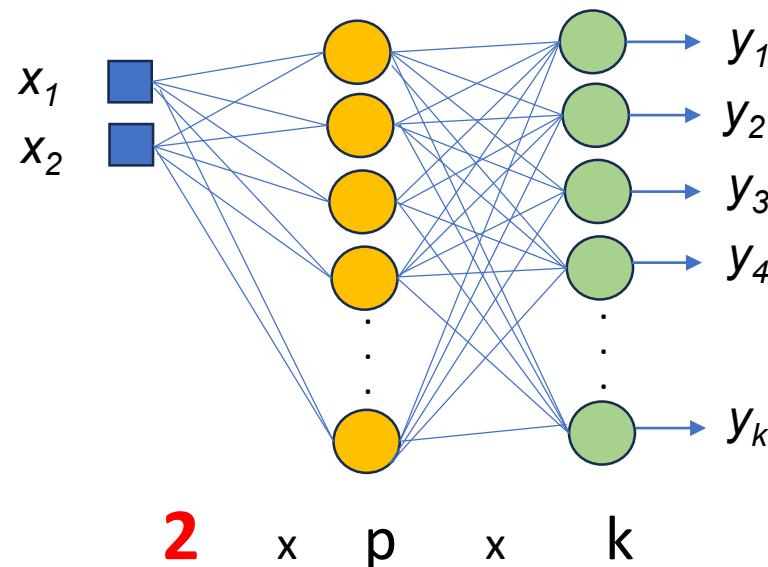
x1	x2	d
0	0	0
0	1	1
1	0	1
1	1	0



Como definir a topologia de uma rede MLP?

Mais um exemplo: Tabela XOR

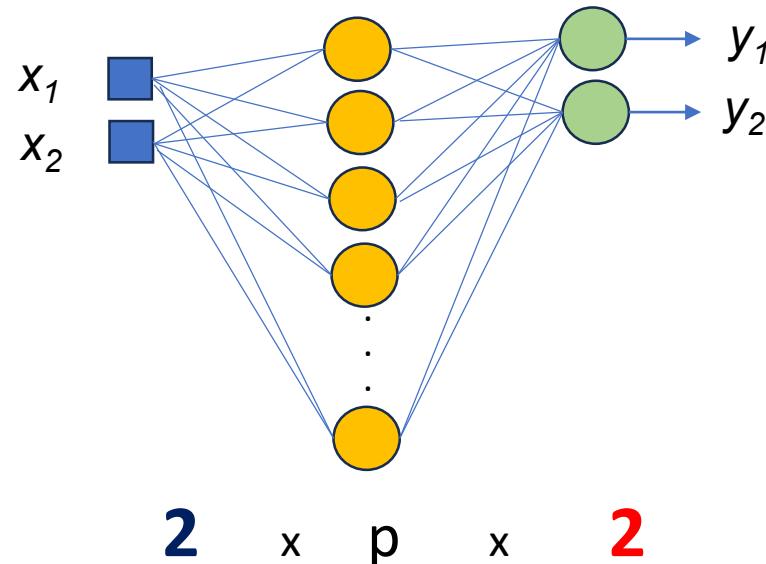
x ₁	x ₂	d
0	0	0
0	1	1
1	0	1
1	1	0



Como definir a topologia de uma rede MLP?

Mais um exemplo: Tabela XOR

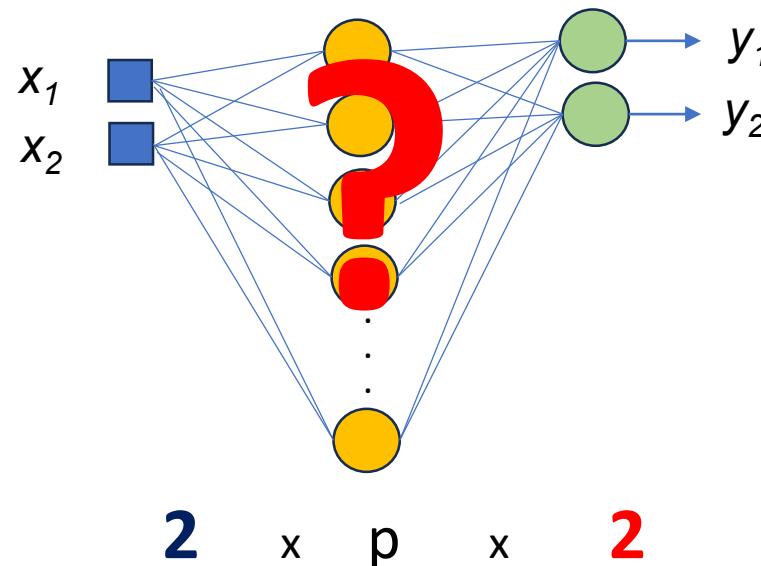
x ₁	x ₂	d
0	0	0
0	1	1
1	0	1
1	1	0



Como definir a topologia de uma rede MLP?

Mais um exemplo: Tabela XOR

x ₁	x ₂	d
0	0	0
0	1	1
1	0	1
1	1	0

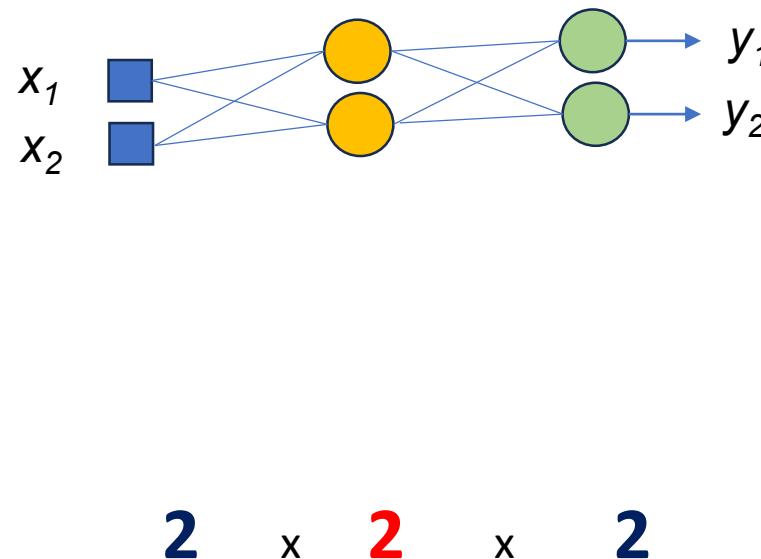


$$\begin{aligned}p &= \sqrt{n \times k} \\&= \sqrt{2 \times 2} \\&= \sqrt{4} = 2\end{aligned}$$

Como definir a topologia de uma rede MLP?

Mais um exemplo: Tabela XOR

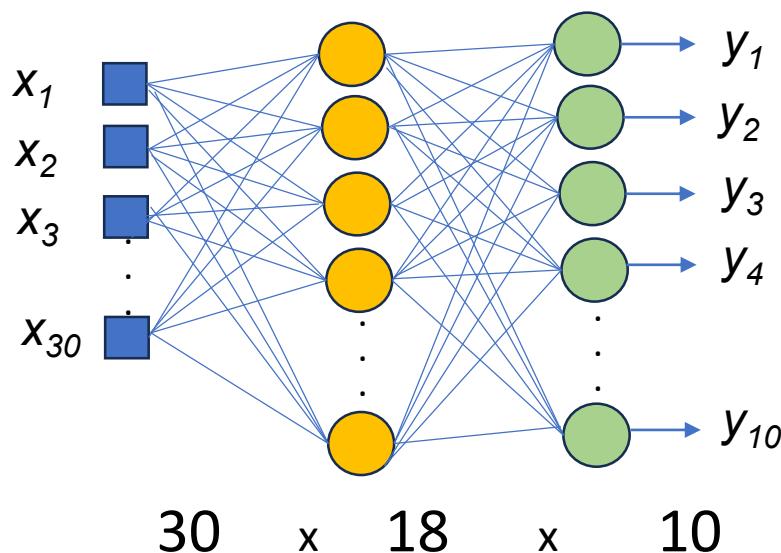
x ₁	x ₂	d
0	0	0
0	1	1
1	0	1
1	1	0



$$\begin{aligned} p &= \sqrt{n \times k} \\ &= \sqrt{2 \times 2} \\ &= \sqrt{4} = 2 \end{aligned}$$

Como definir a topologia de uma rede MLP?

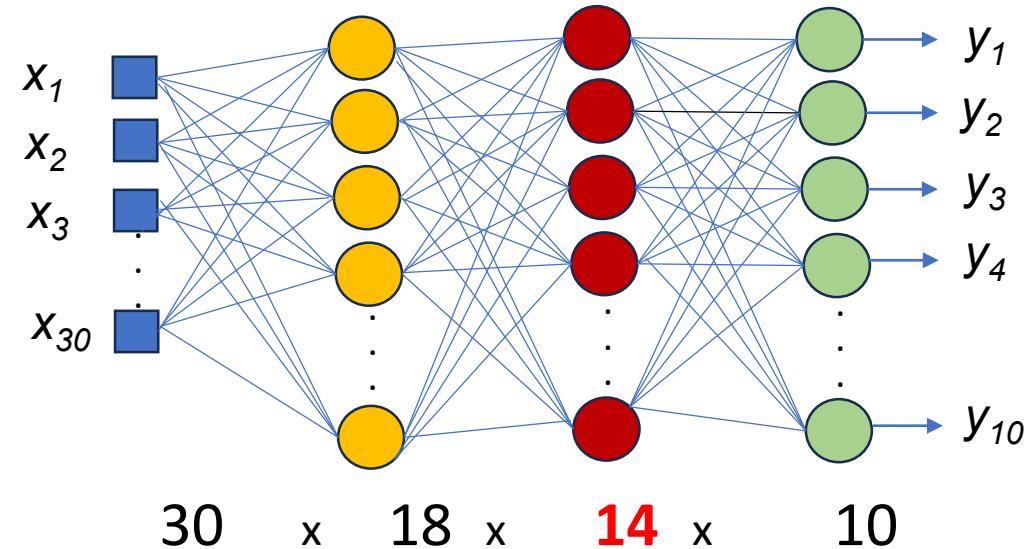
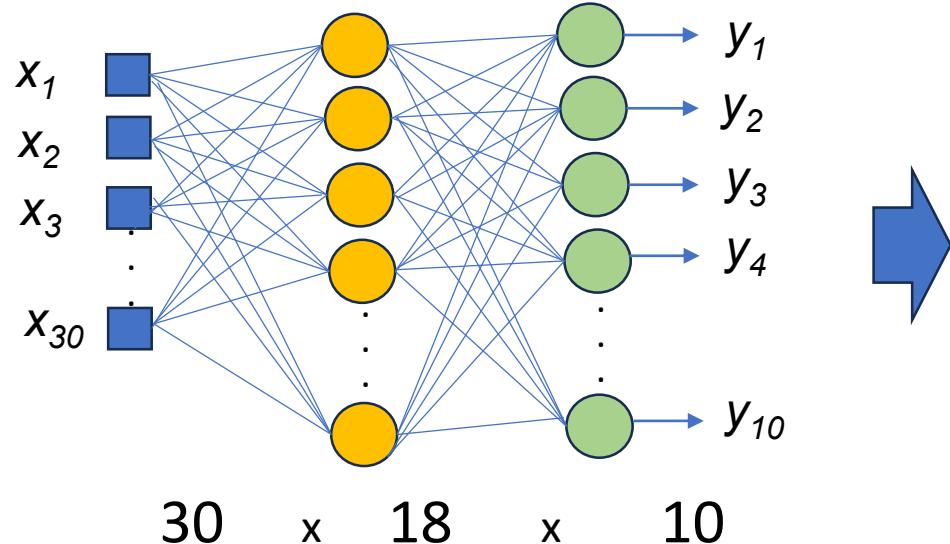
- E o que fazer quando são necessárias mais camadas?
 - Você pode tentar usar a mesma regra.



$$p = \sqrt{n \times k} = \sqrt{30 \times 10} = \sqrt{300} = 17,32$$

Como definir a topologia de uma rede MLP?

- E o que fazer quando são necessárias mais camadas?
 - Você pode tentar usar a mesma regra.



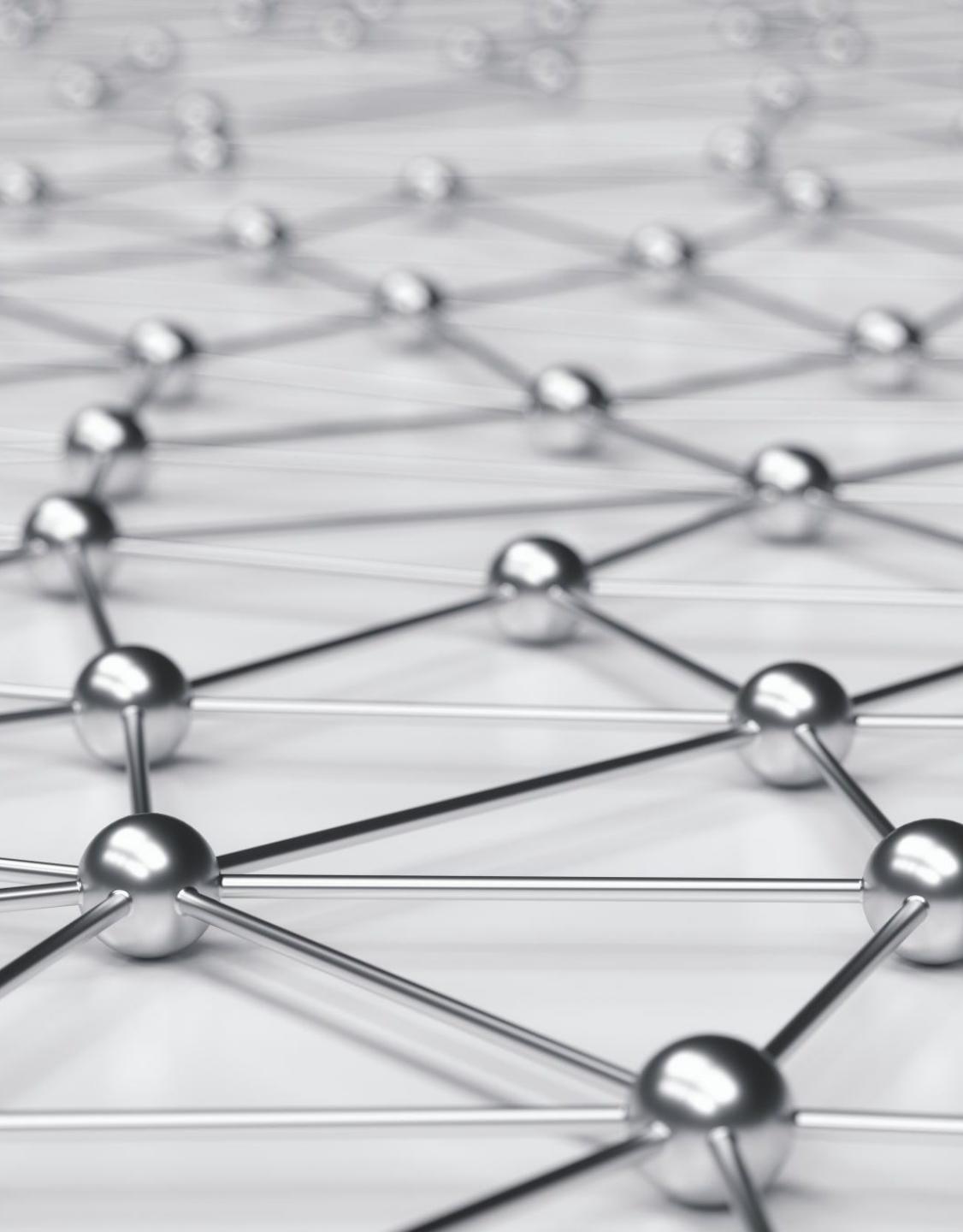
$$p = \sqrt{n \times k} = \sqrt{30 \times 10} = \sqrt{300} = 17,32$$

$$p = \sqrt{n \times k} = \sqrt{18 \times 10} = \sqrt{180} = 13,41$$

A black silhouette of a person's head and shoulders is facing right, holding a white smartphone. The screen of the phone displays a glowing blue network graph with numerous nodes and connecting lines.

Como definir a topologia de uma rede MLP?

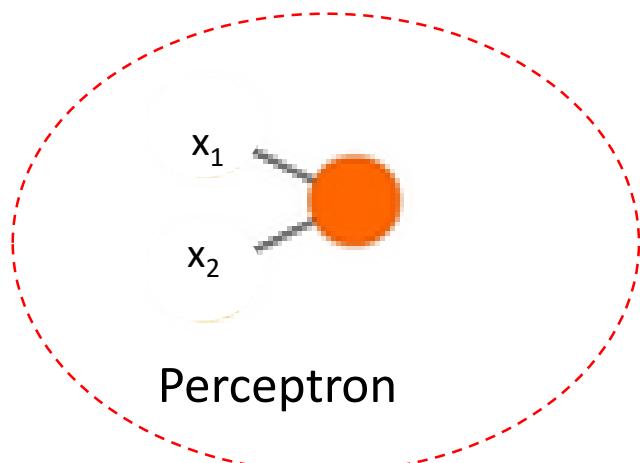
- Lembre-se:
Independentemente da heurística usada para definir a topologia inicial, **você terá que realizar testes para encontrar a melhor topologia para a rede MLP.**



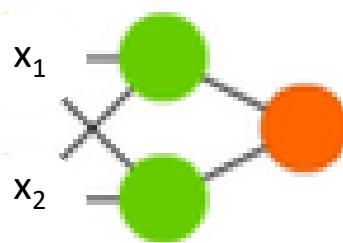
Revisitando a rede Perceptron

Redes Feed Forward

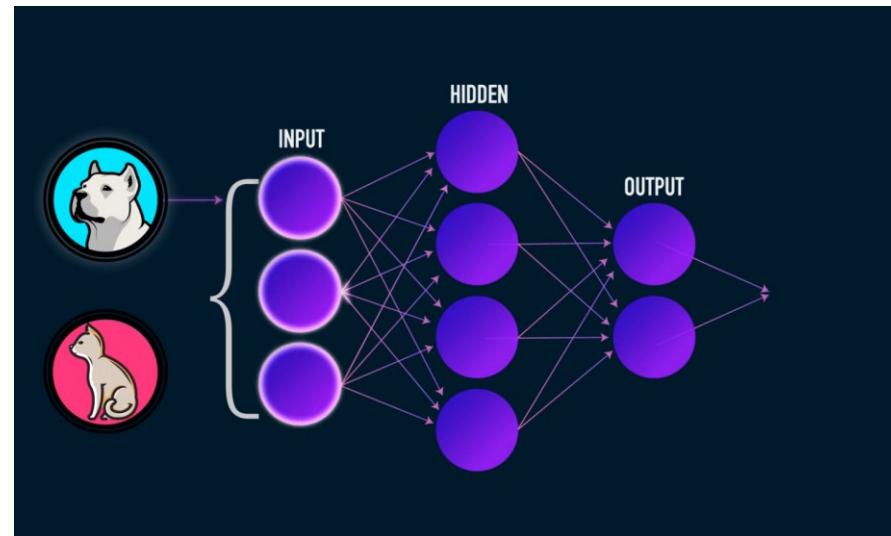
Tarefas Supervisionadas: **classificação** o e **regressão**



Perceptron



MultiLayer Perceptron

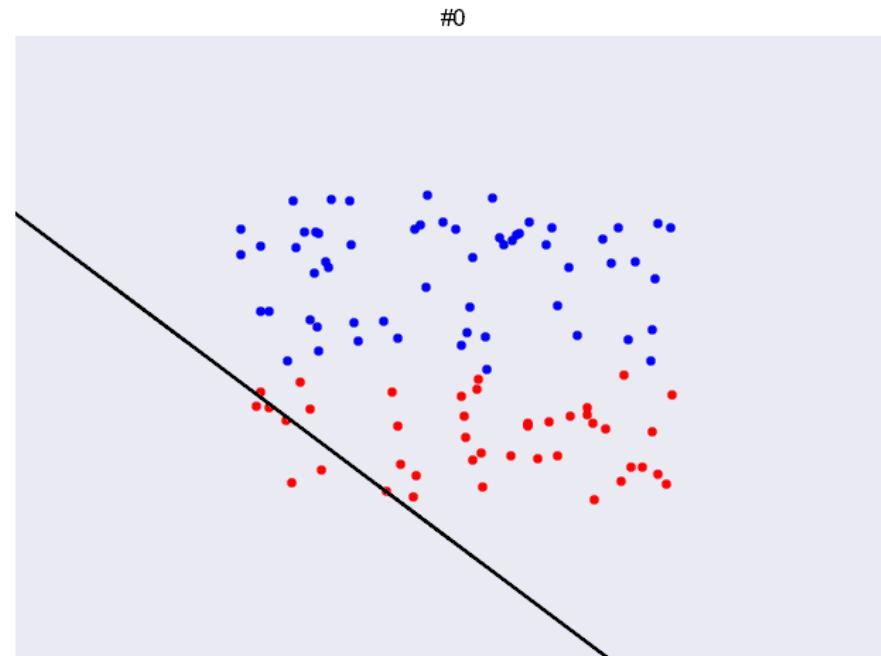


<https://sites.icmc.usp.br/andre/research/neural/>

Propagação: fluxo do sinal sempre para frente

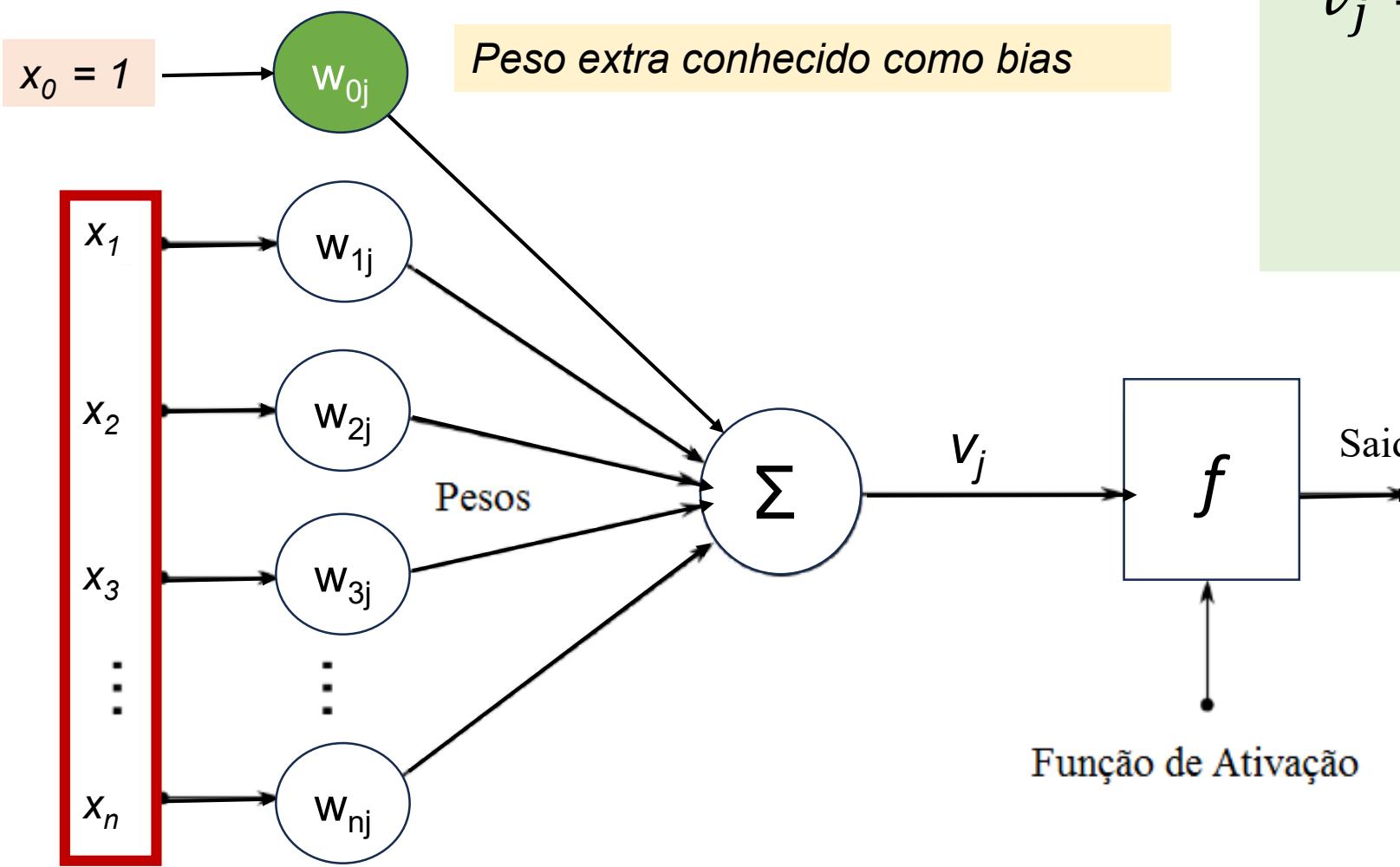
Rede Perceptron

- Limitação: só consegue resolver problemas linearmente separáveis.
- O algoritmo de aprendizado desse tipo de rede, procura os coeficientes que traçam a reta que separa linearmente os dados de uma classe de outra.



Rede Perceptron

Revisitando um neurônio artificial j:

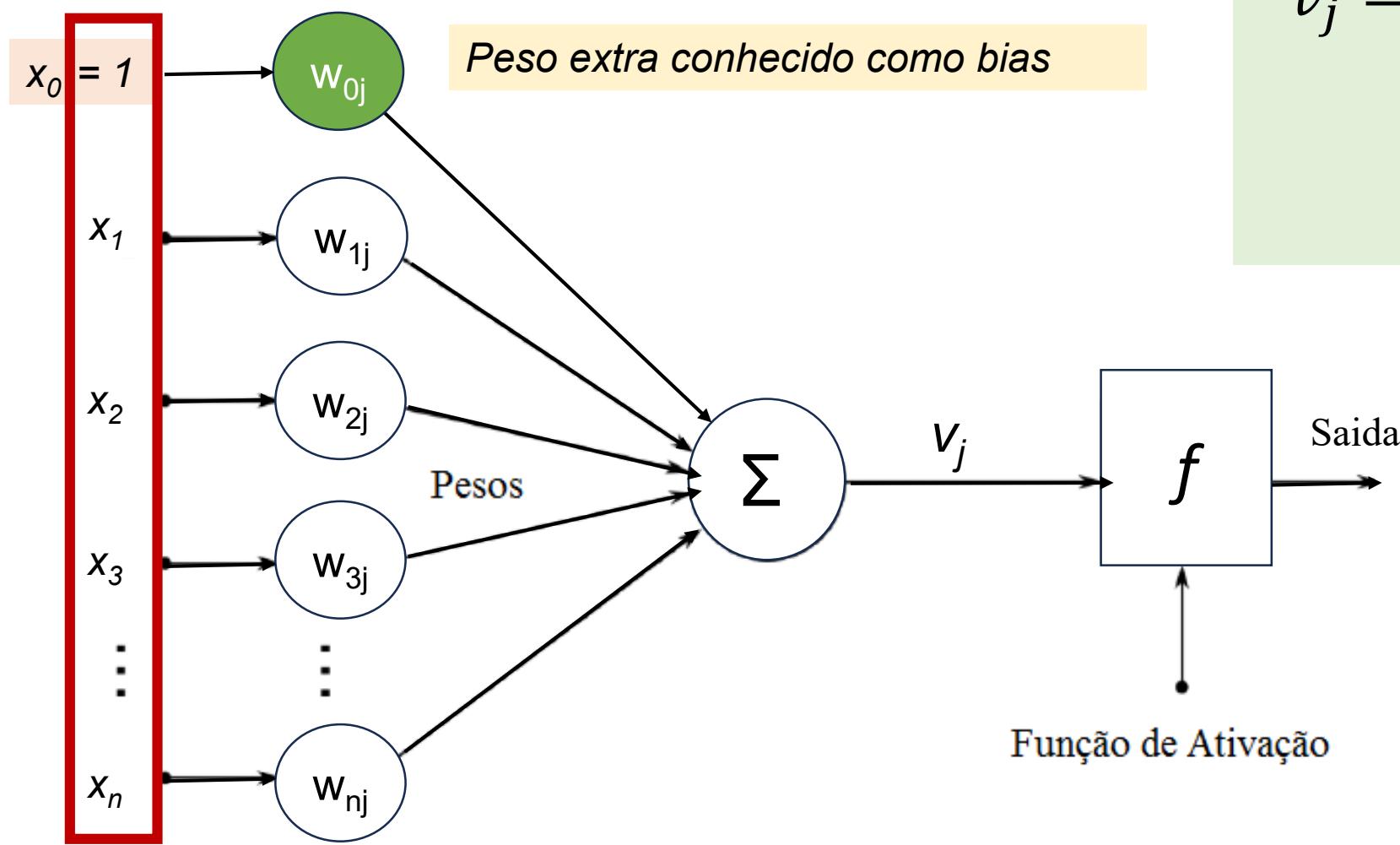


$$v_j = w_{0j} + \sum_{i=1}^n (x_i \times w_{ij})$$
$$y_j = f(v_j)$$

X: $x_1, x_2, x_3, \dots, x_p$
O vetor X representa a entrada do neurônio.
Pode ser os dados de entrada da rede ou o sinal de saída de outros neurônios.

Rede Perceptron

Revisitando um neurônio artificial j:

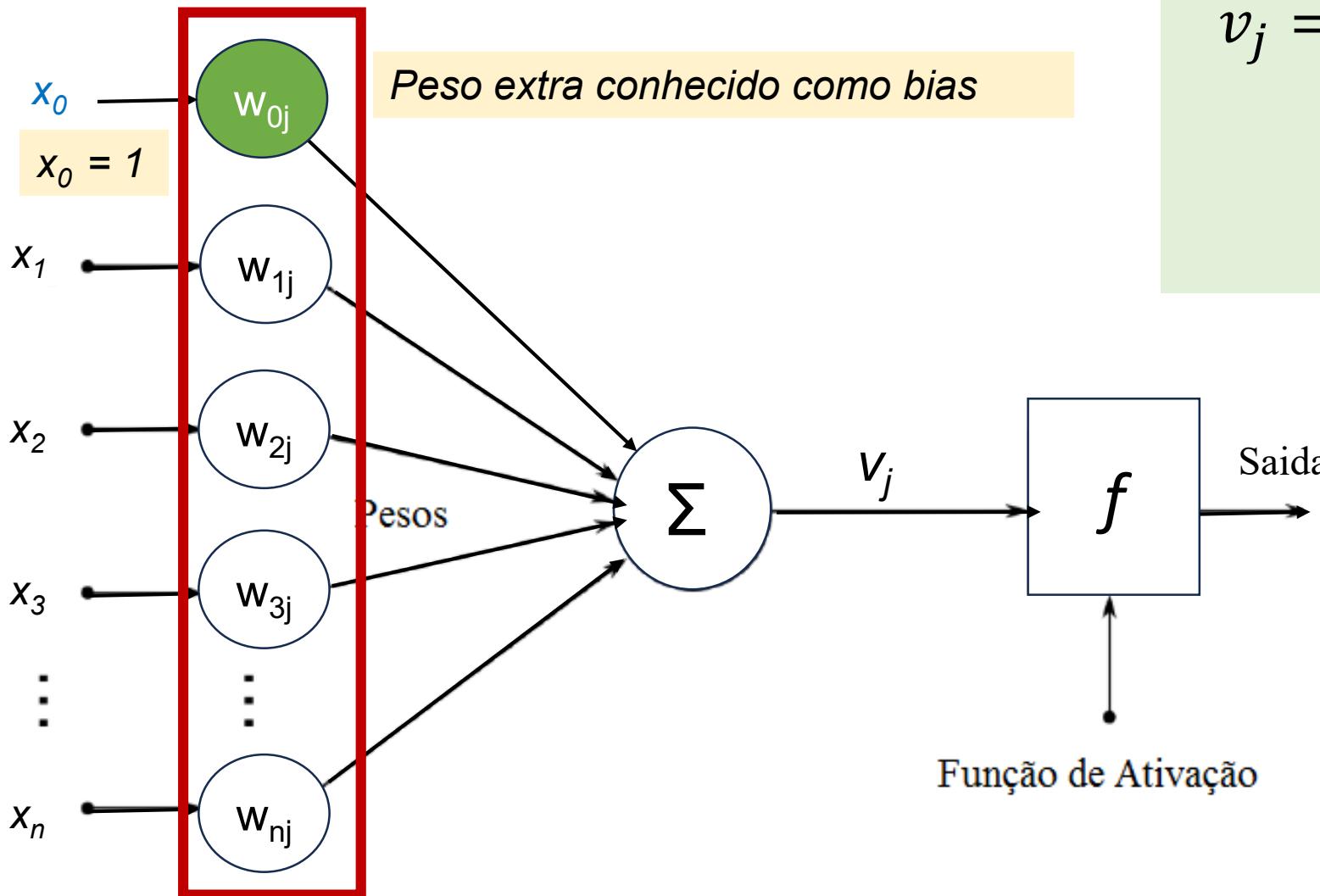


$$v_j = w_{0j} + \sum_{i=1}^n (x_i \times w_{ij})$$
$$y_j = f(v_j)$$

X: $x_1, x_2, x_3, \dots, x_p$
O vetor X representa a entrada do neurônio.
Pode ser os dados de entrada da rede ou o sinal de saída de outros neurônios.

Rede Perceptron

Revisitando um neurônio artificial j:

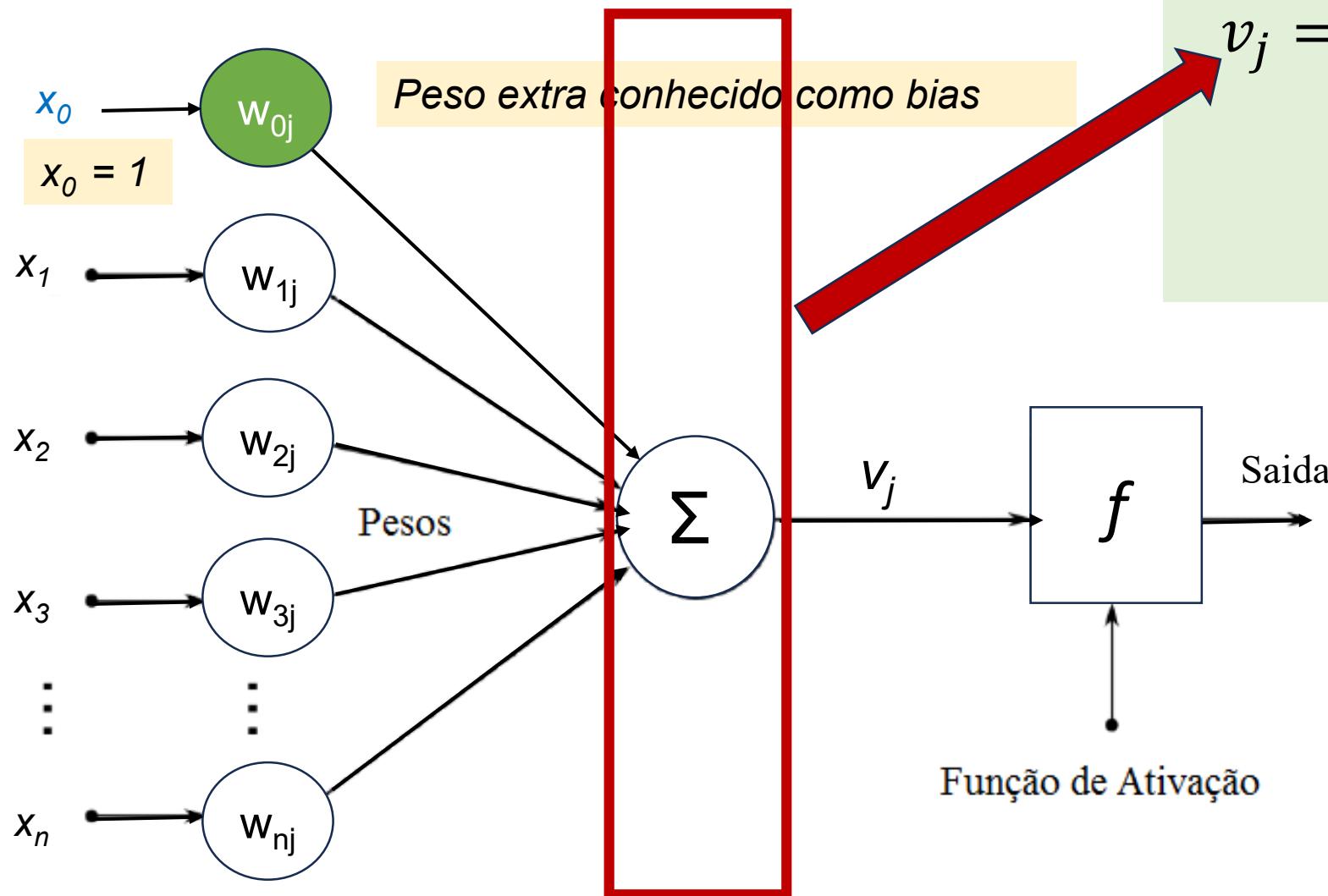


$$v_j = w_{0j} + \sum_{i=1}^n (x_i \times w_{ij})$$
$$y_j = f(v_j)$$

W: $w_1, w_2, w_3, \dots, w_p$
O vetor de pesos sinápticos
W. Esses valores ponderam as entradas simulando variações na intensidade do sinal.

Rede Perceptron

Revisitando um neurônio artificial j:

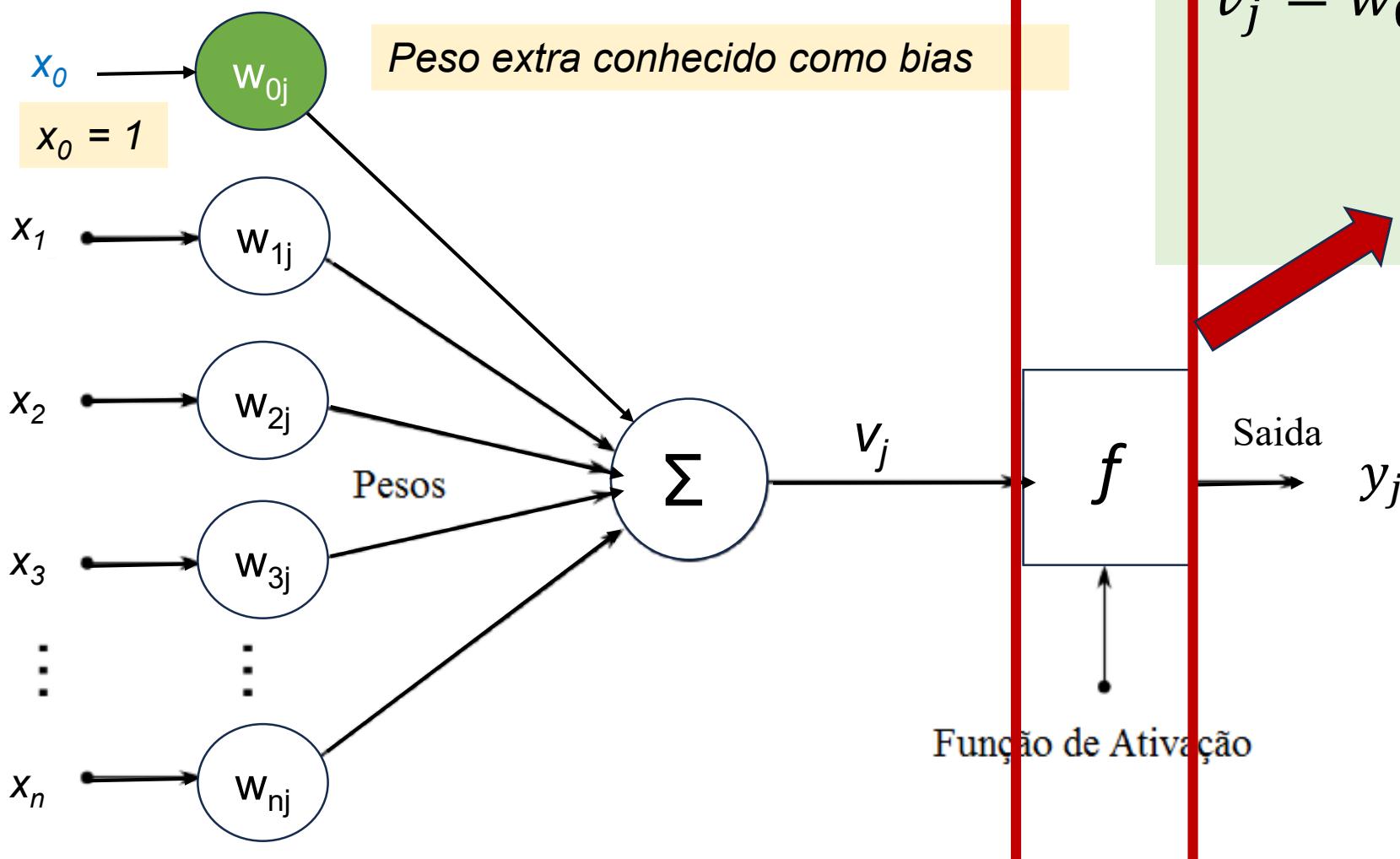


$$v_j = w_{0j} + \sum_{i=1}^n (x_i \times w_{ij})$$
$$y_j = f(v_j)$$

Σ : Responsável por combinar os valores de entrada. Realiza uma soma ponderada das entradas pelos pesos.

Rede Perceptron

Revisitando um neurônio artificial j:



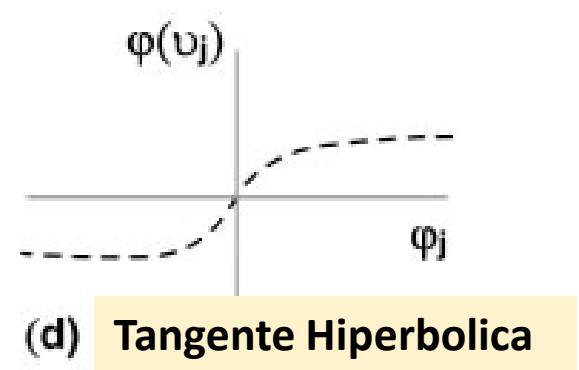
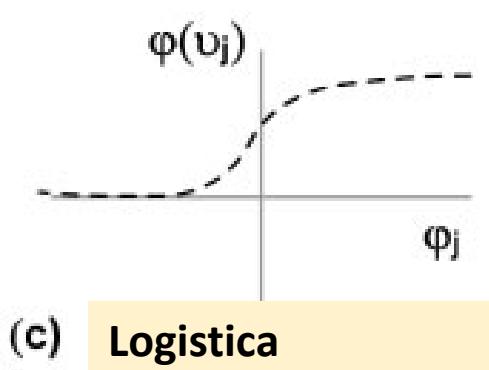
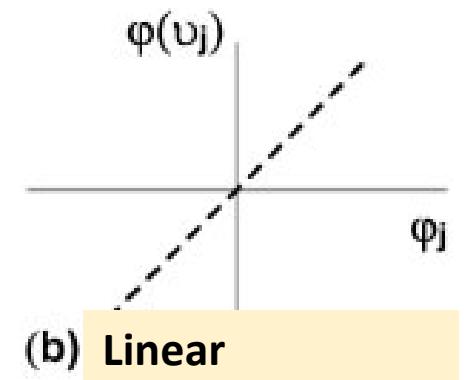
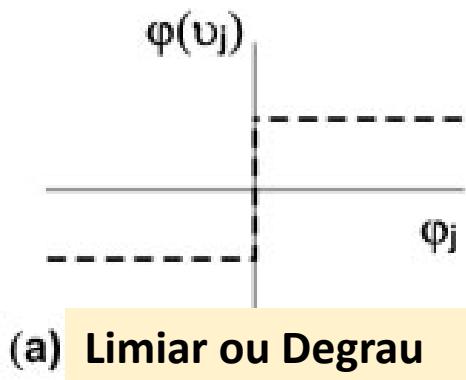
$$v_j = w_{0j} + \sum_{i=1}^n (x_i \times w_{ij})$$
$$y_j = f(v_j)$$

Função de transferência (ou ativação):
Responsável pela modulação do sinal propagado, gera valores excitatórios (positivos) ou inibitórios (zero).

Rede Perceptron

Revisitando um neurônio artificial j :

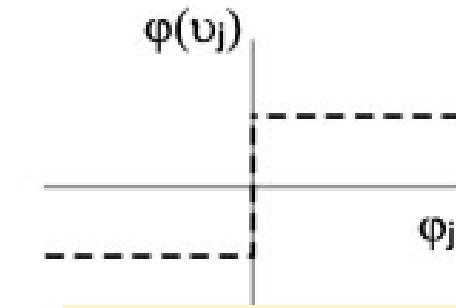
- Existem várias funções de ativação para as redes neurais.
- As clássicas são:
 - Limiar ou degrau
 - Linear
 - Sigmoid: logística ou tangente hiperbólica
- Geralmente o Perceptron usa a função limiar.



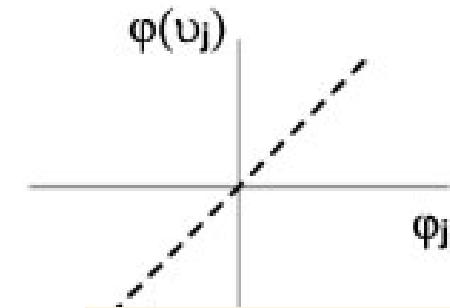
Rede Perceptron

Revisitando um neurônio artificial j :

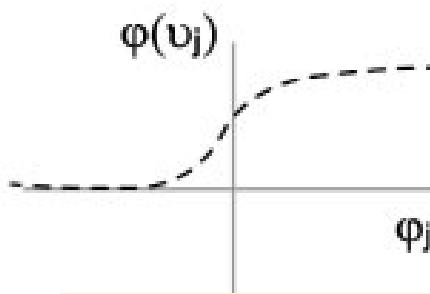
- Existem várias funções de ativação para as redes neurais.
- As clássicas são:
 - Limiar ou degrau
 - Linear
 - Sigmoid: logística ou tangente hiperbólica
- Geralmente o Perceptron usa a função limiar.



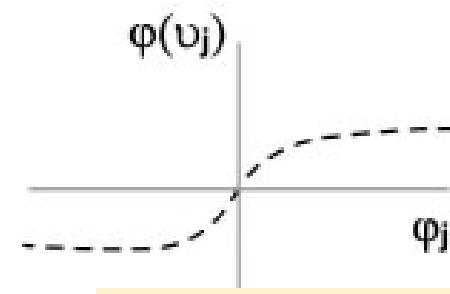
(a) Limiar ou Degrau



(b) Linear



(c) Logistica

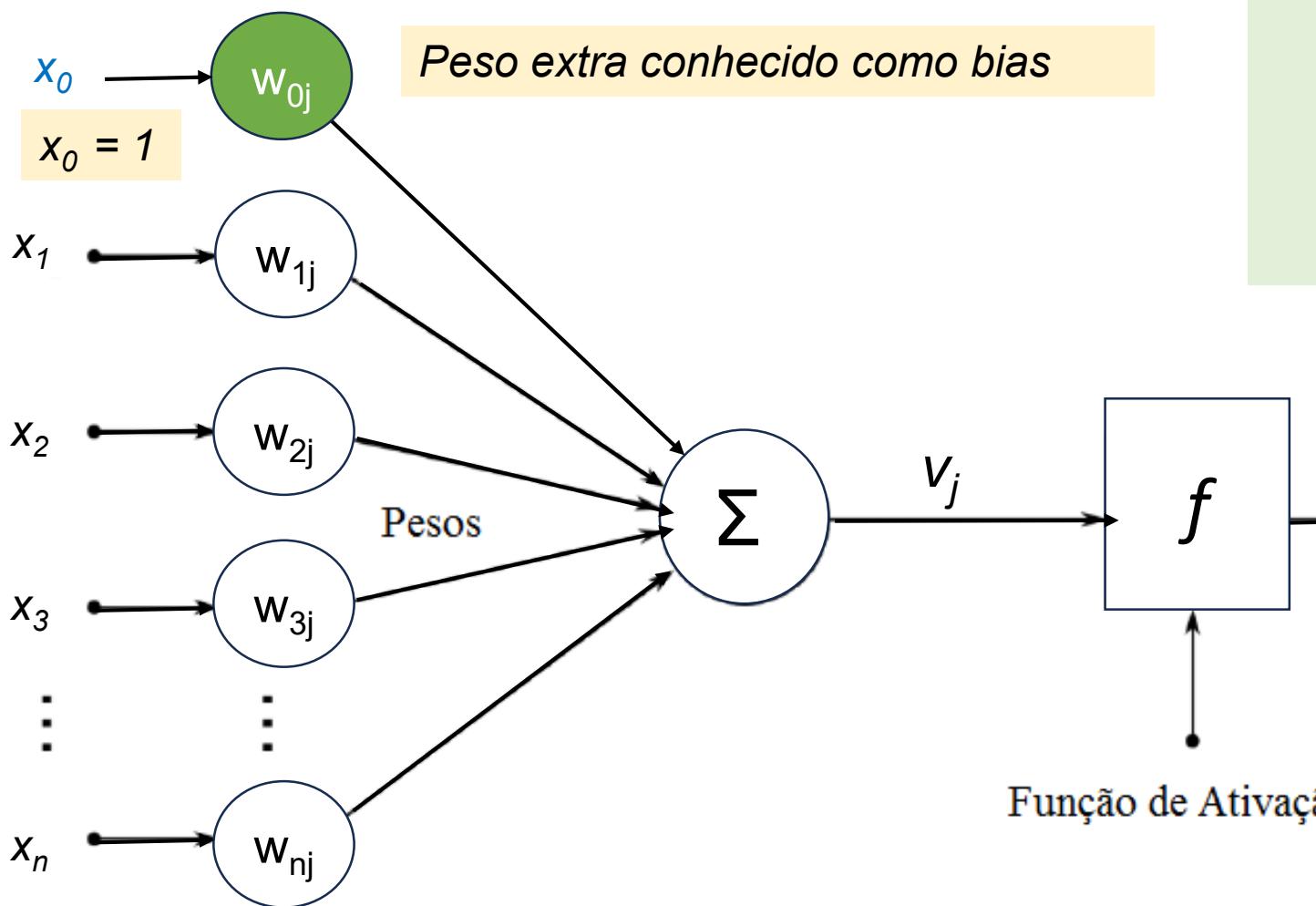


(d) Tangente Hiperbolica

Usuais em MLP

Rede Perceptron

Revisitando um neurônio artificial j:



$$v_j = w_{0j} + \sum_{i=1}^n (x_i \times w_{ij})$$
$$y_j = f(v_j)$$

Y:
Sinal de saída do
neurônio. Se o neurônio
estiver em uma camada
de saída, corresponde a
saída rede.

Função de Ativação

Rede Perceptron - Algoritmo de Treinamento

Algoritmo de Treinamento do Perceptron: usa Regra Delta (correção de erro)

- Sendo $X = \{ (\text{dados}1; \text{saídaDesejada}1); (\text{dados}1; \text{saídaDesejada}2), \dots \}$
- A taxa de aprendizagem η (deve ser positiva).

Iniciar todos pesos da rede com zero : $w_{ij} = 0$

Repetir ate encontrar erroGeral=0:

 epocas = epocas +1

 erroGeral = 0

 Para cada neurônio j: 1 a k da rede :

$v_j = 0$

 Para cada atributo i: 1 a n dos dados de X :

$v_j = v_j + x_i * w_{ij}$

$y_j = f(v_j)$

$\text{erro}_j = d_j - y_j$

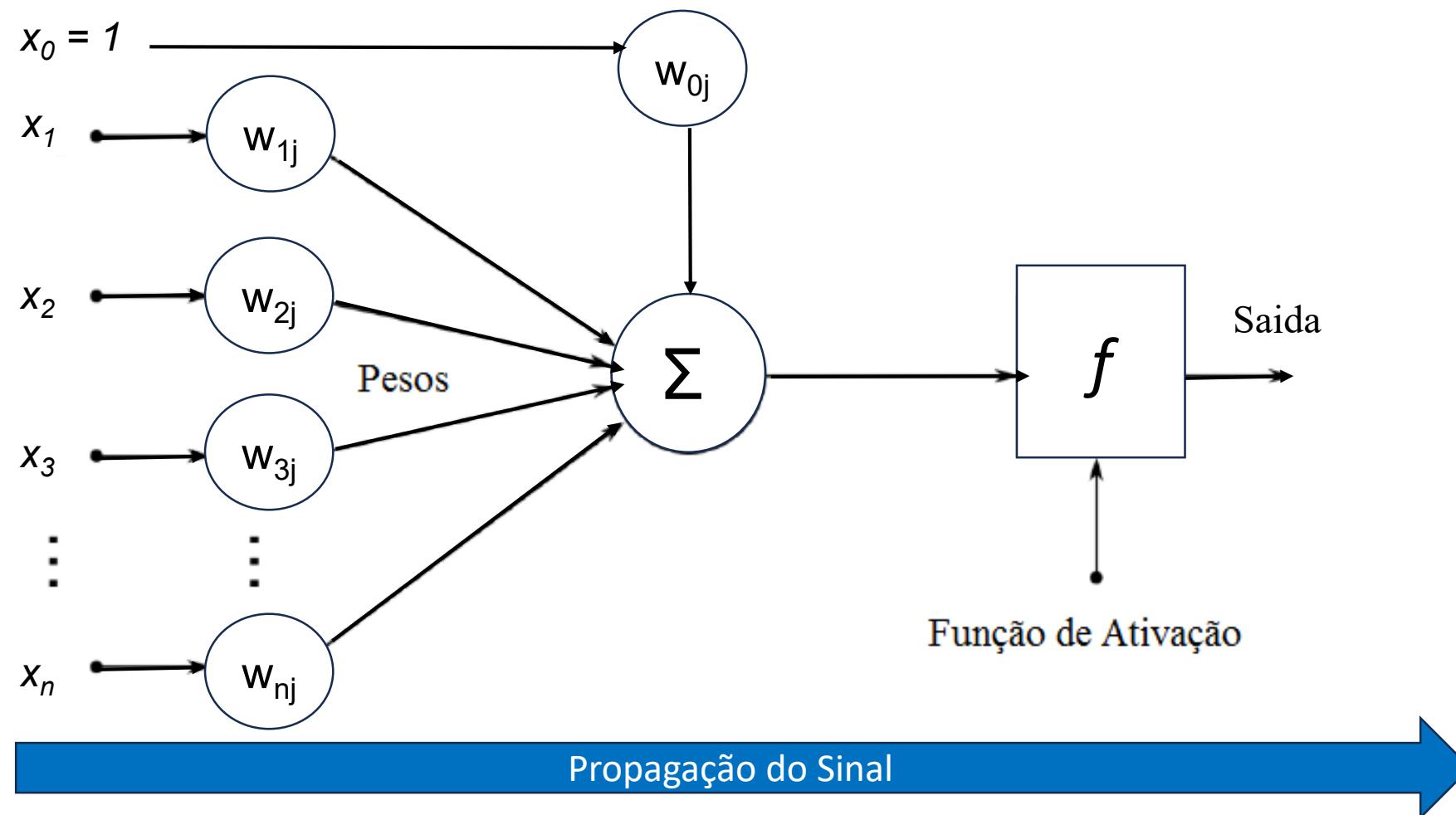
 se $\text{erro}_j \neq 0$ entao :

$\delta = \eta * \text{erro}_j * x_i$

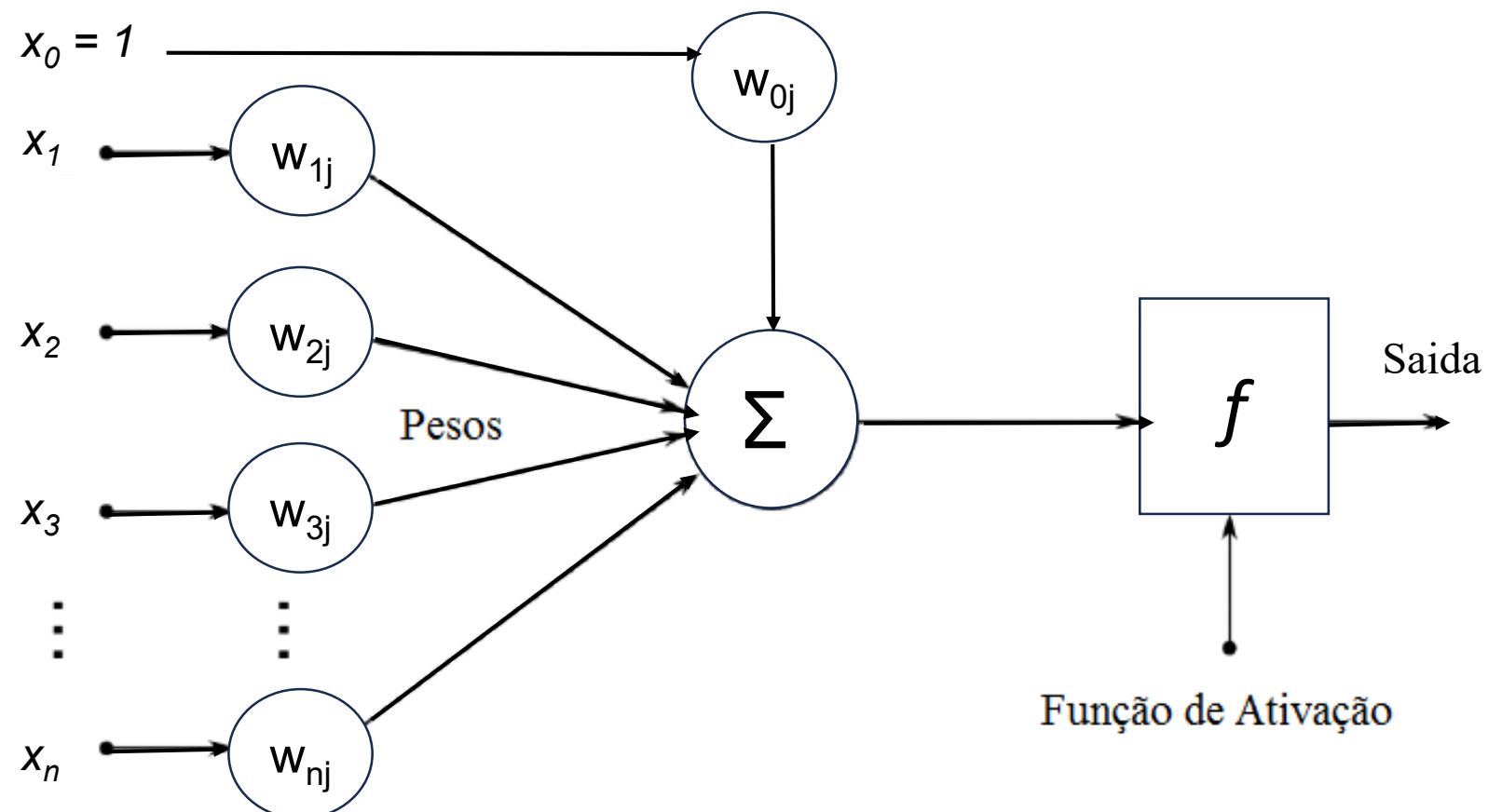
$w_{ij} = w_{ij} + \delta$

$\text{erroGeral} = \text{erroGeral} + \text{abs}(\text{erro}_j)$

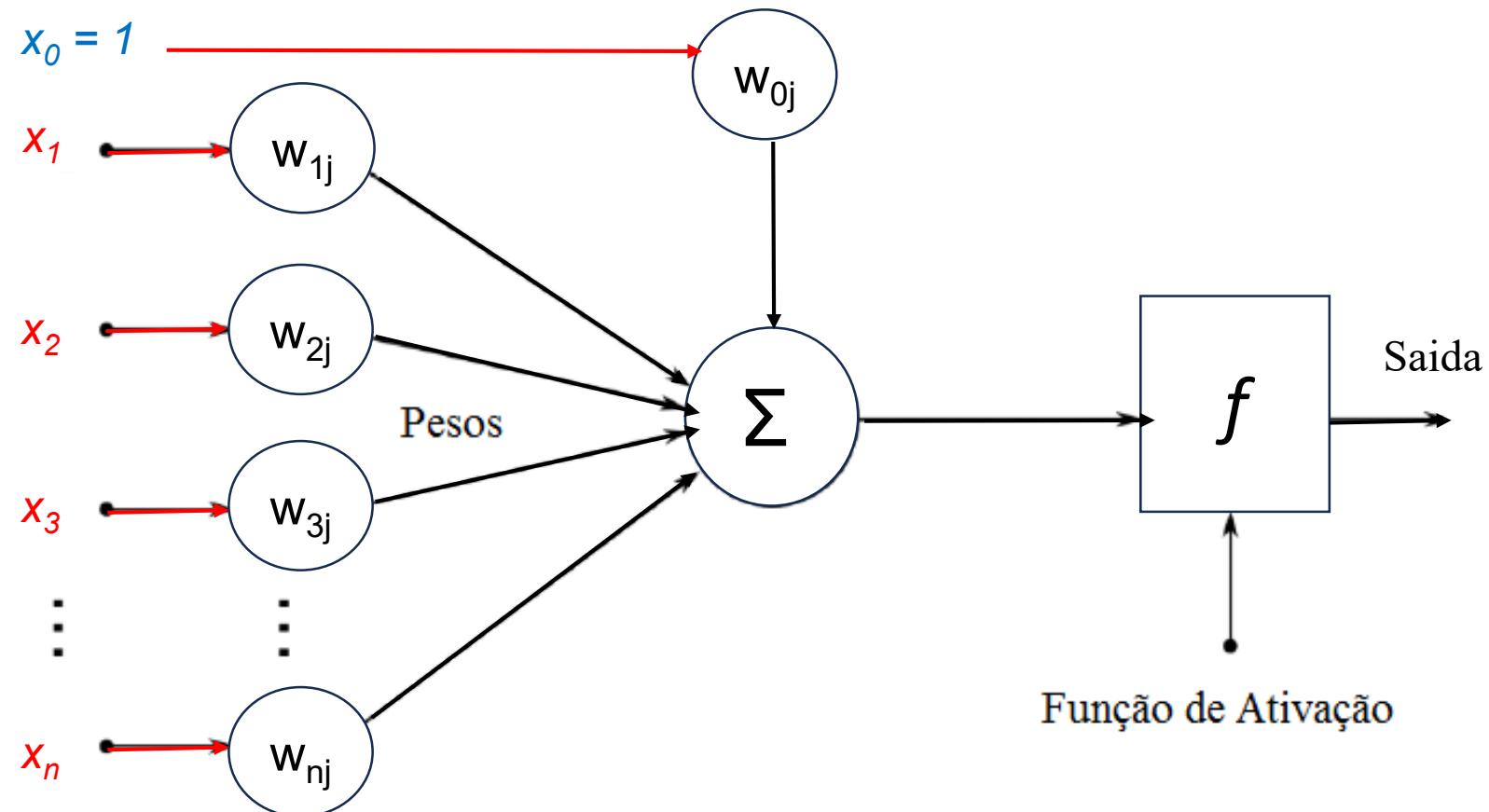
Rede Perceptron - Algoritmo de Treinamento



Dado um neurônio artificial j :

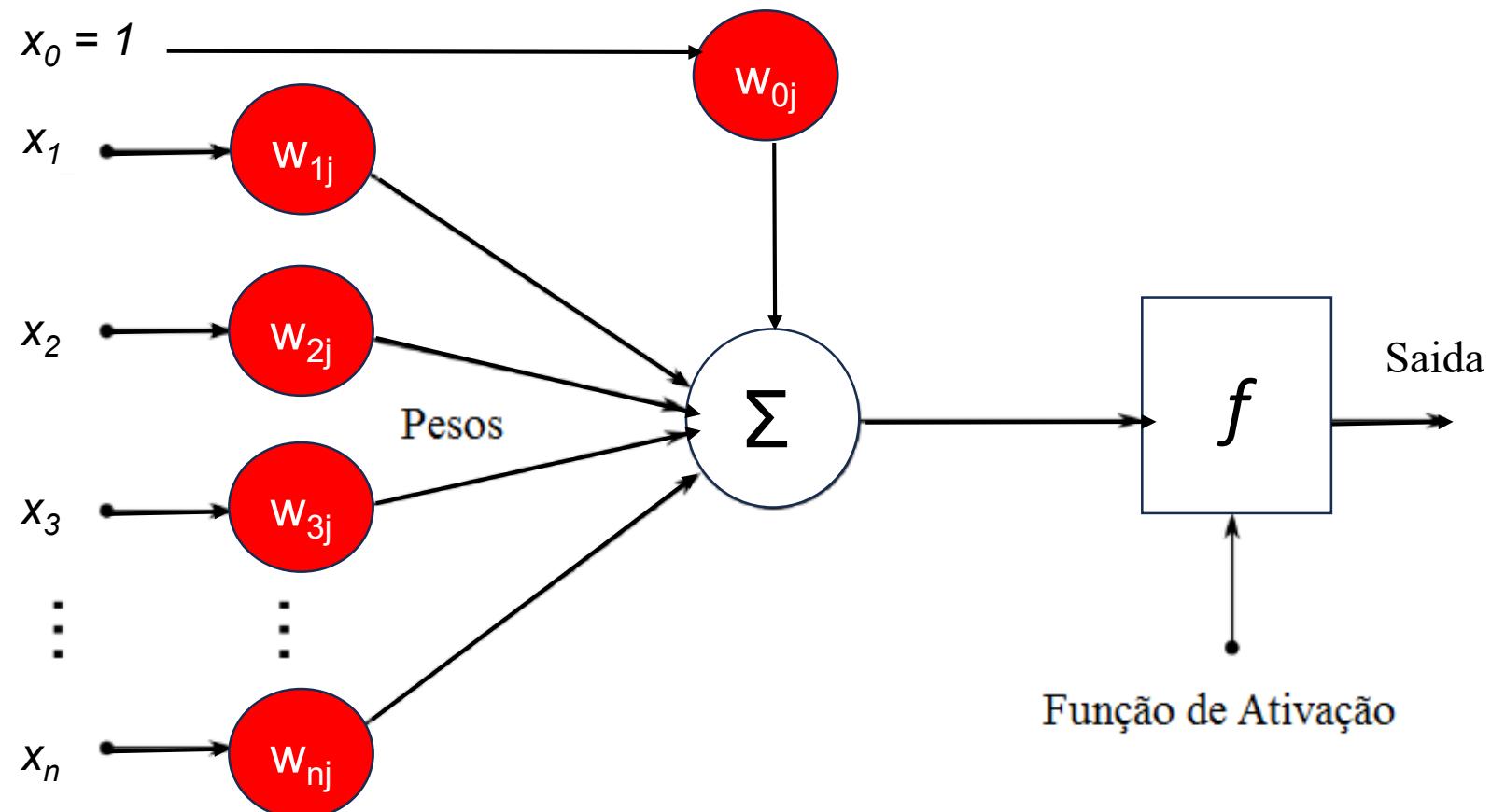


Dado um neurônio artificial j :



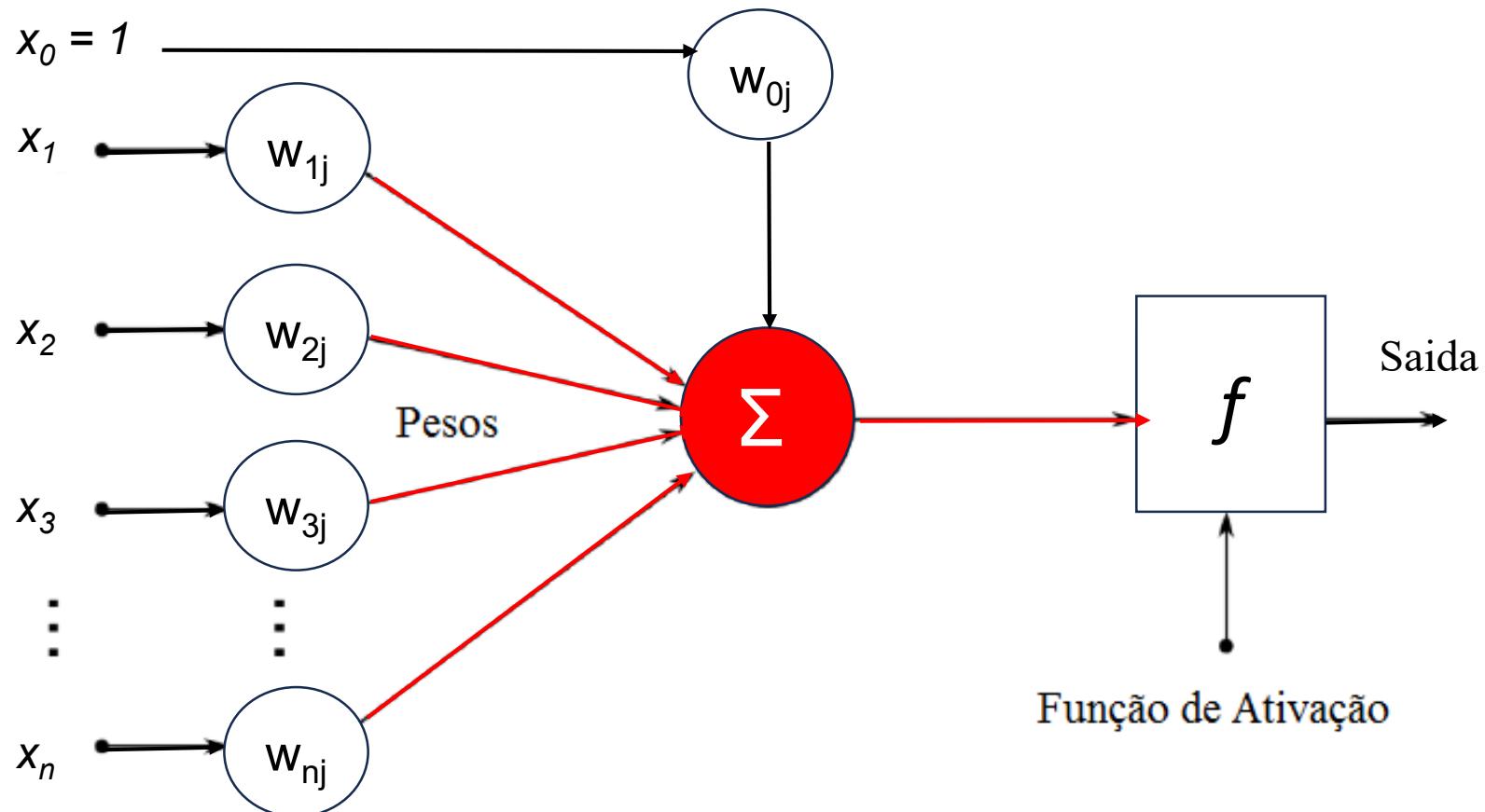
Recebendo os dados de entrada X

Dado um neurônio artificial j :



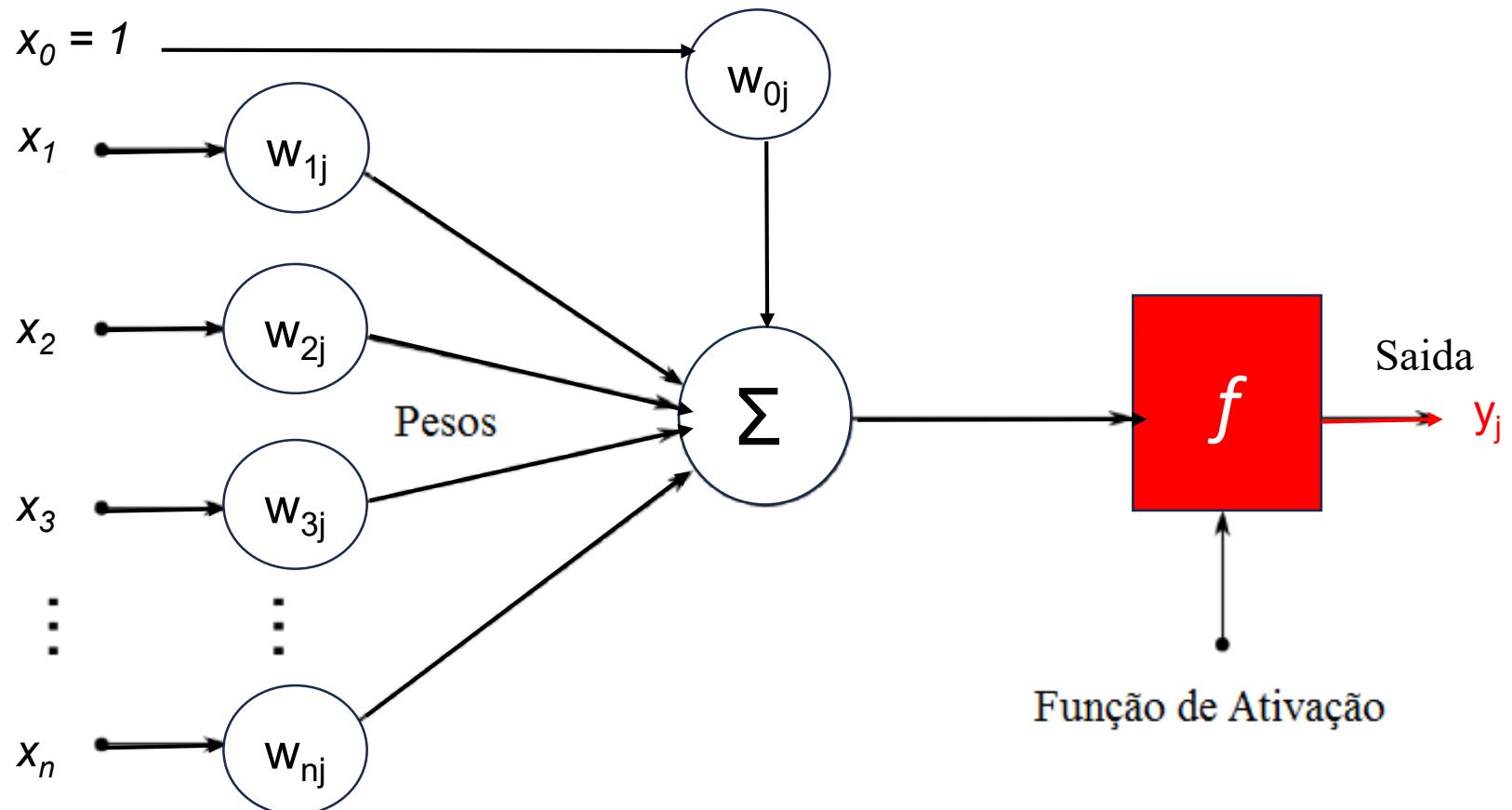
Ponderando as entradas X pelos pesos W : $x_i * w_{ij}$

Dado um neurônio artificial j:



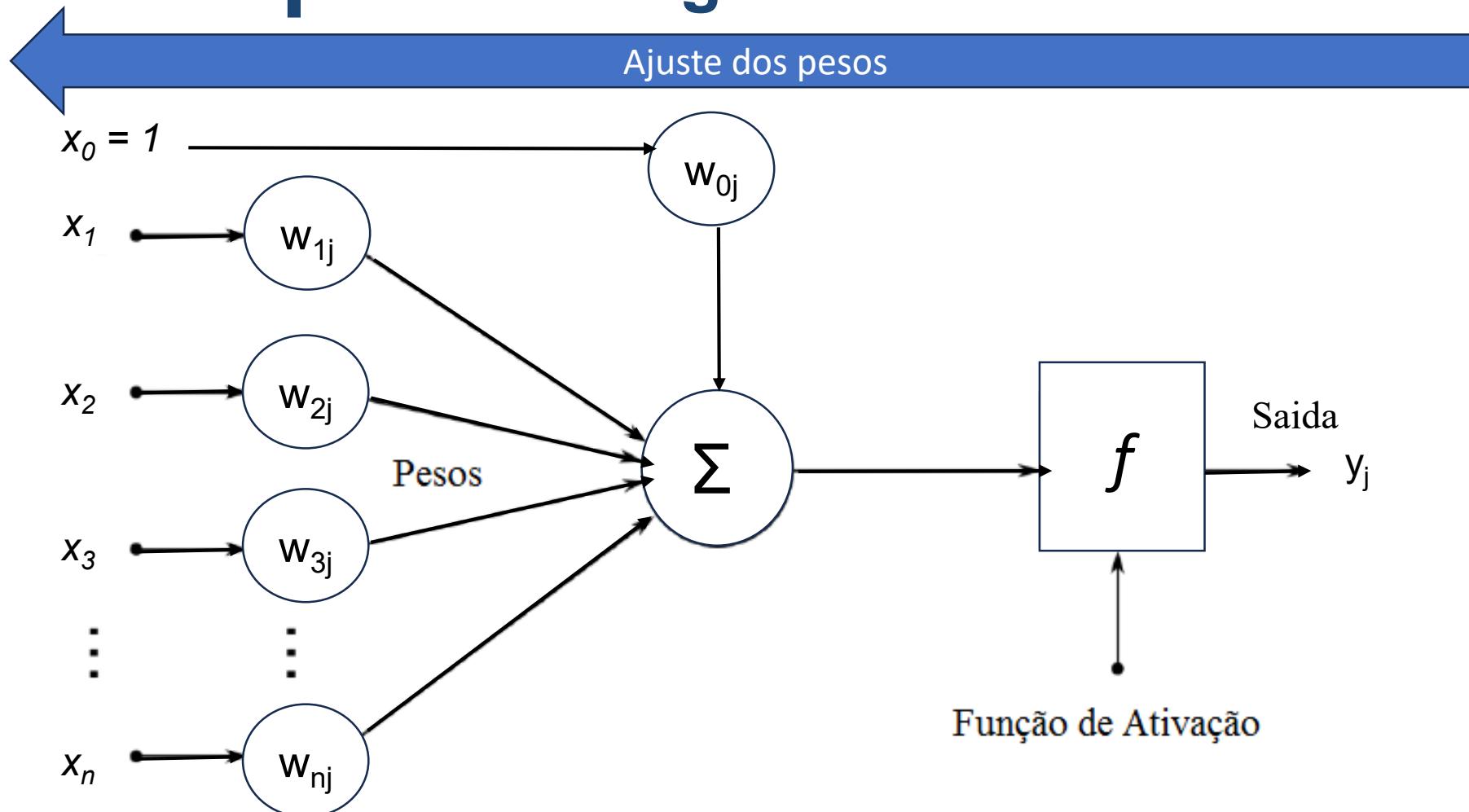
Combinando os sinais: soma= $w_{0j} + x_1 * w_{1j} + x_2 * w_{2j} + x_3 * w_{3j} + \dots x_n * w_{nj}$

Dado um neurônio artificial j :

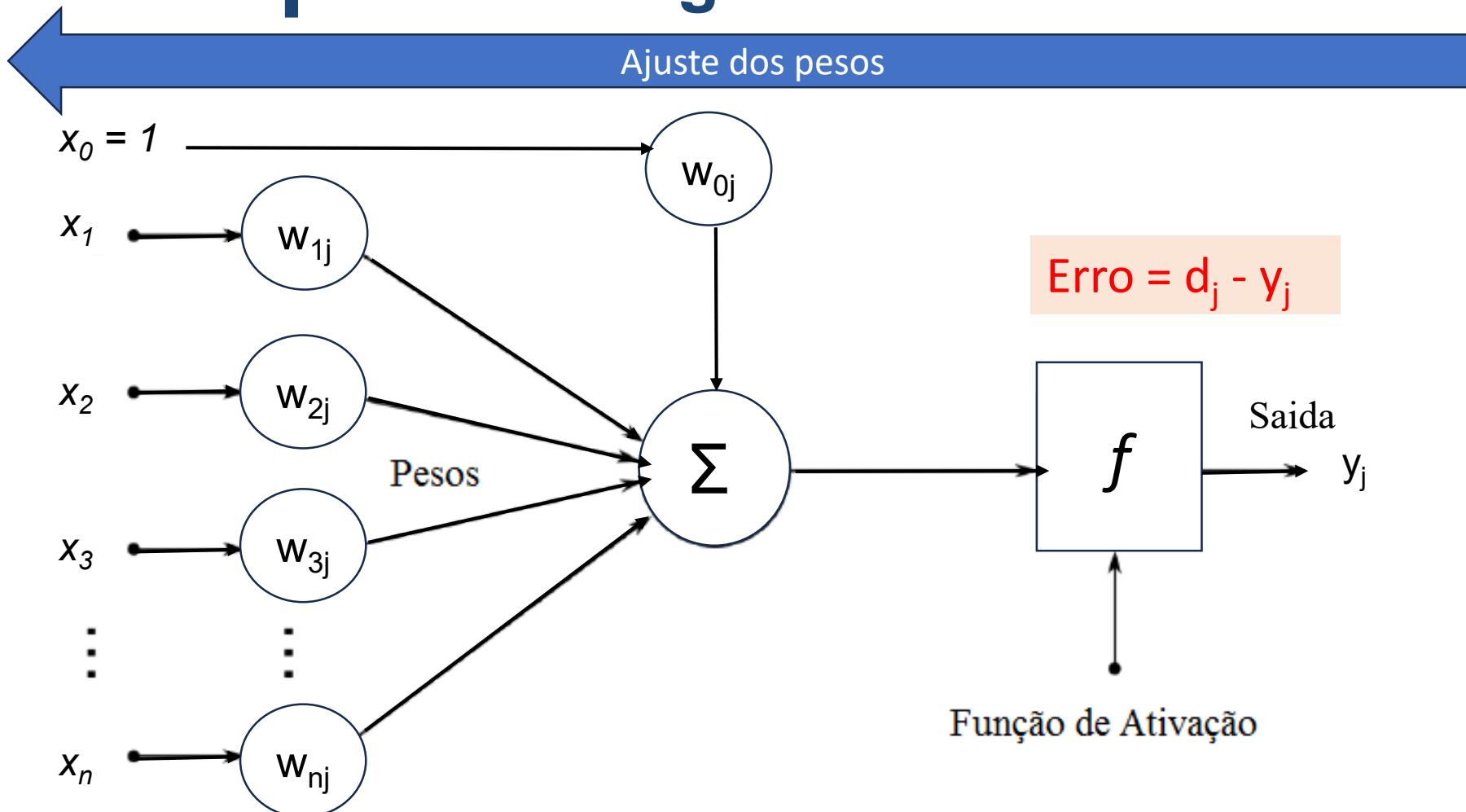


Gerando o sinal de saída: $y_j = f(\text{soma})$

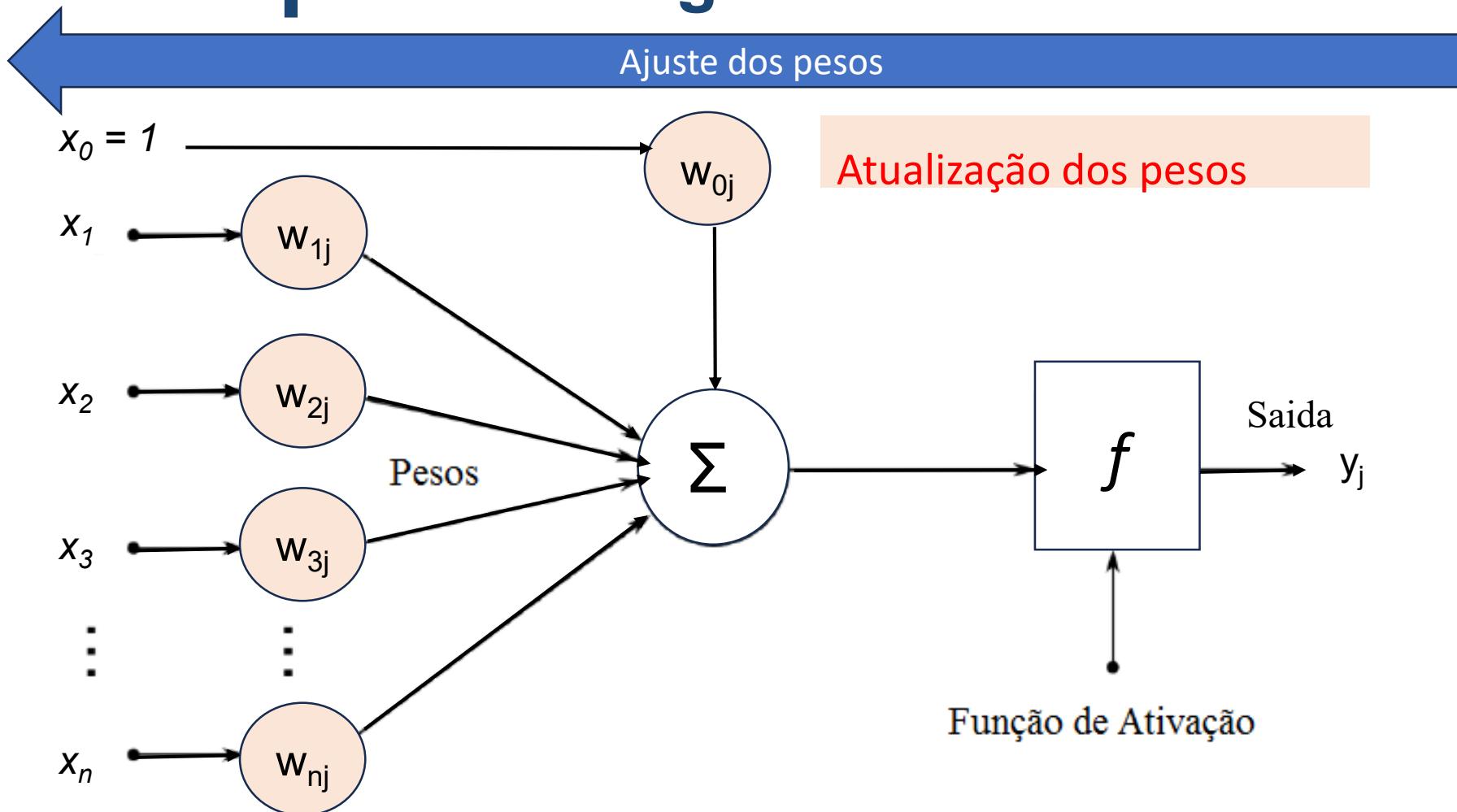
Rede Perceptron - Algoritmo de Treinamento



Rede Perceptron - Algoritmo de Treinamento



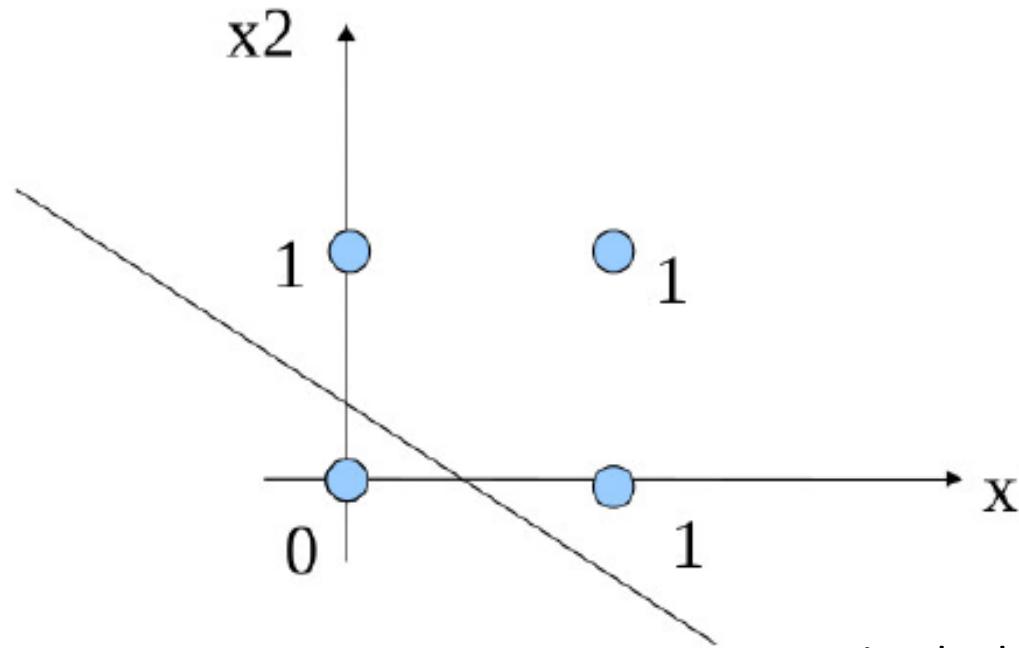
Rede Perceptron - Algoritmo de Treinamento



Rede Perceptron

Tabela OR

x1	x2	d
0	0	0
0	1	1
1	0	1
1	1	1



No treinamento, encontramos os pesos:

$$w_{10} = -1.0, w_{11} = 1.0, w_{12} = 1.0$$

Fronteira de decisão:

$$w_{11}x_1 + w_{12}x_2 + w_{10} = 0$$

$$1.0x_1 + 1.0x_2 - 1.0 = 0$$

$$x_1 + x_2 - 1.0 = 0$$

Algoritmo Backpropagation

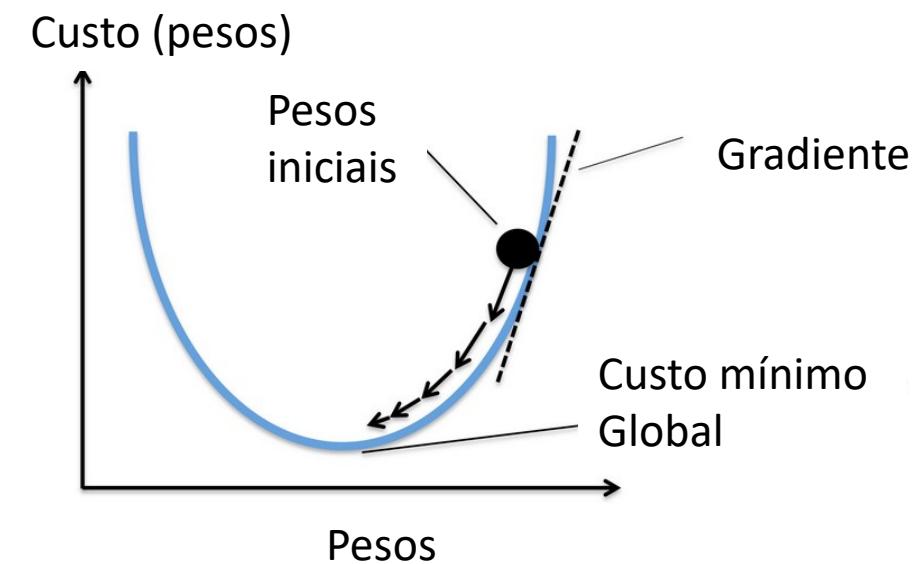


Algoritmo de Treinamento para MLP

- O algoritmo **Error Backpropagation** é usado para treinar a rede MLP.
- Foi introduzido na década de 70, mas sua importância só foi reconhecida **por volta de 1986**.
- Corresponde a uma **generalização da regra delta** usada no Perceptron.
- Este algoritmo tem importância também por ter **acabado com o pessimismo** sobre a técnica de redes neurais.
- **Caixa preta:** Na mesma proporção que ajudou, a introdução das camadas ocultas deixaram ainda mais difícil de entender o papel de cada elemento da rede.

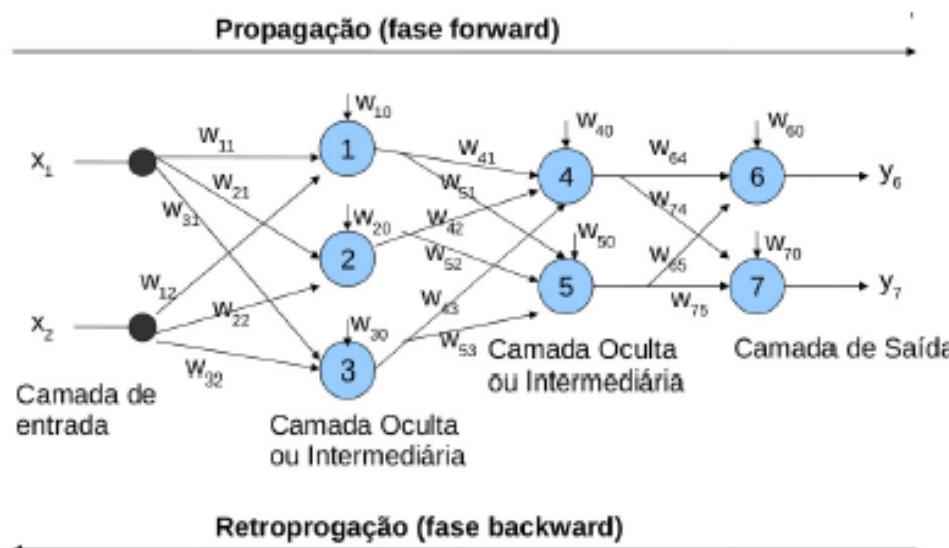
Algoritmo de Treinamento para MLP

- O algoritmo **Error Backpropagation** usa gradiente descendente.
- O método gradiente descendente consiste em encontrar, **de forma iterativa**, os valores dos parâmetros que minimizam determinada função de interesse.
- No caso, consiste em encontrar os pesos que minimizam o erro da rede.



Algoritmo de Treinamento para MLP

- O backpropagation possui 2 etapas: **forward** e **backward**. Em cada etapa a rede é percorrida em um sentido.
 - A fase **forward (para frente) – propagação** - é utilizada para definir a saída da rede para um dado de entrada.
 - A fase **backward (para trás) – retropropagação** - utiliza a saída desejada e a saída gerada pela rede para atualizar os pesos de suas conexões.



Algoritmo de Treinamento para MLP

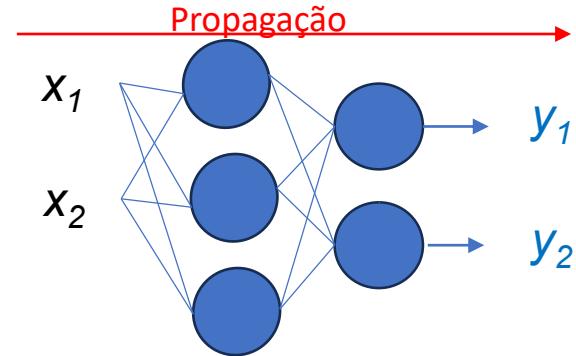
Cada neurônio da rede foi projetado para realizar dois cálculos:

- **Cálculo do sinal de saída:** corresponde a saída y de cada neurônio, expresso por uma função não linear aplicada à soma ponderada dos pesos pelas entradas em cada neurônio. Acontece na etapa de propagação.
- **Cálculo de uma estimativa do vetor gradiente:** corresponde os gradientes da superfície do erro em relação aos pesos conectados às entradas de um neurônio. Os gradientes são usados na etapa de retropropagação.

Algoritmo de Treinamento para MLP

- Considere que a topologia da rede já está definida e que há um conjunto de Treino com N pares (X, D) , onde:
 - X é o conjunto de entrada: $\{x_1, x_2, x_3, x_4, \dots, x_Z\}$, com Z entradas (atributos)
 - D é o conjunto de saídas desejadas: $\{d_1, d_2, d_3, d_4, \dots, x_M\}$, com M saídas (uma para cada neurônio da camada de saída)

Algoritmo de Treinamento para MLP



1. Iniciar **os pesos da rede arbitrariamente** com valores não nulos.
2. Apresentar **cada dado n de entrada do conjunto de treino e propagá-lo até a saída** da rede (geração dos y 's da camada de saída), onde $n = 1$ até N .
3. **Propagação:** Para cada neurônio k e entrada i .

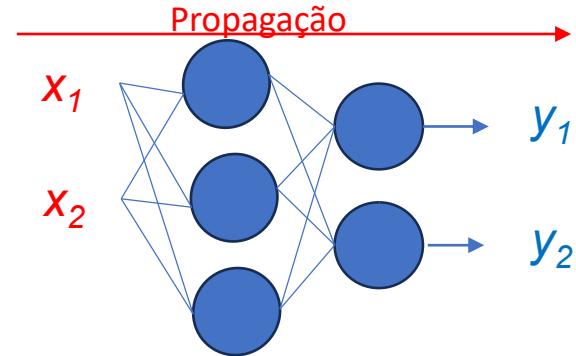
$$v_k(n) = \sum_{i=0}^z (w_{ki}(n) \times x_i(n)) \quad y_k(n) = Q(v_k(n))$$

4. **Retropropagação:** Para cada neurônio k : $\text{erro}_k = d_k - y_k$

$$\xi(n) = \frac{1}{2} \sum_{k=1}^j \text{erro}_k^2$$

Energia do erro instantâneo: combina os erros de todos neurônios da camada de saída para a entrada propagada.

Algoritmo de Treinamento para MLP



1. Iniciar **os pesos da rede arbitrariamente** com valores não nulos.
2. Apresentar **cada dado n de entrada do conjunto de treino e propagá-lo até a saída** da rede (geração dos y 's da camada de saída), onde $n = 1$ até N .
3. **Propagação:** Para cada neurônio k e entrada i .

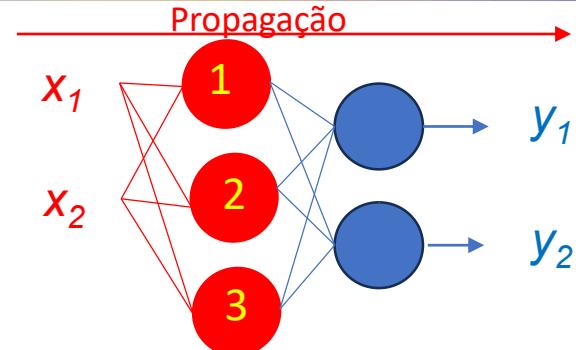
$$v_k(n) = \sum_{i=0}^z (w_{ki}(n) \times x_i(n)) \quad y_k(n) = Q(v_k(n))$$

4. **Retropropagação:** Para cada neurônio k : $\text{erro}_k = d_k - y_k$

$$\xi(n) = \frac{1}{2} \sum_{k=1}^j \text{erro}_k^2$$

Energia do erro instantâneo: combina os erros de todos neurônios da camada de saída para a entrada propagada.

Algoritmo de Treinamento para MLP



1. Iniciar **os pesos da rede arbitrariamente** com valores não nulos.
2. Apresentar **cada dado n de entrada do conjunto de treino e propagá-lo até a saída** da rede (geração dos y 's da camada de saída), onde $n = 1$ até N .
3. **Propagação:** Para cada neurônio k e entrada i .

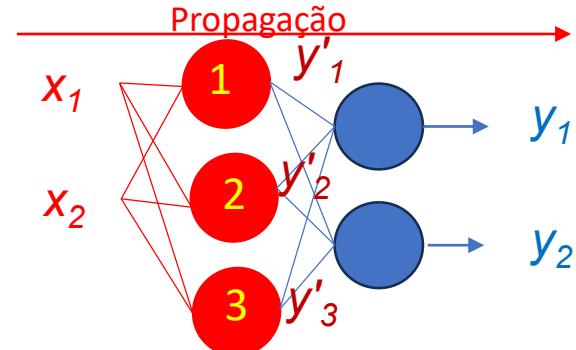
$$v_k(n) = \sum_{i=0}^z (w_{ki}(n) \times x_i(n)) \quad y_k(n) = Q(v_k(n))$$

4. **Retropropagação:** Para cada neurônio k : $\text{erro}_k = d_k - y_k$

$$\xi(n) = \frac{1}{2} \sum_{k=1}^j \text{erro}_k^2$$

Energia do erro instantâneo: combina os erros de todos neurônios da camada de saída para a entrada propagada.

Algoritmo de Treinamento para MLP



1. Iniciar **os pesos da rede arbitrariamente** com valores não nulos.
2. Apresentar **cada dado n de entrada do conjunto de treino e propagá-lo até a saída** da rede (geração dos y's da camada de saída), onde n = 1até N.
3. **Propagação:** Para cada neurônio k e entrada i.

$$v_k(n) = \sum_{i=0}^z (w_{ki}(n) \times x_i(n))$$

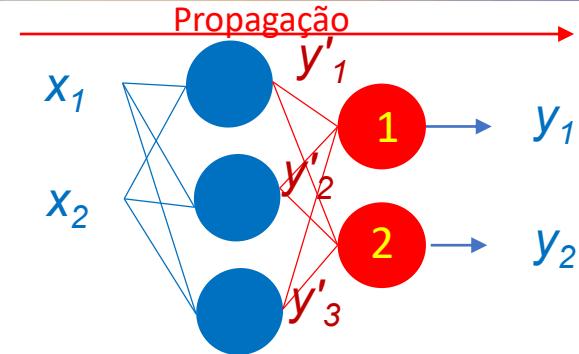
$$y_k(n) = Q(v_k(n))$$

4. **Retropropagação:** Para cada neurônio k: $\text{erro}_k = d_k - y_k$

$$\xi(n) = \frac{1}{2} \sum_{k=1}^j \text{erro}_k^2$$

Energia do erro instantâneo: combina os erros de todos neurônios da camada de saída para a entrada propagada.

Algoritmo de Treinamento para MLP



1. Iniciar **os pesos da rede arbitrariamente** com valores não nulos.
2. Apresentar **cada dado n de entrada do conjunto de treino e propagá-lo até a saída** da rede (geração dos y 's da camada de saída), onde $n = 1$ até N .
3. **Propagação:** Para cada neurônio k e entrada i .

$$v_k(n) = \sum_{i=0}^z (w_{ki}(n) \times x_i(n))$$

$$y_k(n) = Q(v_k(n))$$

4. **Retropropagação:** Para cada neurônio k : $\text{erro}_k = d_k - y_k$

$$\xi(n) = \frac{1}{2} \sum_{k=1}^j \text{erro}_k^2$$

Energia do erro instantâneo: combina os erros de todos neurônios da camada de saída para a entrada propagada.

Algoritmo de Treinamento para MLP

1. Iniciar **os pesos da rede arbitrariamente** com valores não nulos.
2. Apresentar **cada dado n de entrada do conjunto de treino e propagá-lo até a saída** da rede (geração dos y's da camada de saída), onde n = 1até N.
3. **Propagação:** Para cada neurônio k e entrada i.

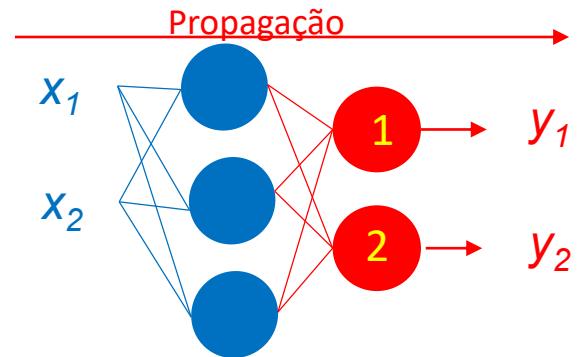
$$v_k(n) = \sum_{i=0}^z (w_{ki}(n) \times x_i(n))$$

$$y_k(n) = Q(v_k(n))$$

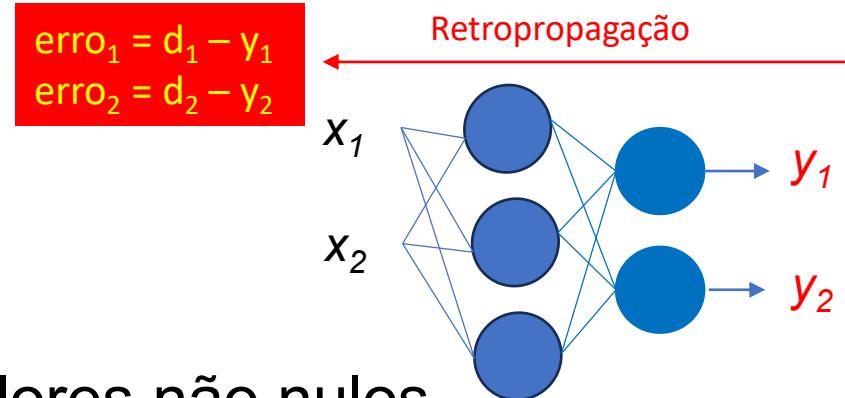
4. **Retropropagação:** Para cada neurônio k: $\text{erro}_k = d_k - y_k$

$$\xi(n) = \frac{1}{2} \sum_{k=1}^j \text{erro}_k^2$$

Energia do erro instantâneo: combina os erros de todos neurônios da camada de saída para a entrada propagada.



Algoritmo de Treinamento para MLP



1. Iniciar **os pesos da rede arbitrariamente** com valores não nulos.
2. Apresentar **cada dado n de entrada do conjunto de treino e propagá-lo até a saída** da rede (geração dos y 's da camada de saída), onde $n = 1$ até N .
3. **Propagação:** Para cada neurônio k e entrada i .

$$v_k(n) = \sum_{i=0}^z (w_{ki}(n) \times x_i(n)) \quad y_k(n) = Q(v_k(n))$$

4. **Retropropagação:** Para cada neurônio k : $\text{erro}_k = d_k - y_k$

$$\xi(n) = \frac{1}{2} \sum_{k=1}^j \text{erro}_k^2$$

Energia do erro instantâneo: combina os erros de todos neurônios da camada de saída para a entrada propagada.

Algoritmo de Treinamento para MLP

4. Retropropagação:

...

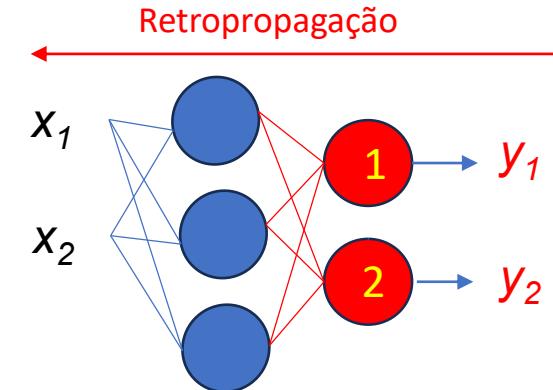
- Calcular os **gradientes da camada de saída**: $\delta_k(n) = Q'(v_k(n)) \times erro_k(n)$

- Atualizar os pesos dos neurônios

$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

Exemplos: $\eta = \{0.01, 0.1, 0.5,$

- Quanto menor o valor de η , menor será a variação dos pesos da rede de uma iteração para outra e mais suave será e lenta a trajetória no espaço de busca pelos pesos;
- Quanto maior o valor de η , haverá maiores modificações nos pesos da rede de uma iteração para outra e mais rápido será a trajetória no espaço de busca pelos pesos; porém isso não garante uma boa aprendizagem, a rede pode se tornar instável e oscilar.



onde η é a taxa de aprendizado (learning rate) intervalo típico $(0,1]$

Algoritmo de Treinamento para MLP

4. Retropropagação: onde α é a constante de momento (momentum rate) intervalo típico
...

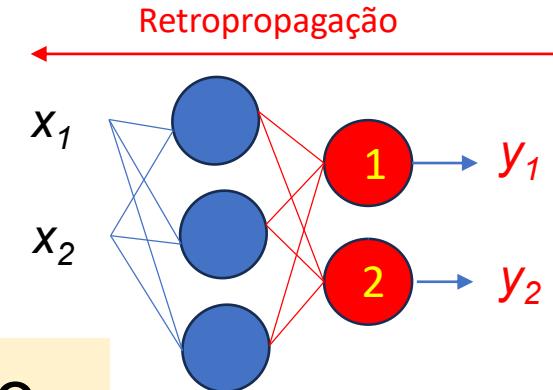
- Calcular os **gradientes** da **camada de saída**:

- Atualizar os pesos dos neurônios $\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$

$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$

Exemplos: $\alpha = \{0.0, 0.1, 0.5,$

- Para evitar o perigo de instabilidade do algoritmo, foi introduzido uma **constante de momento** (momentum rate) : α
- Ela controla e estabiliza a atualização dos pesos.
- Evita que mínimos locais
- Acelera a descida do gradiente em direções com declividade constante.



Algoritmo de Treinamento para MLP

4. Retropropagação:

...

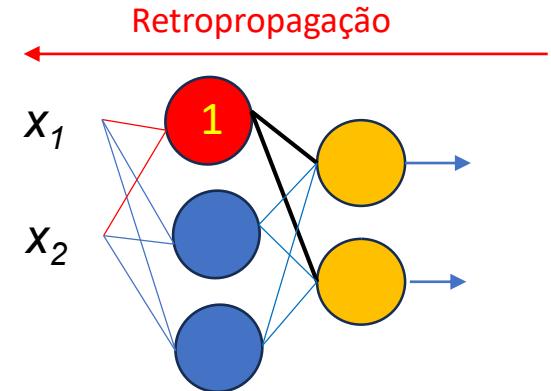
- Calcular os **gradiêntes das camadas ocultas**:

$$\delta_k(n) = Q'(v_k(n)) * \sum_{i=1}^t (\delta_j(n) * w_{jk}(n+1))$$

- Atualizar os pesos dos neurônios

$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$



onde j são os neurônios com os quais o neurônio k tem conexão à direita.

Como a camada à direita já sofreu retropropagação, seus pesos já foram atualizados, por isso aparece n+1.

Algoritmo de Treinamento para MLP

4. Retropropagação:

...

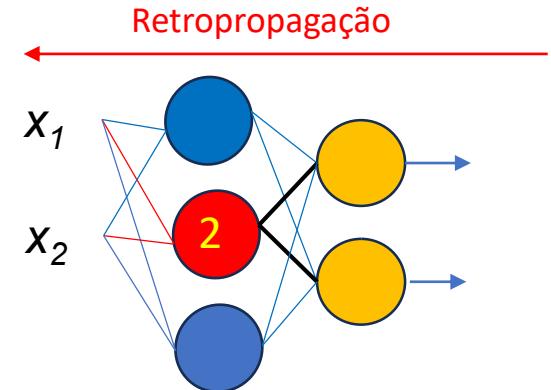
- Calcular os **gradientes das camadas ocultas**:

$$\delta_k(n) = Q'(v_k(n)) * \sum_{i=1}^t (\delta_j(n) * w_{jk}(n+1))$$

- Atualizar os pesos dos neurônios

$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$



onde j são os neurônios com os quais o neurônio k tem conexão à direita.

Como a camada à direita já sofreu retropropagação, seus pesos já foram atualizados, por isso aparece n+1.

Algoritmo de Treinamento para MLP

4. Retropropagação:

...

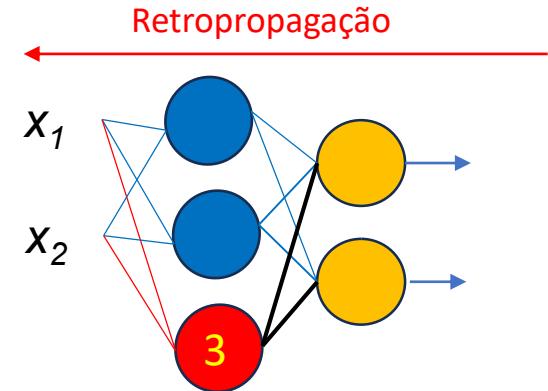
- Calcular os **gradientes das camadas ocultas**:

$$\delta_k(n) = Q'(v_k(n)) * \sum_{i=1}^t (\delta_j(n) * w_{jk}(n+1))$$

- Atualizar os pesos dos neurônios

$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$



onde j são os neurônios com os quais o neurônio k tem conexão à direita.

Como a camada à direita já sofreu retropropagação, seus pesos já foram atualizados, por isso aparece n+1.

Algoritmo de Treinamento para MLP

- Ao final de uma época (passagem de todos os dados do conjunto de treino pela rede)
- Calcular o Erro Médio Quadrado (função de custo)

$$EMQ = \left(\sum_{i=1}^N \xi_i \right) / N$$

Corresponde ao erro médio encontrado para todo o conjunto de treino

É usado como critério de parada. Pode oscilar no início da aprendizagem, mas deve decrescer ao longo do treinamento.

$$\xi(n) = \frac{1}{2} \sum_{k=1}^j \text{erro}_k^2(n)$$

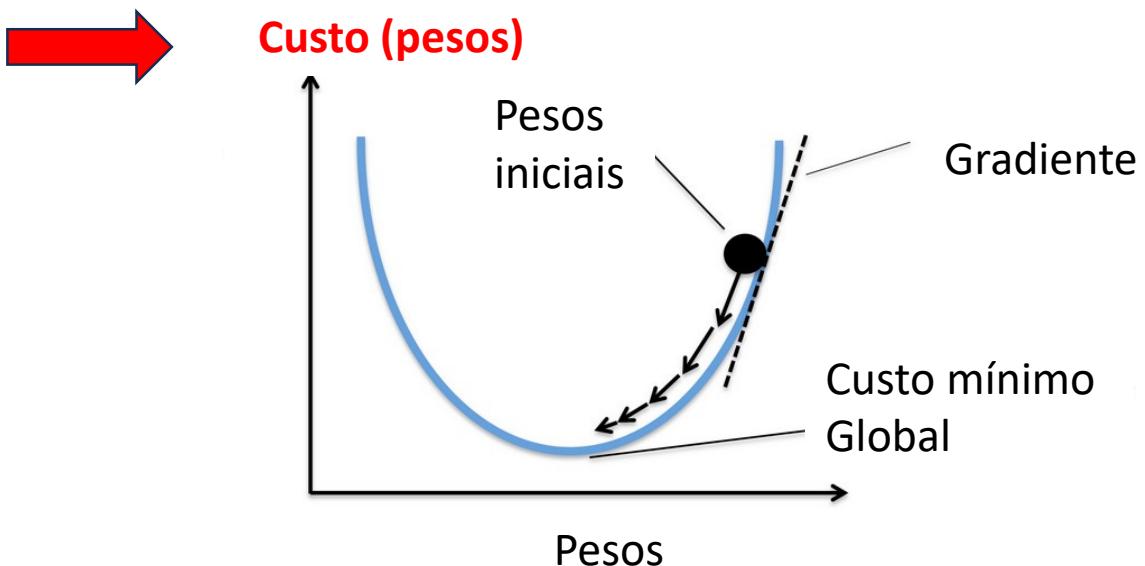
Lembrando que a energia do erro corresponde à combinação dos erros dos neurônios da camada de saída para uma entrada n .

Algoritmo de Treinamento para MLP

- Ao final de uma época (passagem de todos os dados do conjunto de treino pela rede)
- Calcular o Erro Médio Quadrado (função de custo)

$$EMQ = \left(\sum_{i=1}^N \xi_i \right) / N$$

Corresponde ao erro médio encontrado para todo o conjunto de treino

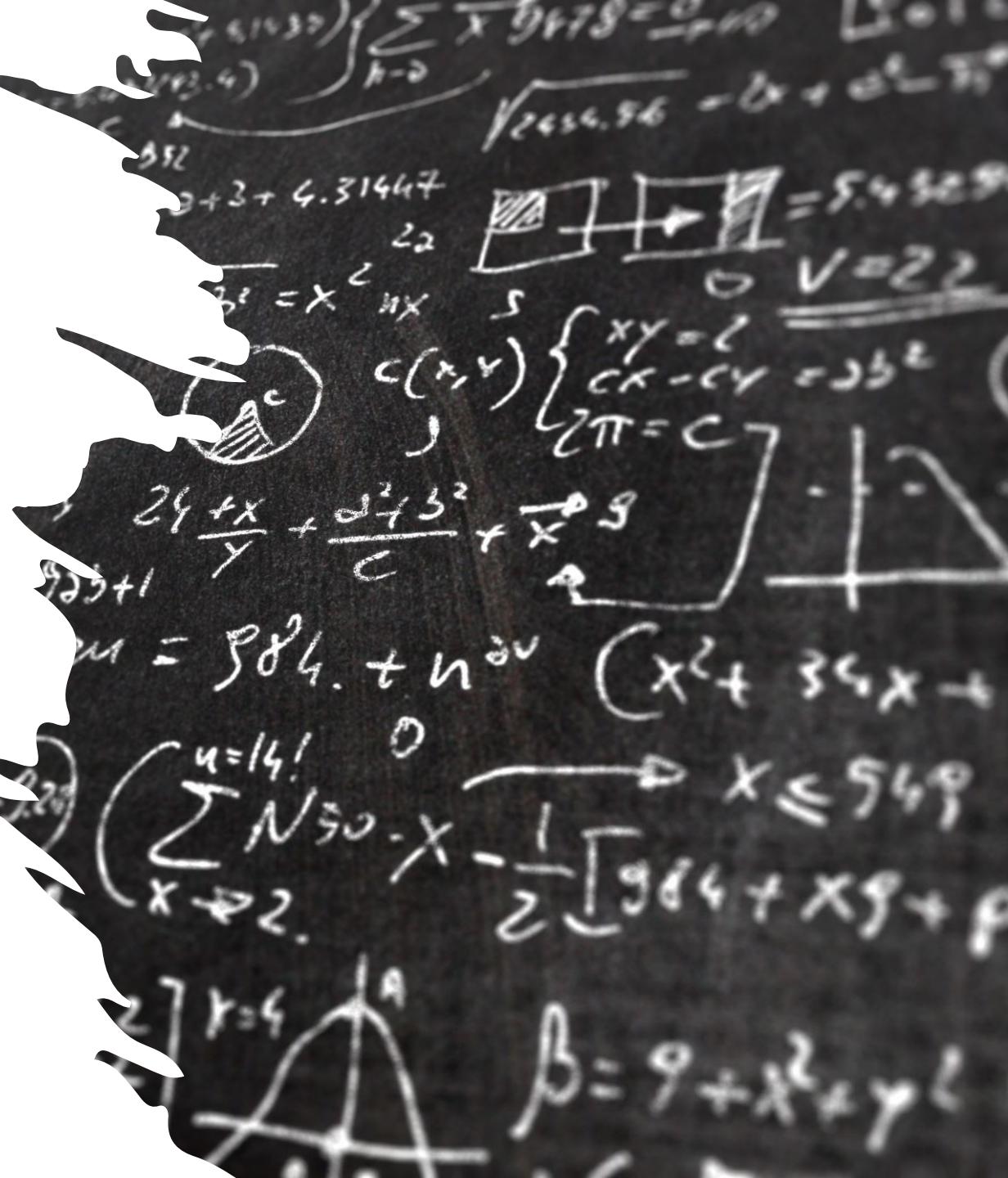


Algoritmo de Treinamento para MLP

Critérios de parada do algoritmo backpropagation:

- número pré-definido de épocas;
- valor pré-definido para o erro quadrado médio;
- variação do erro médio nas últimas x épocas inferior a um valor pré-definido (convergência);
- número de padrões corretamente classificados não se alterar;
- combinação desses critérios.

Revisitando as funções de ativação



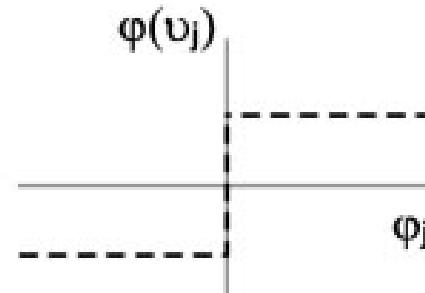
Algoritmo de Treinamento para MLP

- **Funções de ativação (ou de transferência):**
 - Modelos de redes neurais modernas usam funções de ativação não lineares.
 - Permitem que o modelo crie mapeamentos complexos entre as entradas e saídas da rede: essenciais para o aprendizado de dados complexos, como imagens, vídeo, áudio e conjuntos de dados que não são lineares ou têm alta dimensionalidade.

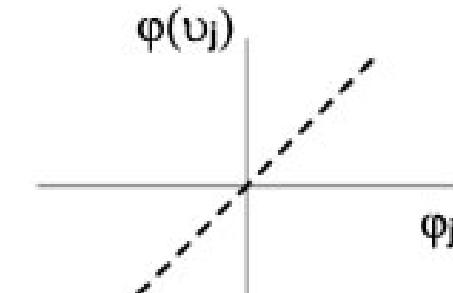
Algoritmo de Treinamento para MLP

- Existem várias funções de ativação para as redes neurais.

Perceptron

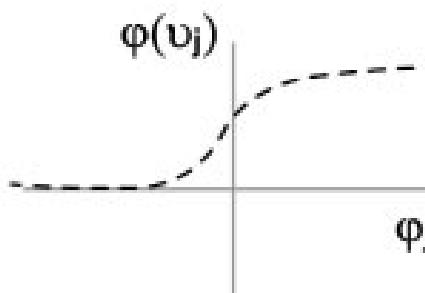


(a) Limiar ou Degrau

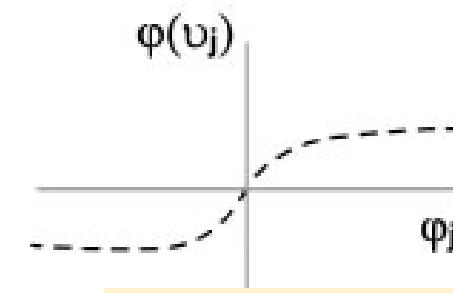


(b) Linear

MultiLayer Perceptron
(não lineares)

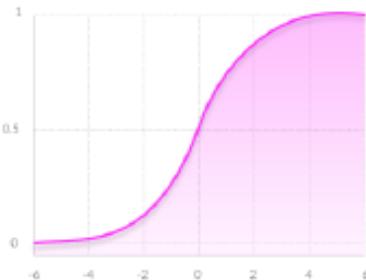


(c) Logistica



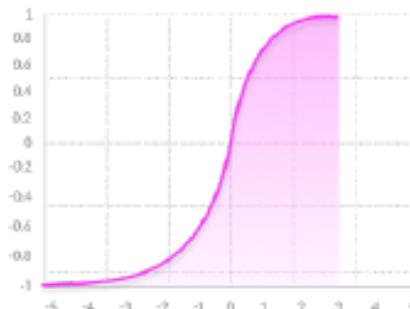
(d) Tangente Hiperbolica

Algoritmo de Treinamento para MLP



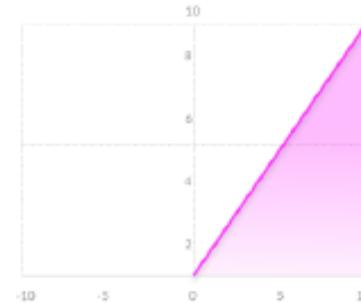
- Logística: valores entre $[0;1]$: $Q(v_k) = \frac{1}{1+\exp(-v_k)}$
 - Gradiente suave, evitando “saltos” nos valores de saída.
 - Predições claras: Para X acima de 2 ou abaixo de -2, tende a trazer o valor Y (a predição) para a borda da curva, muito próximo de 1 ou 0.
 - Problema: Gradiente de fuga - para valores muito altos ou muito baixos de X , quase não há alteração na predição. Rede não aprende ou o aprendizado é lento. Saídas não centradas em zero. Computacionalmente caro

Algoritmo de Treinamento para MLP



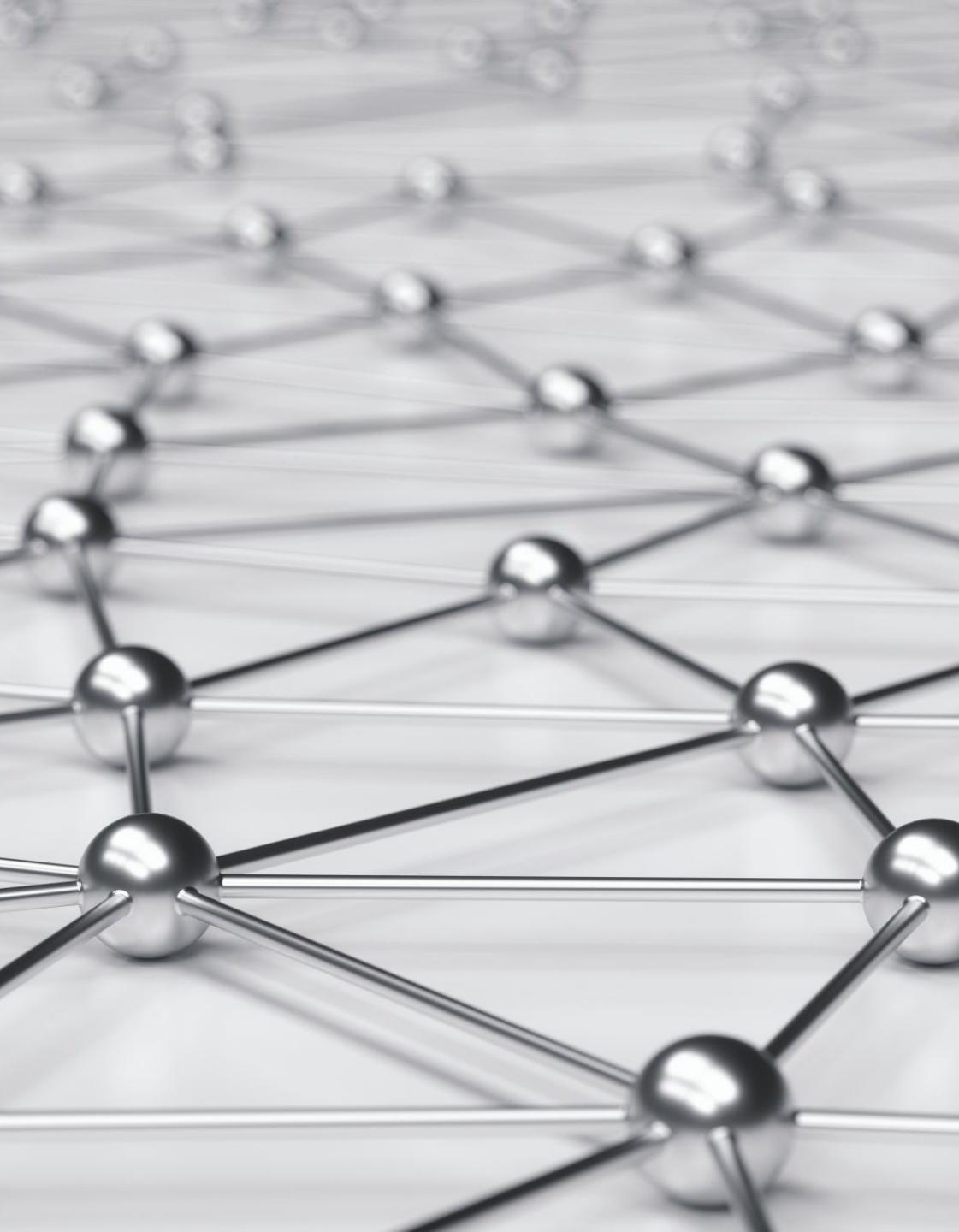
- Tangente hiperbólica: valores entre $[-1;1]$: $Q(v_k) = \tanh(v_k)$
 - Zero centrado - facilitando a modelagem de entradas com valores fortemente negativos, neutros e fortemente positivos.
 - Problema: Gradiente de fuga também.

Algoritmo de Treinamento para MLP



RELU: função mais recente

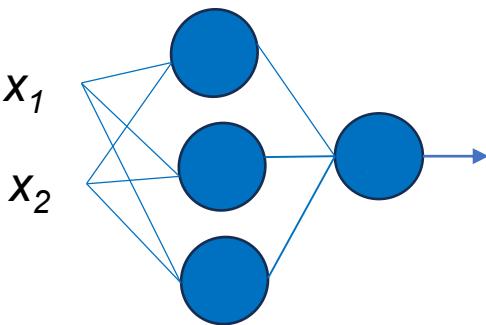
- RELU (RECTIFIED LINEAR UNIT): $ReLU(x) = \max(0, x)$
 - Computacionalmente eficiente - converge rapidamente
 - Não linear - embora pareça uma função linear
 - Problema (Dying ReLU): quando as entradas se aproximam de zero, ou são negativas, o gradiente da função se torna zero, a rede não aprende.



Simulando uma rede MLP

Simulando o treinamento do XOR

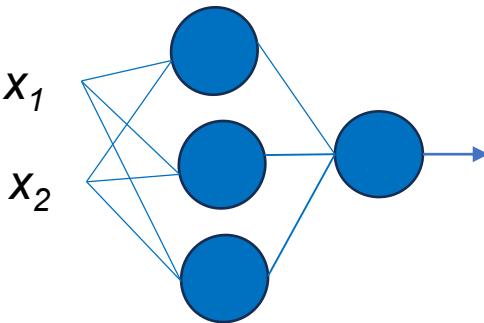
- Conjunto de Treino: (-1 : Falso; 1- Verdadeiro)



x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

- Função de Transferência
 - Camada oculta: tangente hiperbólica
 - Camada de saída: tangente hiperbólica
- Taxa de Aprendizagem: $\eta = 0.3$
- Constante de momentum: $\alpha = 0$
- Critérios de parada:
 - EMQ = 0.01 ou
 - Épocas: 1000

Simulando o treinamento do XOR

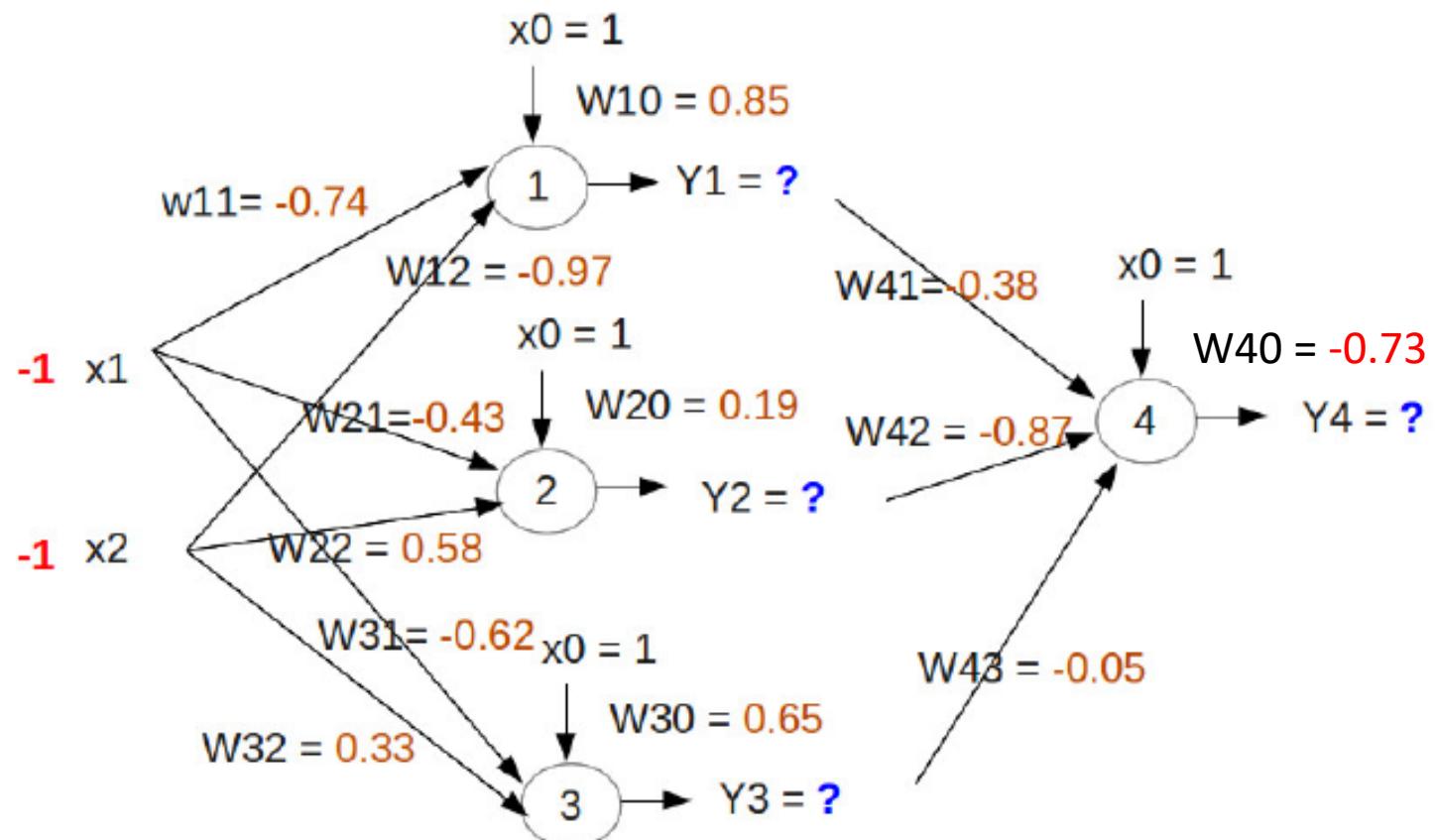


- Observações:
 - Topologia-Exemplo: $2 \times 3 \times 1$
 - Pesos iniciais gerados aleatoriamente
 - Constante de momento = 0 para simplificar o exemplo.

Simulando o treinamento do XOR

Propagação

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



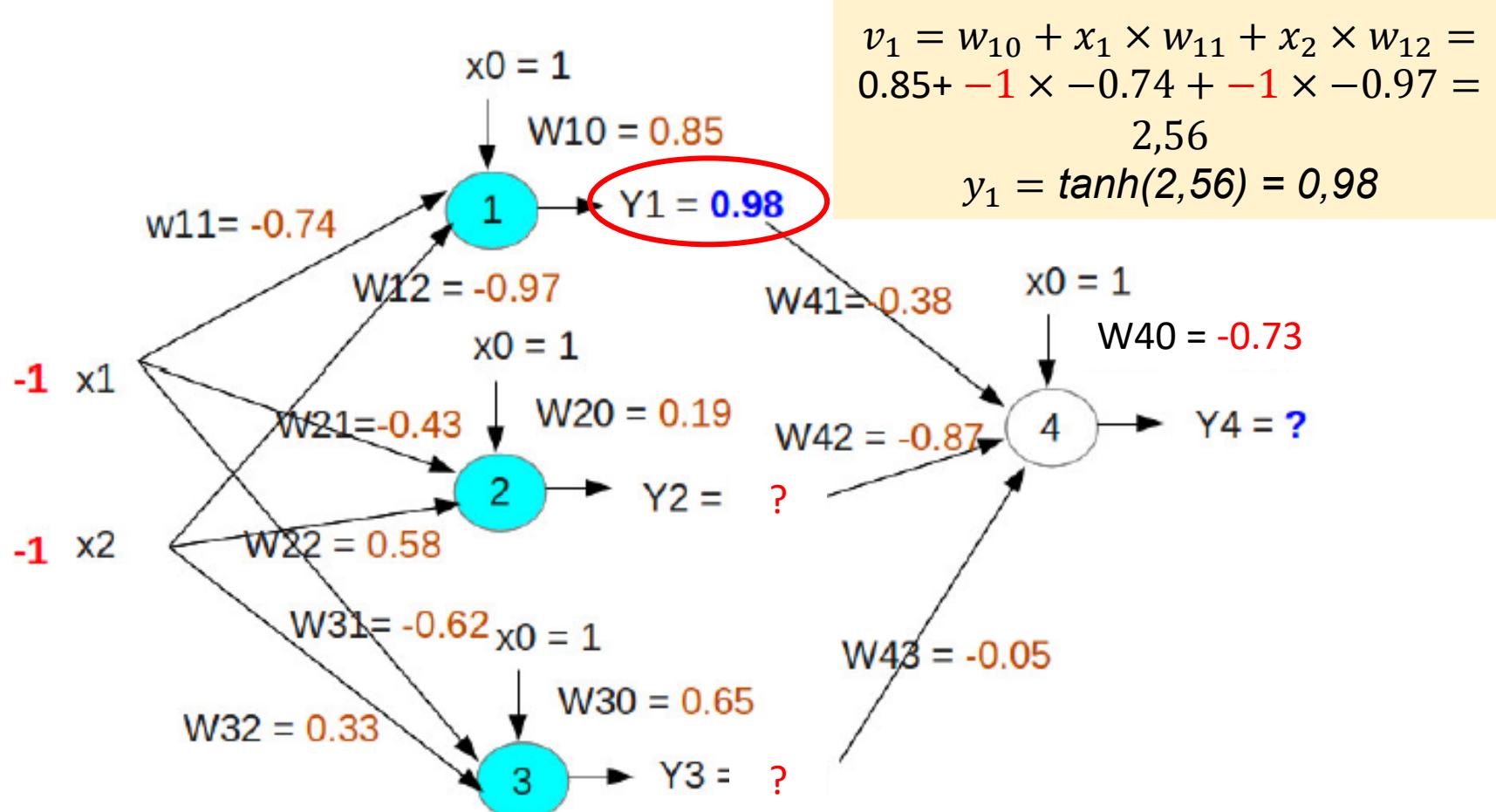
Simulando o treinamento do XOR

Propagação

$$v_j = w_{j0} + \sum_{i=1}^n (x_i \times w_{ji})$$

$y_j = f(v_j)$, onde
 $f(v_j) = \tanh(v_j)$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



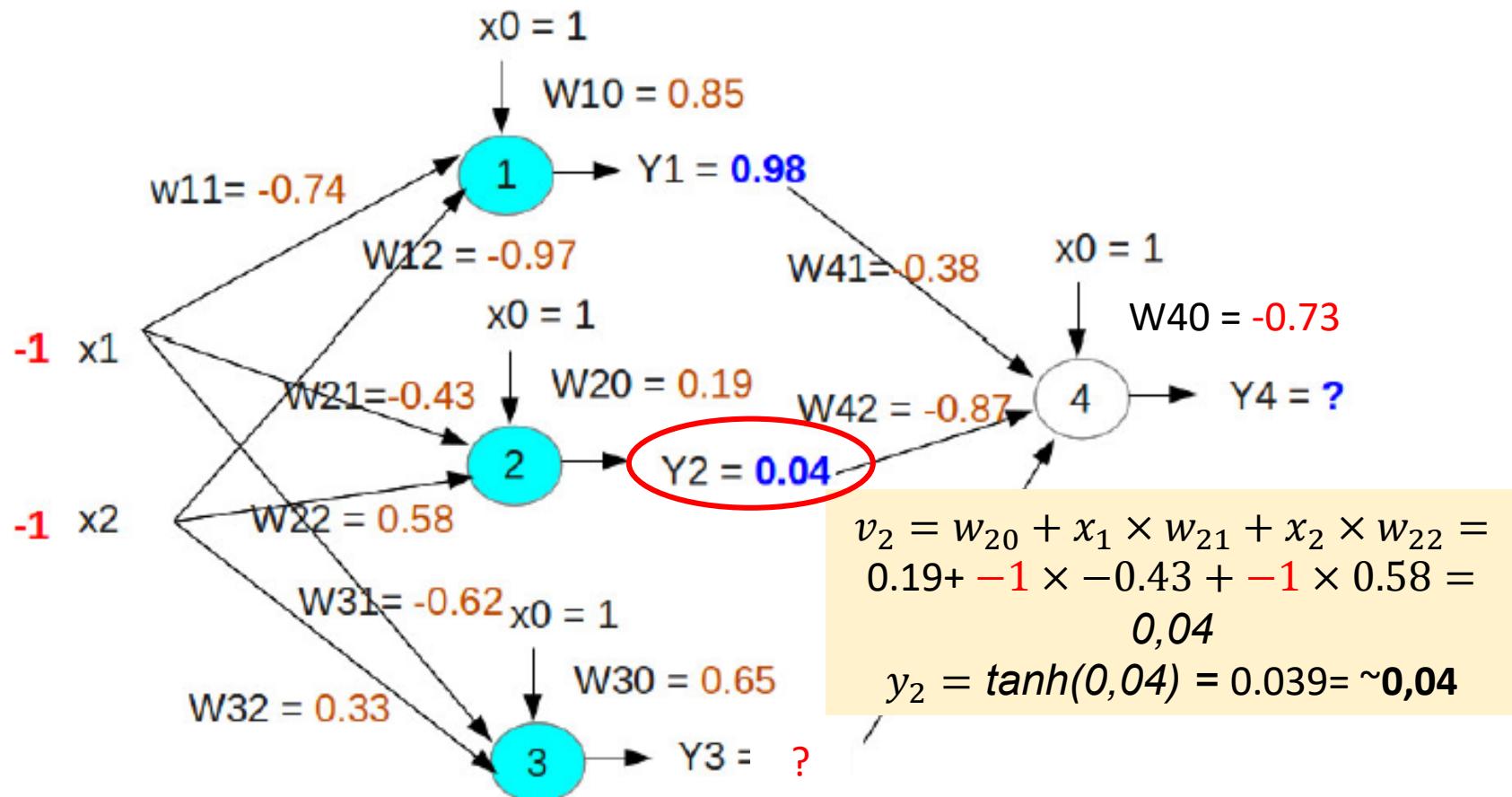
Simulando o treinamento do XOR

Propagação

$$v_j = w_{j0} + \sum_{i=1}^n (x_i \times w_{ji})$$

$y_j = f(v_j)$, onde
 $f(v_j) = \tanh(v_j)$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



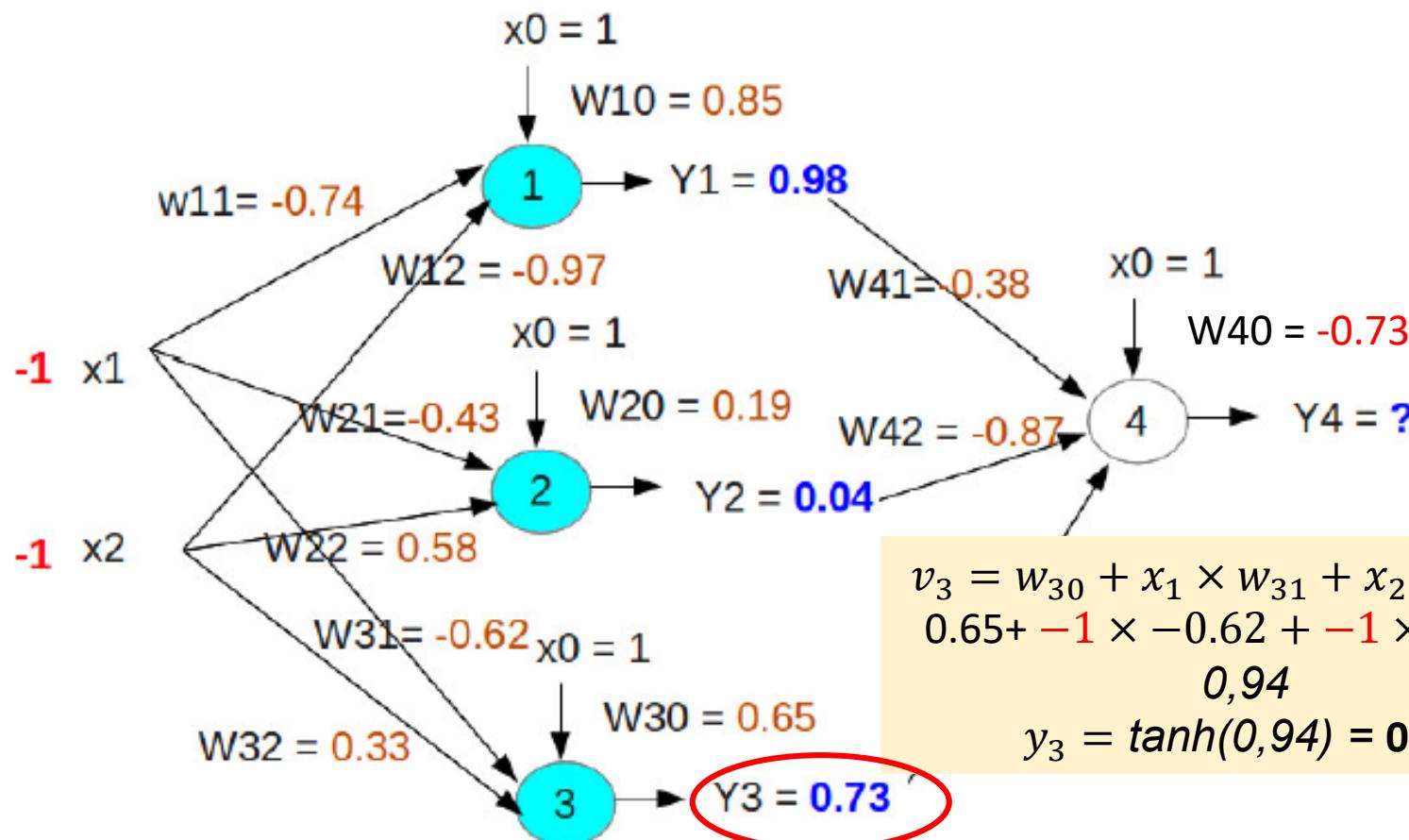
Simulando o treinamento do XOR

Propagação

$$v_j = w_{j0} + \sum_{i=1}^n (x_i \times w_{ji})$$

$y_j = f(v_j)$, onde
 $f(v_j) = \tanh(v_j)$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



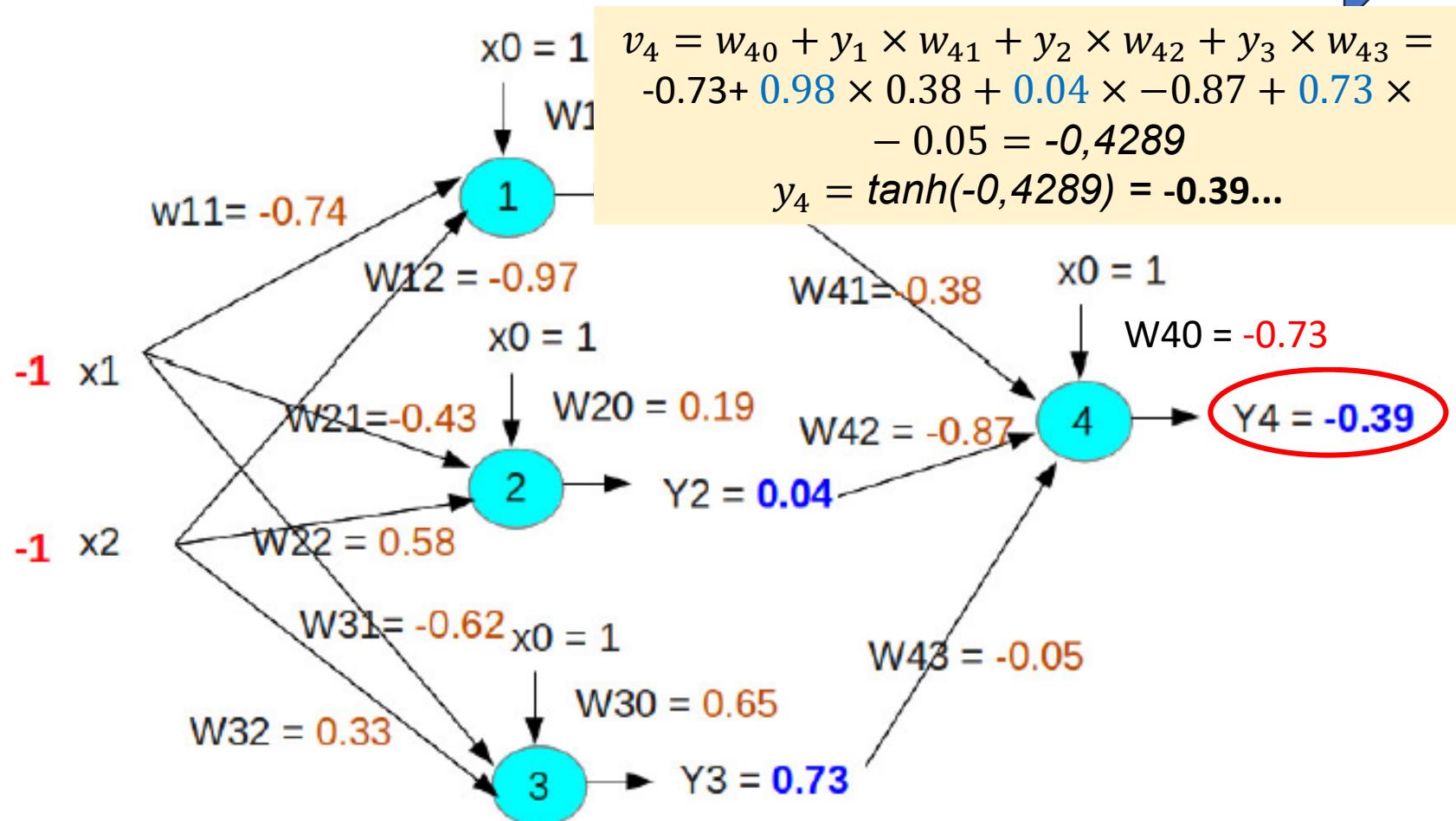
Simulando o treinamento do XOR

Propagação

$$v_j = w_{j0} + \sum_{i=1}^n (x_i \times w_{ji})$$

$y_j = f(v_j)$, onde
 $f(v_j) = \tanh(v_j)$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



Simulando o treinamento do XOR

Retropropagação

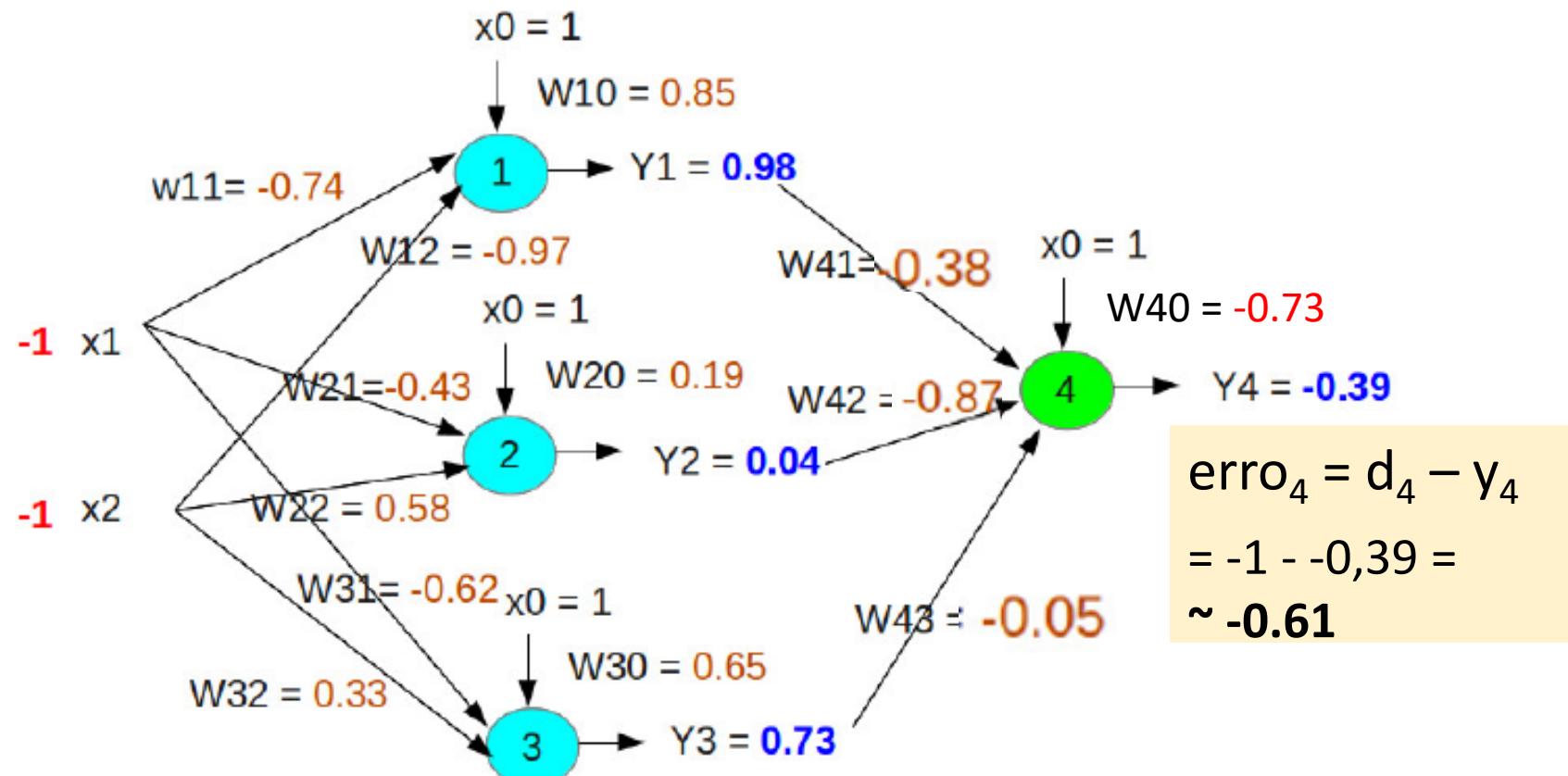
x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Neurônio 4 (Camada de Saída)

Erro: -0.6015309621707821

Erro Instantâneo:
0.1809197492250534

Gradiente =
-0.5060213352461277



Simulando o treinamento do XOR

Retropropagação

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Neurônio 4 (Camada de Saída)

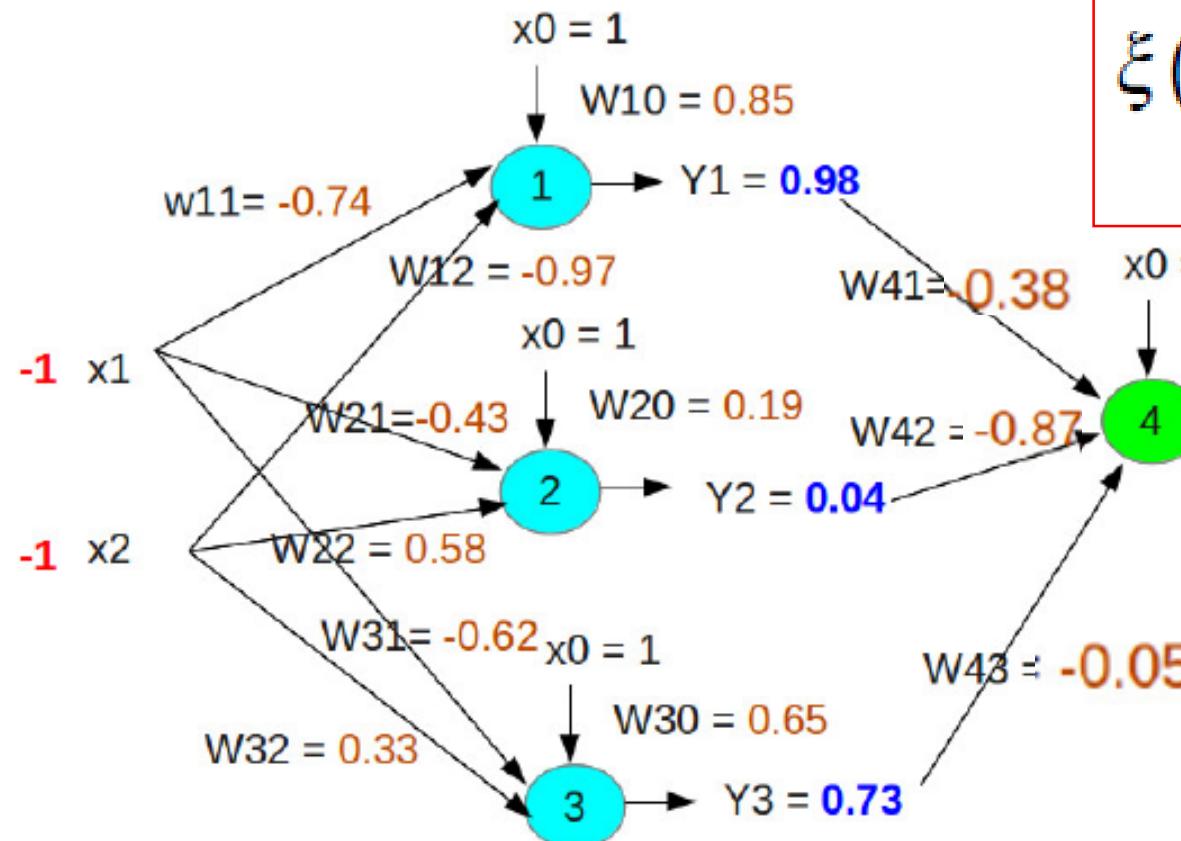
Erro: -0.6015309621707821

Erro Instantâneo:

0.1809197492250534

Gradiente =

-0.5060213352461277



$$\xi(n) = \frac{1}{2} \sum_{k=1}^s erro_k^2$$

$$\begin{aligned}
 erro_4 &= d_4 - y_4 \\
 &= -1 - -0,39 = \sim 0,61 \\
 \xi((-1,-1)) &= -0,61 \times -0,61 / 2 \\
 &= \sim 0,18
 \end{aligned}$$

Simulando o treinamento do XOR

Retropropagação

$$\delta_k(n) = Q'(v_k(n)) \times erro_k(n)$$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

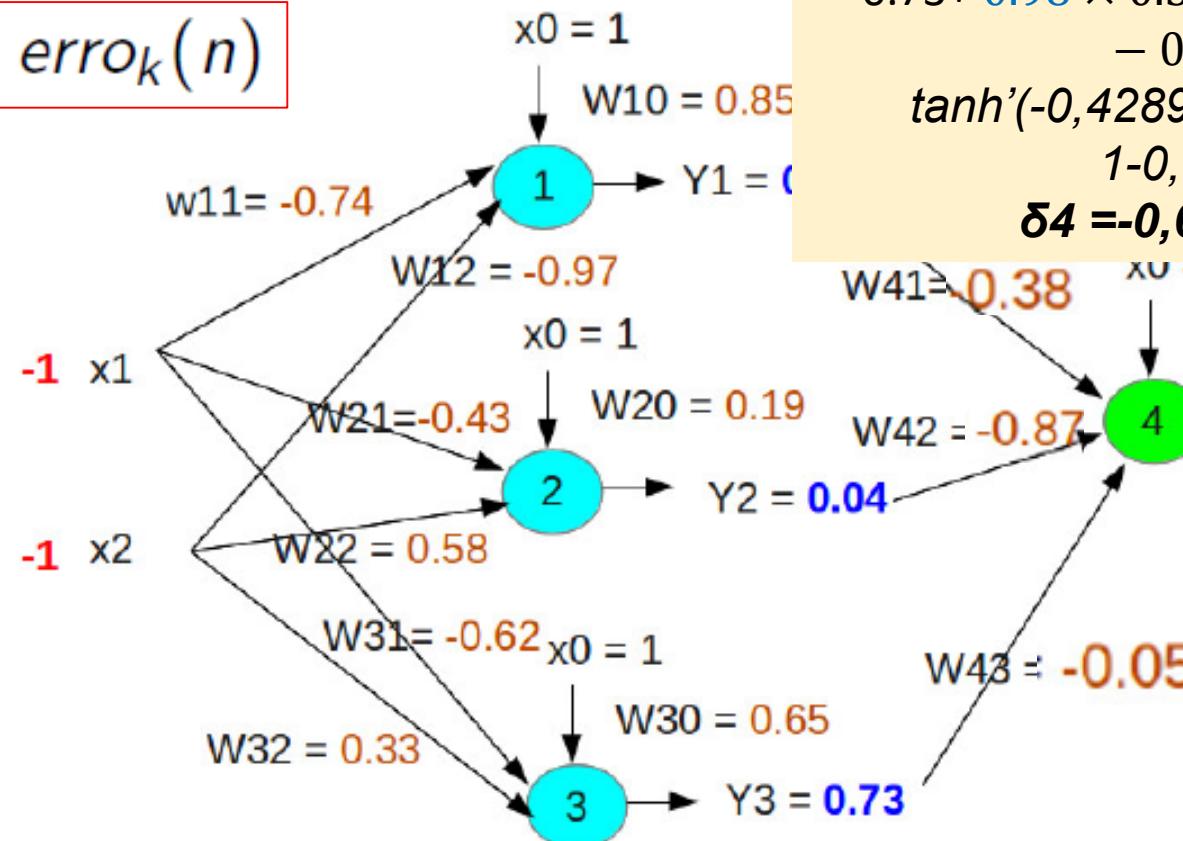
Neurônio 4 (Camada de Saída)

Erro: -0.6015309621707821

Erro Instantâneo:

0.1809197492250534

Gradiente =
-0.5060213352461277



$$v_4 = w_{40} + y_1 \times w_{41} + y_2 \times w_{42} + y_3 \times w_{43} = \\ -0.73 + 0.98 \times 0.38 + 0.04 \times -0.87 + 0.73 \times \\ -0.05 = -0.4289$$

$$\tanh'(-0.4289) = 1 - \tanh(-0.4289)^2 = \\ 1 - 0.1521 = 0.8479$$

$$\delta_4 = -0.61 * 0.8479 = -0.51$$

$$erro_4 = d_4 - y_4 \\ = -1 - -0.39 = \sim -0.61$$

Simulando o treinamento do XOR

Retropropagação

$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Neurônio 4 (Camada de Saída)

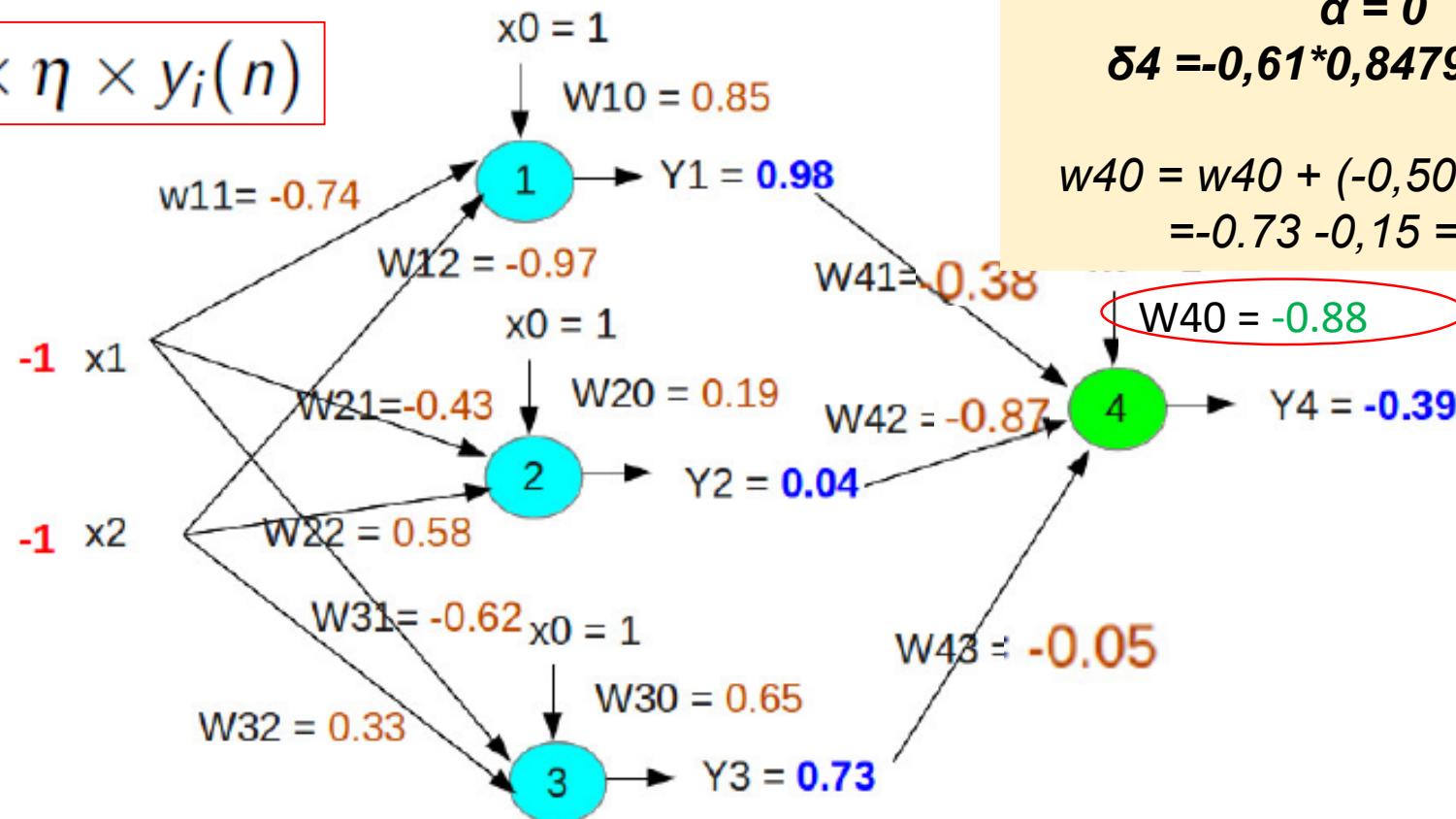
Erro: -0.6015309621707821

Erro Instantâneo:

0.1809197492250534

Gradiente =

-0.5060213352461277



$$\eta = 0.3$$

$$\alpha = 0$$

$$\delta_4 = -0.61 * 0.8479 = -0.51...$$

$$w_{40} = w_{40} + (-0.50 \times 0.3 \times 1) =$$

$$= -0.73 - 0.15 = \boxed{-0.88}$$

$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$

Simulando o treinamento do XOR



$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Neurônio 4 (Camada de Saída)

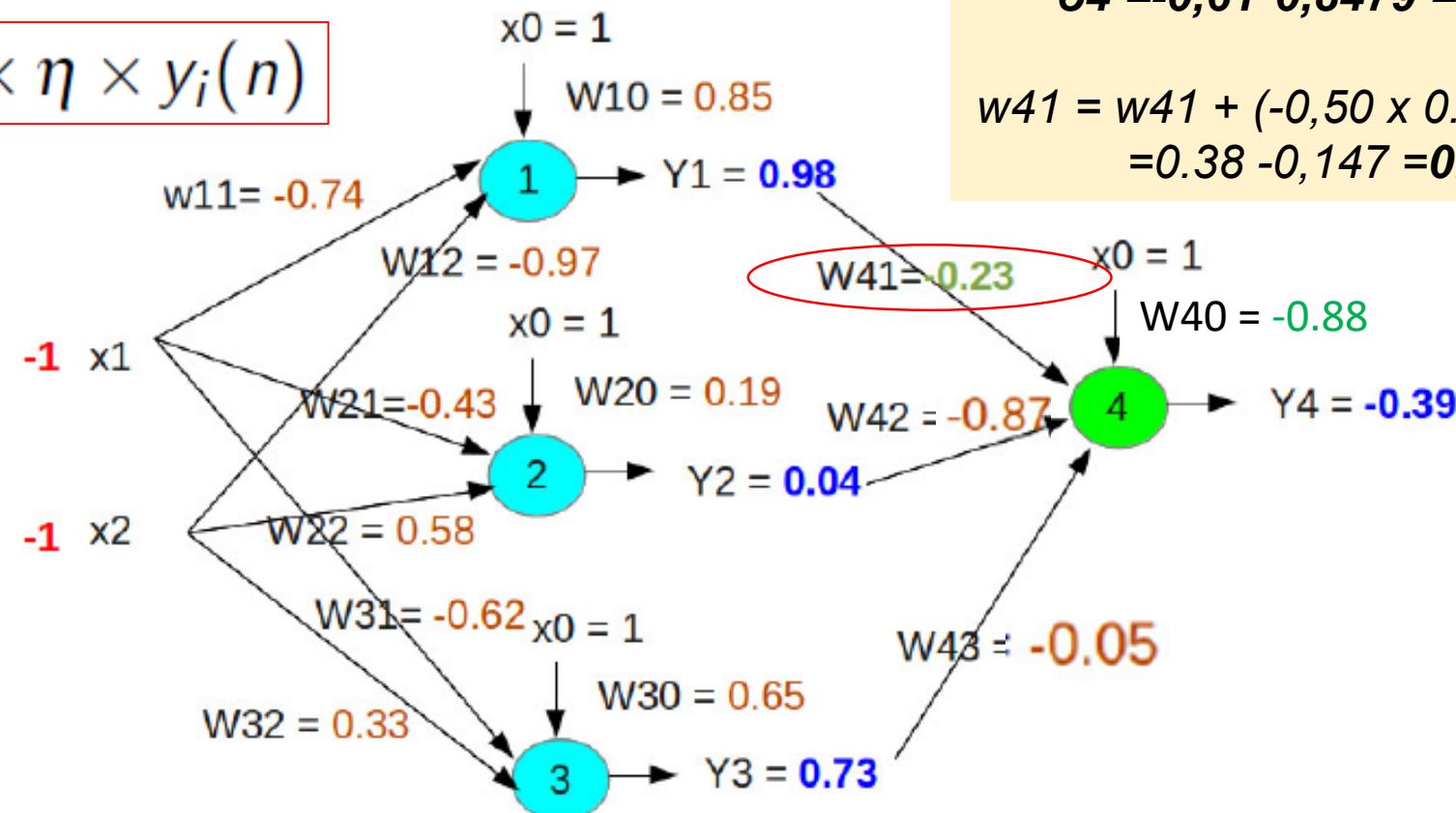
Erro: -0.6015309621707821

Erro Instantâneo:

0.1809197492250534

Gradiente =

-0.5060213352461277



$$\eta = 0.3$$

$$\alpha = 0$$

$$\delta_4 = -0.61 * 0.8479 = -0.5...$$

$$w_{41} = w_{41} + (-0.50 \times 0.3 \times 0.98) = \\ = 0.38 - 0.147 = 0.23$$

$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$

Simulando o treinamento do XOR



$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Neurônio 4 (Camada de Saída)

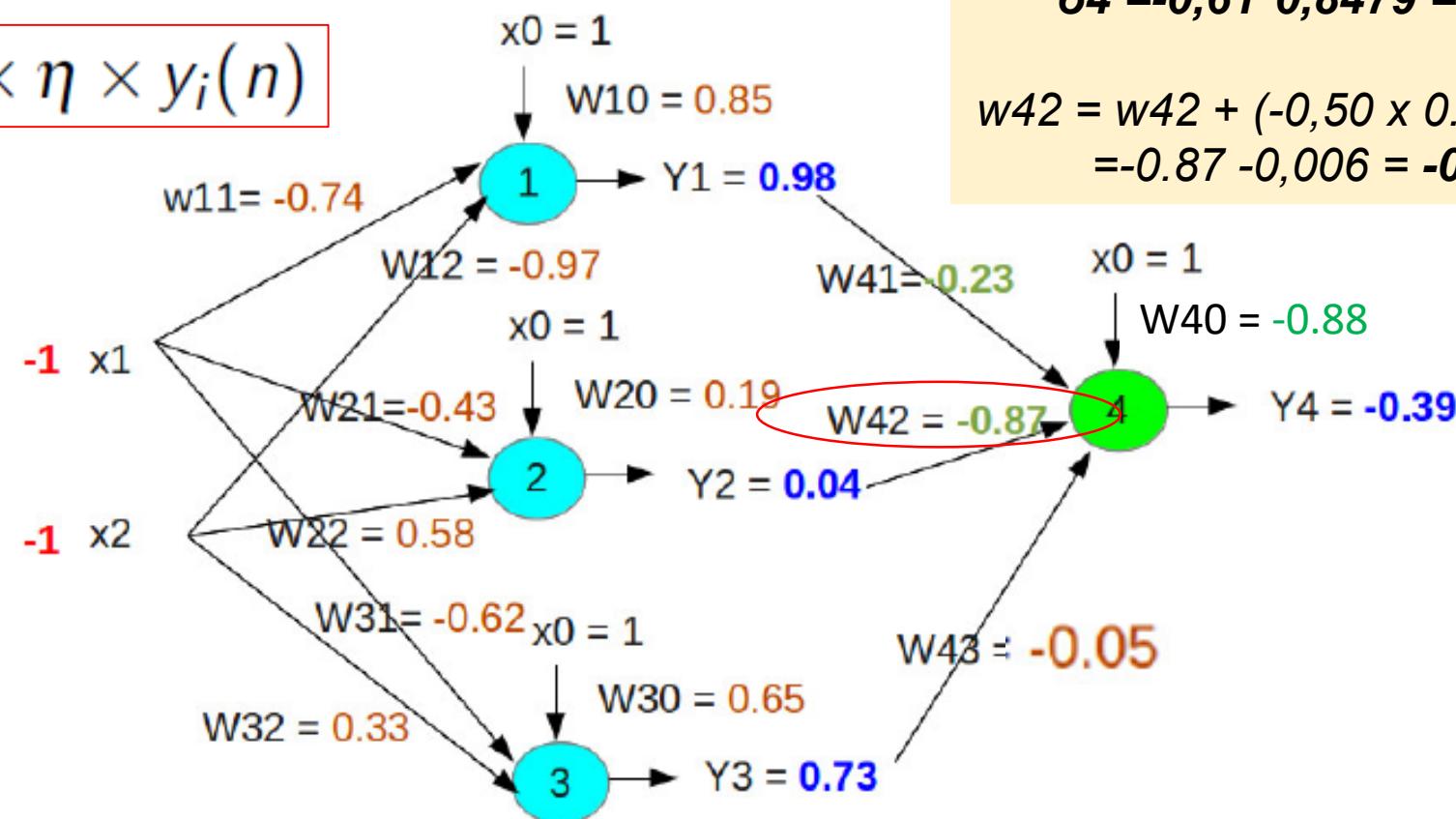
Erro: -0.6015309621707821

Erro Instantâneo:

0.1809197492250534

Gradiente =

-0.5060213352461277



$$\eta = 0.3$$

$$\alpha = 0$$

$$\delta_4 = -0.61 * 0.8479 = -0.5...$$

$$w_{42} = w_{42} + (-0.50 \times 0.3 \times 0.04) = \\ = -0.87 - 0.006 = \mathbf{-0.876}$$

$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$

Simulando o treinamento do XOR



$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Neurônio 4 (Camada de Saída)

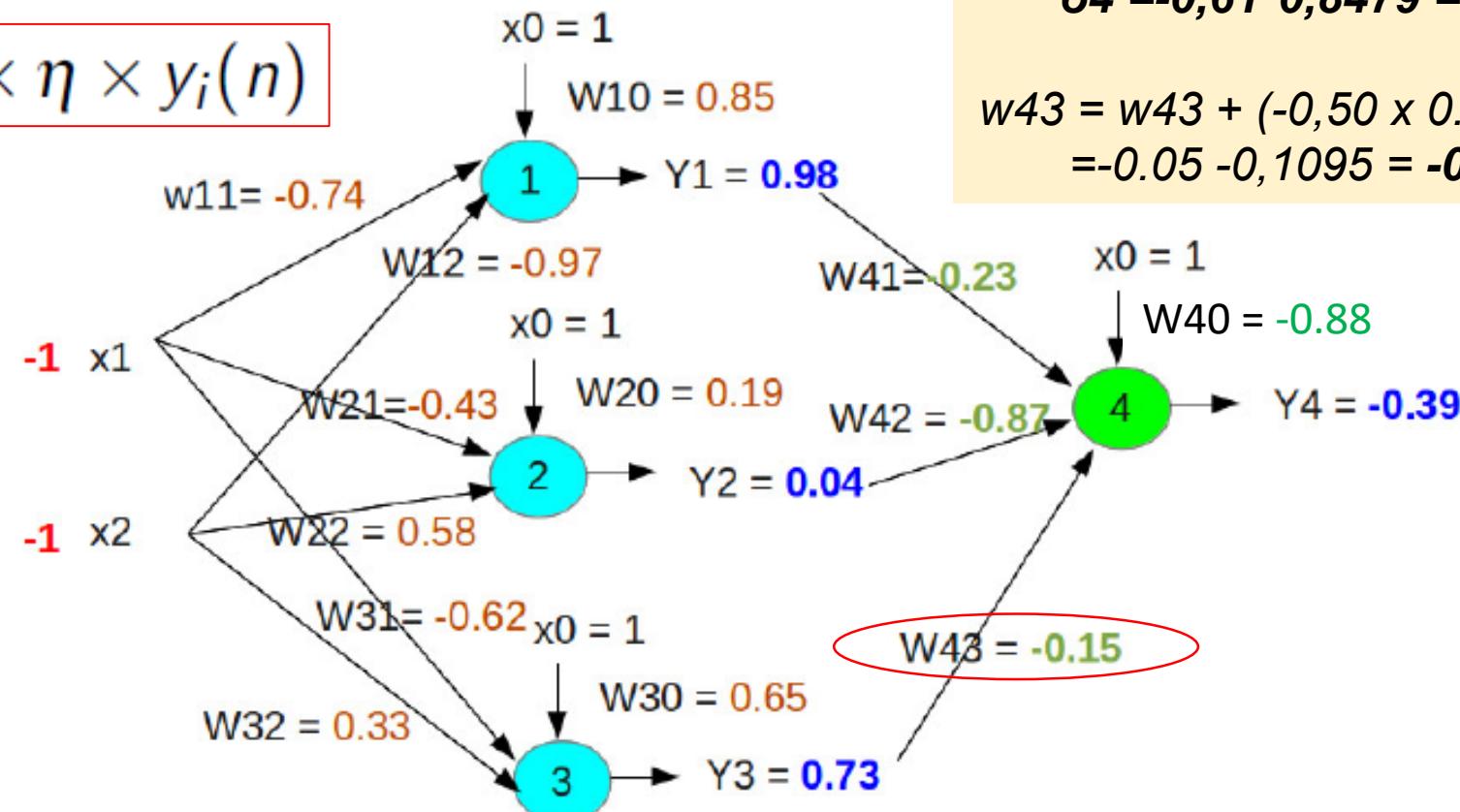
Erro: -0.6015309621707821

Erro Instantâneo:

0.1809197492250534

Gradiente =

-0.5060213352461277



$$\eta = 0.3$$

$$\alpha = 0$$

$$\delta_4 = -0.61 * 0.8479 = -0.5...$$

$$w_{43} = w_{43} + (-0.50 \times 0.3 \times 0.73) = \\ = -0.05 - 0.1095 = \textcolor{red}{-0.1595}$$

$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$

Simulando o treinamento do XOR



x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

- Camada Oculta

- Gradiente Neurônio 1: -0.002781842410268394

- Gradiente Neurônio 2: 0.4444999763647362

- Gradiente Neurônio 3: 0.03803417896970686

$$v_1 = w_{10} + x_1 \times w_{11} + x_2 \times w_{12} = 0.85 + -1 \times -0.74 + -1 \times -0.97 = 2,56$$

$$\tanh'(2,56) = 1 - \tanh(2,56)^2 = 1 - 0,97 = 0,03$$

$$\delta_4 = -0,5...$$

$$\delta_1 = 0,03 * [-0,5 * 0,23] = \sim -0,003$$

$$\delta_k(n) = Q'(v_k(n)) * \sum_{i=1}^t (\delta_j(n) * w_{jk}(n+1))$$

Simulando o treinamento do XOR



$$w_{10} = w_{10} + (-0,0027 \times 0,3 \times 1) = \\ 0,85 + -0,00081 = \mathbf{0,84}$$

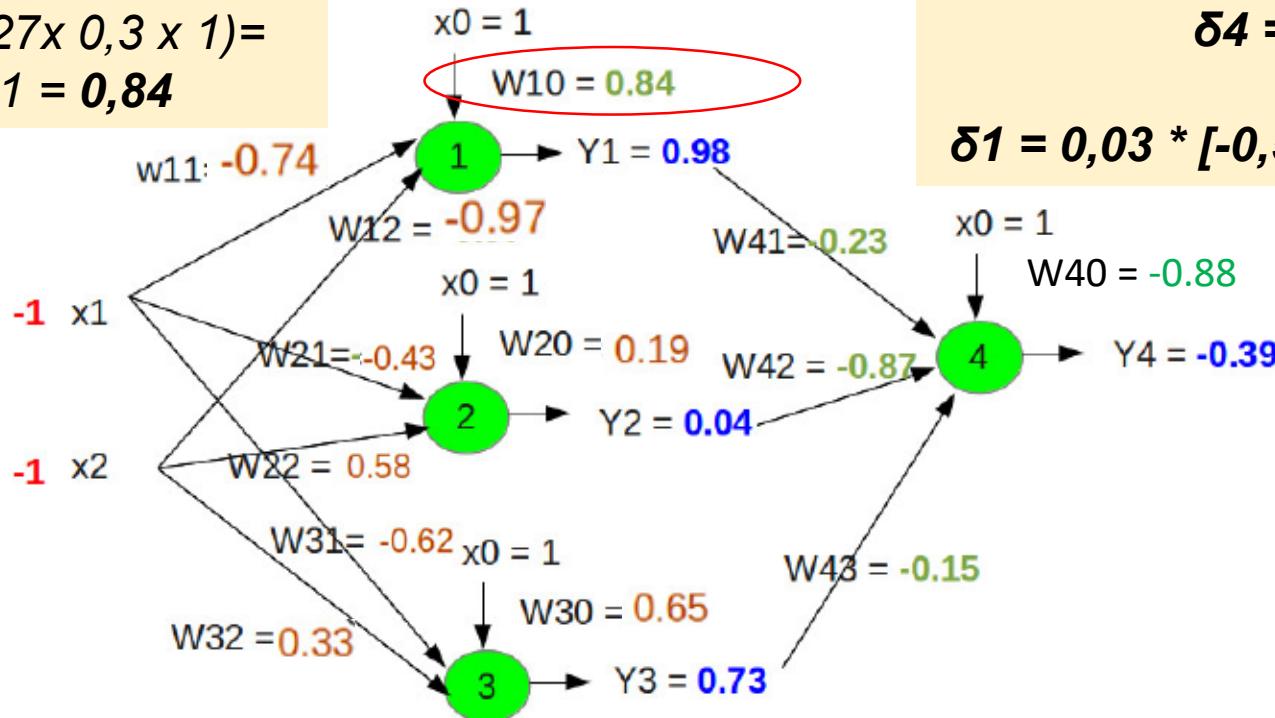
x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

● Camada Oculta

● Gradiente Neurônio 1: -0,002781842410268394

● Gradiente Neurônio 2: 0,4444999763647362

● Gradiente Neurônio 3: 0,03803417896970686



$$v_1 = w_{10} + x_1 \times w_{11} + x_2 \times w_{12} = \\ 0,85 + -1 \times -0,74 + -1 \times -0,97 = \\ 2,56$$

$$\tanh'(2,56) = 1 - \tanh(2,56)^2 = 1 - \\ 0,97 = \mathbf{0,03}$$

$$\delta_4 = -0,5\dots$$

$$\delta_1 = 0,03 * [-0,5 * 0,23] = \sim -0,003$$

$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$

Simulando o treinamento do XOR



$$w_{11} = w_{11} + (-0,0027 \times 0,3 \times -1) = \\ -0,74 + 0,00081 = \mathbf{-0,739} = \sim -0,74$$

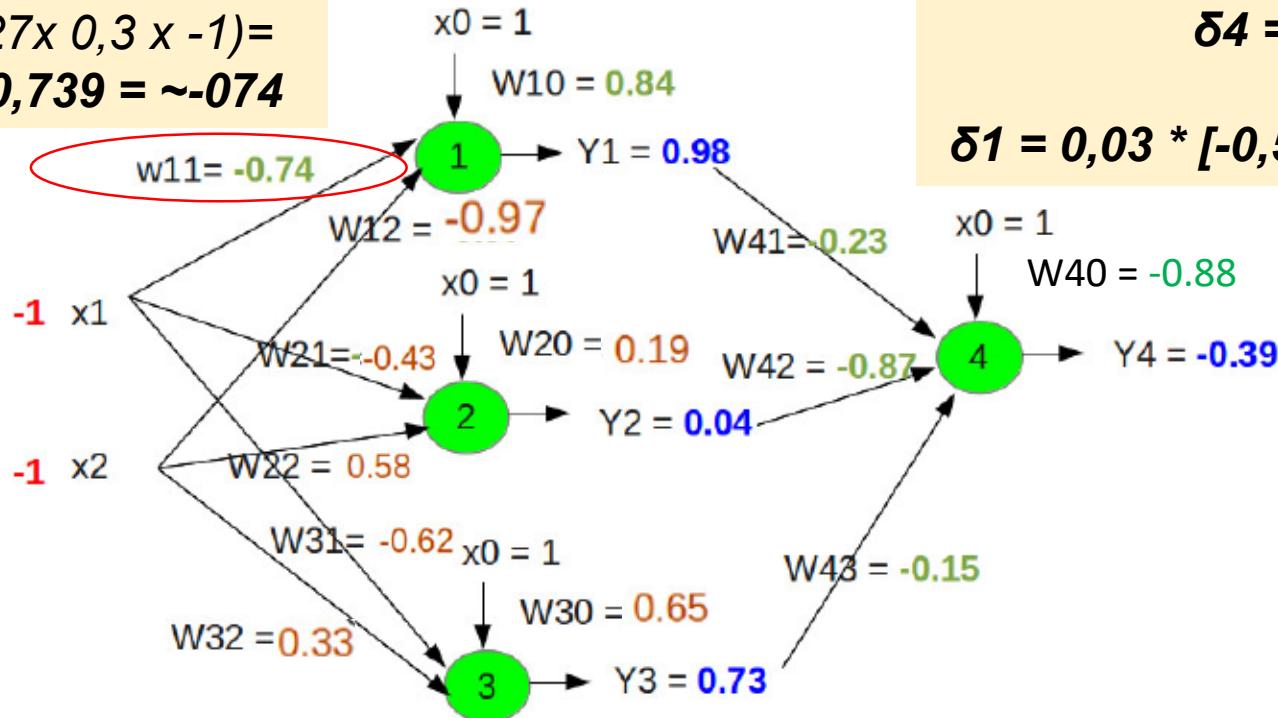
x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

● Camada Oculta

● Gradiente Neurônio 1: -0,002781842410268394

● Gradiente Neurônio 2: 0,4444999763647362

● Gradiente Neurônio 3: 0,03803417896970686



$$v_1 = w_{10} + x_1 \times w_{11} + x_2 \times w_{12} = \\ 0.85 + \mathbf{-1} \times -0.74 + \mathbf{-1} \times -0.97 = \\ 2,56$$

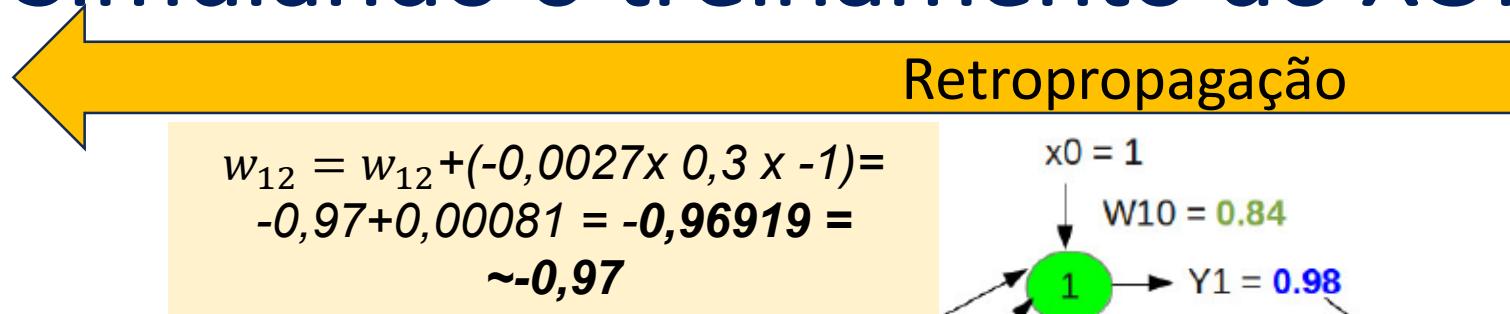
$$\tanh'(2,56) = 1 - \tanh(2,56)^2 = 1 - \\ 0,97 = \mathbf{0,03} \\ \delta_4 = \mathbf{-0,5...}$$

$$\delta_1 = 0,03 * [-0,5 * 0,23] = \sim -0,003$$

$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$

Simulando o treinamento do XOR



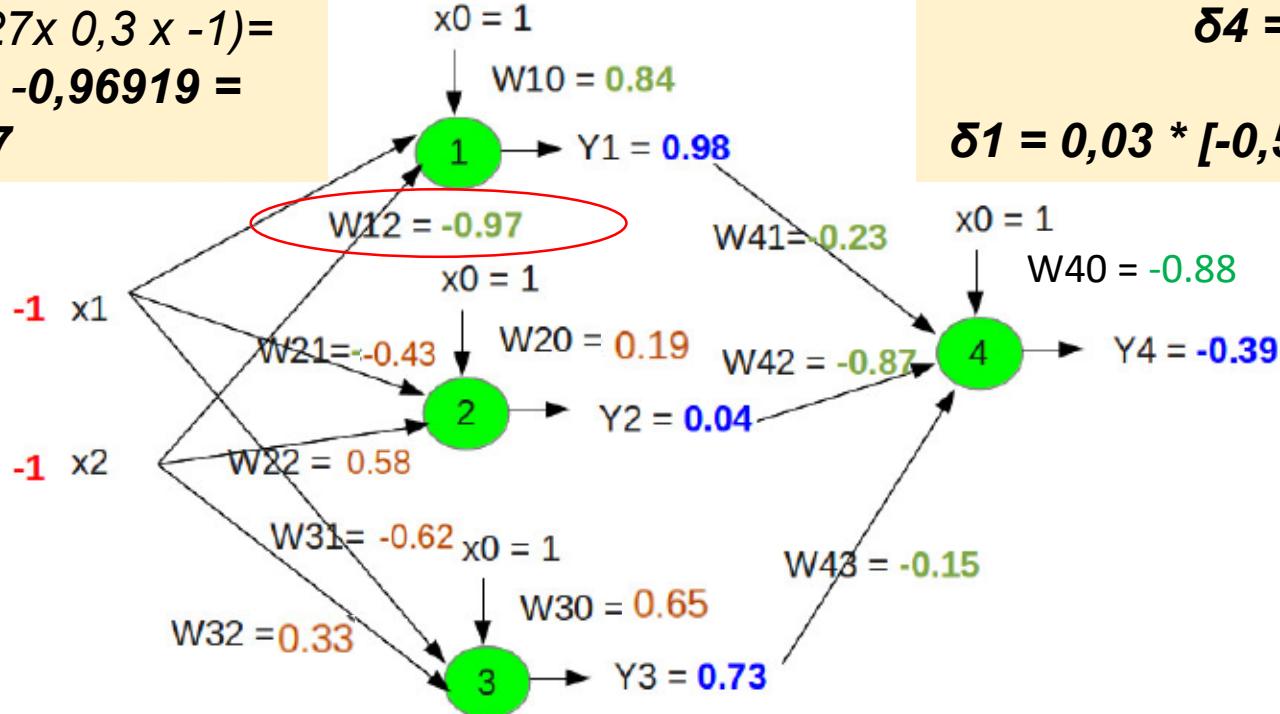
x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

● Camada Oculta

- Gradiente Neurônio 1: -0.002781842410268394

- Gradiente Neurônio 2: 0.4444999763647362

- Gradiente Neurônio 3: 0.03803417896970686



$$v_1 = w_{10} + x_1 \times w_{11} + x_2 \times w_{12} = \\ 0.85 + \mathbf{-1} \times -0.74 + \mathbf{-1} \times -0.97 = \\ 2,56$$

$$\tanh'(2,56) = 1 - \tanh(2,56)^2 = 1 - \\ 0,97 = \mathbf{0,03}$$

$$\delta_4 = -0,5\dots$$

$$\delta_1 = 0,03 * [-0,5 * 0,23] = \sim -0,003$$

$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

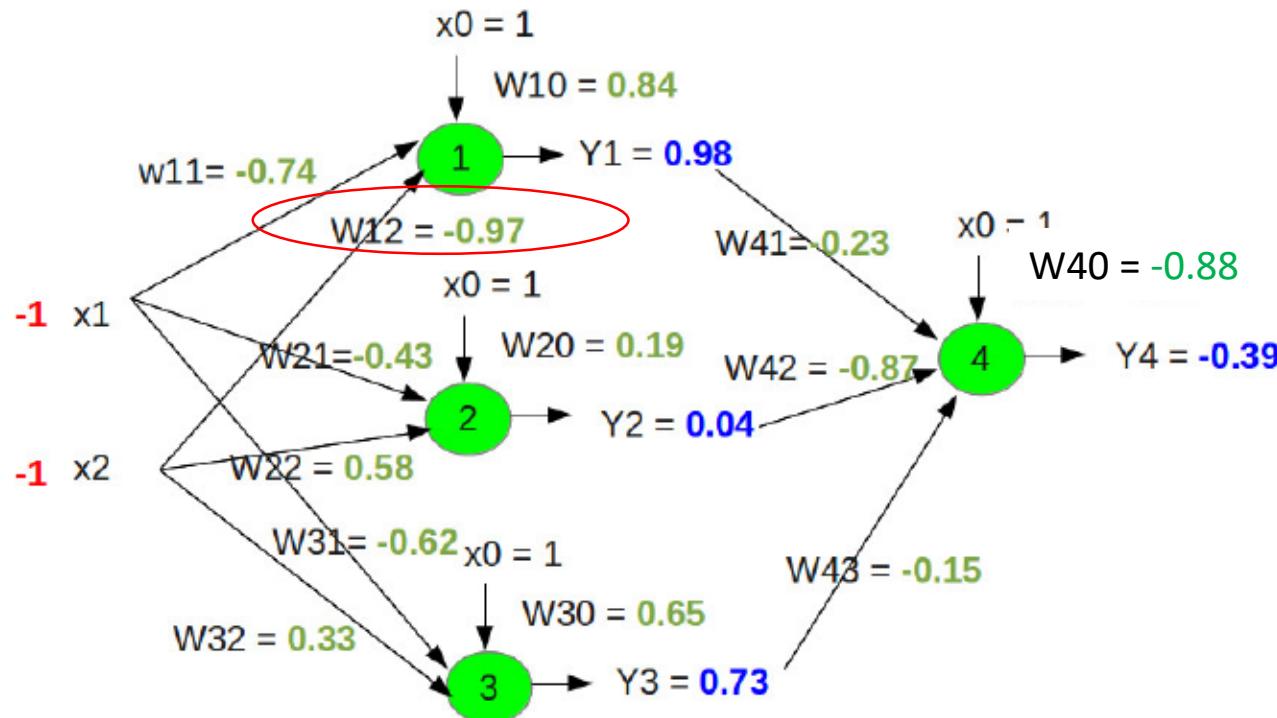
$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$

Simulando o treinamento do XOR

Retropropagação

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

- Camada Oculta
- Gradiente Neurônio 1: -0.002781842410268394
- Gradiente Neurônio 2: 0.4444999763647362
- Gradiente Neurônio 3: 0.03803417896970686



$$\Delta w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$$

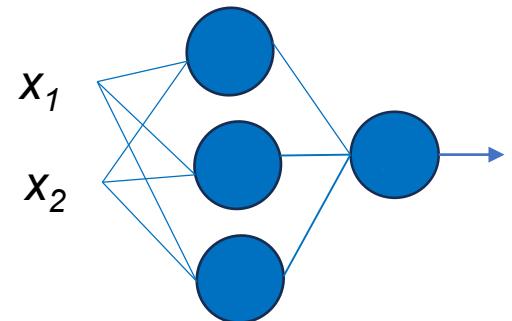
$$w_{ki}(n+1) = w_{ki}(n) + \Delta w_{ki}(n) + \alpha \times w_{ki}(n-1)$$

Simulando o treinamento do XOR

Função de Custo

- Época: 0 - Erro médio quadrado: 0.6382619197564264
- Época: 1 - Erro médio quadrado: 0.6217486395123959
- Época: 2 - Erro médio quadrado: 0.5710208319681408
- Época: 3 - Erro médio quadrado: 0.5473148706283855
- Época: 4 - Erro médio quadrado: 0.5355908980006899
- Época: 5 - Erro médio quadrado: 0.5285727922709779
- ...
- Epoca: 446 - Erro médio quadrado: 0.009972902630117393

Simulando o treinamento do XOR



Pesos Finais:

- Camada Oculta
 - Neuronio: 1: $w[10]=2.545363016862031$,
 $w[11]=-1.8662421473793356$, $w[12]=-1.7588858258650666$
 - Neuronio: 2: $w[20]=0.1914996892738543$,
 $w[21]=-0.4332485003836084$, $w[22]=0.5833743630800029$
 - Neuronio: 3: $w[30]=0.655200823402003$
 $w[31]=-0.6209562373205424$ $w[32]=0.3376845283666482$
- Camada de Saída
 - Neuronio: 4: $w[40]=0.36071557007625865$,
 $w[41]=2.388002191368662$, $w[42]=4.357544216452374$,
 $w[43]=-5.4985276529155485$

Implementando uma MLP

`sklearn.neural_network.MLPClassifier`

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001,  
batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None,  
tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False,  
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000) \[source\]
```

- **hidden_layer_sizes** : vetor com a quantidade de neurônios da camada oculta - default=(100,)
- **activation**{‘identity’, ‘logistic’, ‘tanh’, ‘relu’}, default=’relu’ - função de ativação para camada oculta
 - ‘identity’, sem ativação, retorna $f(x) = x$
 - ‘logistic’, função sigmoide logística , retorna $f(x) = 1 / (1 + \exp(-x))$.
 - ‘tanh’, função tangente hiperbólica, retorna $f(x) = \tanh(x)$.
 - ‘relu’, função *rectified linear unit*, retorna $f(x) = \max(0, x)$

Implementando uma MLP

- **solver**{‘lbfgs’, ‘sgd’, ‘adam’}, default=’adam’
Usado para otimizacao dos pesos.
 - ‘sgd’ : refere-se ao gradiente descente estocástico..
 - ‘adam’ : refere-se ao otimizador baseado em gradiente.
 - ‘lbfgs’ : otimizador quasi-Newton methods.
- **learning_rate_init** float, default=0.001
Taxa de aprendizao, usada quando=’sgd’ ou ‘adam’.
- **max_iter** int, default=200 - Numero máximo de iteracoes.
- **verbose** bool, default=False – Exibe mensagens sobre a rede.
- **Momentum** float, default=0.9 - Constante de momento para atualização do gradiente .
Deve estar entre [0;1]. Somente usar quando solver solver=’sgd’.

Dinâmica

- **Atividade 1:** Implemente em Python o algoritmo de treinamento da rede MultiLayer Perceptron para o XOR.

x1	x2	d
0	0	0
0	1	1
1	0	1
1	1	0

