



Resumo IA

ver. 2

Sophia Carrazza

Regras de Associação

$$\text{Supóte} = \frac{\text{nº de transações que contêm } A \text{ e } B}{\text{nº de transações}}$$

Regras de associação relacionam atributos que "co-ocorrem" em conjuntos de itens (itenset)

• Supóte* = nº de transações que contêm todos os itens da transação dividida pelo nº de transações.

↳ ocorrência registrada de um conjunto de itens

• acurácia/confiança de uma regra ($A \rightarrow B$) = proporção de vezes que, em uma transação contendo A, também tem B.

$$\text{Confiança } (A \rightarrow B) = \frac{P(A \cup B)}{P(A)} = \frac{\text{supóte}(A \cup B)}{\text{supóte}(A)}$$

ou $\frac{\text{supóte}(A \cup B \cup C)}{\text{supóte}(A \cup B)}$

ou $\frac{\text{supóte}(A \cup B \cup C)}{\text{supóte}(A)}$
 $A \cup B \rightarrow C$
 $A \rightarrow B \cup C$

• lift (coeficiente de interesse) de uma regra ($A \rightarrow B$) = o quanto mais frequente torna-se B quando A ocorre.

$$\text{Lift} = \frac{\text{confiança } (A \rightarrow B)}{\text{supóte}(B)}$$

> **Algoritmo Apriori:** Usado para criar as regras de associação. Tem 2 fases:
 1- geramos os itens com o supóte mínimo especificado
 2- para cada itemset, geramos as regras com a confiança mínima especificada

* O lift é usado para calcular as melhores regras a partir das geradas pelo Apriori.

Valor de lift igual a 1 para a regra ($A \rightarrow B$) indica que A e B são independentes

Um valor de lift superior a 1 indica que A e B aparecem mais frequentemente juntos do que esperado. Isso significa que a ocorrência de A tem um efeito positivo sobre a ocorrência de B.

Um valor de lift menor do que 1 indica que A e B aparecem com menos frequência do que o esperado em conjunto, indicando que a ocorrência de A tem um efeito negativo sobre a ocorrência de B.

Exercício:

Questão 01

Considere que em um determinado supermercado foram efetuadas as seguintes transações:

Nº	Leite	Café	Cerveja	Pão	Manteiga	Arroz	Feijão
1	Não	Sim	Não	Sim	Sim	Não	Não
2	Sim	Não	Sim	Sim	Sim	Não	Não
3	Não	Sim	Não	Sim	Sim	Não	Não
4	Sim	Sim	Não	Sim	Sim	Não	Não
5	Não	Não	Sim	Não	Não	Não	Não
6	Não	Não	Não	Não	Sim	Não	Não
7	Não	Não	Não	Sim	Não	Não	Não
8	Não	Não	Não	Não	Não	Não	Sim
9	Não	Não	Não	Não	Não	Sim	Sim
10	Não	Não	Não	Não	Não	Sim	Não

Utilizando-se o algoritmo Apriori, um suporte mínimo aceitável de 0,3 e confiança de 0,8, o número de itensSets 1, 2, 3 e de regras a partir desta base de dados são:

SUporte (0,3)

Itenset de 1 item:

$$\text{leite: } 2/10 = 0,2$$

$$\text{café: } 3/10 \approx 0,3 \}$$

$$\text{cerveja: } 2/10 = 0,2$$

$$\text{pão: } 5/10 = 0,5 \} \text{ artes panam}$$

$$\text{manteiga: } 5/10 = 0,5$$

$$\text{arroz: } 2/10 = 0,2$$

$$\text{feijão: } 2/10 = 0,2$$

suporte

Itenset de 2 itens:

$$\begin{aligned} \text{café e pão: } & 3/10 = 0,3 \\ \text{pão e manteiga: } & 4/10 = 0,4 \\ \text{café e manteiga: } & 3/10 = 0,3 \end{aligned} \} \text{ todos panam}$$

Itenset de 3 itens:

$$\text{café, manteiga e pão: } 3/10 = 0,3 \} \text{ para}$$

CONFIANÇA (0,8)

Itenset 2:

$$\text{café} \rightarrow \text{manteiga} = \frac{0,3}{0,3} = 1 \checkmark$$

$$\text{manteiga} \rightarrow \text{café} = \frac{0,3}{0,5} = 0,6$$

$$\text{pão} \rightarrow \text{café} = \frac{0,3}{0,5} = 0,6$$

$$\text{café} \rightarrow \text{pão} = \frac{0,3}{0,3} = 1 \checkmark$$

$$\text{pão} \rightarrow \text{manteiga} = \frac{0,4}{0,5} = 0,8 \checkmark$$

$$\text{manteiga} \rightarrow \text{pão} = \frac{0,4}{0,5} = 0,8 \checkmark$$

Itenset 3:

$$\text{café e manteiga} \rightarrow \text{pão} = \frac{0,3}{0,3} = 1$$

$$\text{café e pão} \rightarrow \text{manteiga} = \frac{0,3}{0,3} = 1$$

$$\text{pão e manteiga} \rightarrow \text{café} = \frac{0,3}{0,4} = 0,75$$

$$\text{café} \rightarrow \text{pão e manteiga} = \frac{0,3}{0,3} = 1$$

$$\text{pão} \rightarrow \text{café e manteiga} = \frac{0,3}{0,5} = 0,6$$

$$\text{manteiga} \rightarrow \text{pão e café} = \frac{0,3}{0,5} = 0,6$$

Ont de regras de associação gerados = 7 //

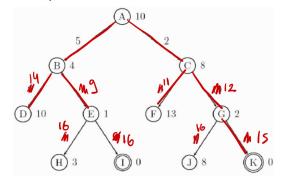
Buscas

Profundidade e Largura - algoritmos de grafos que não verificam heurística

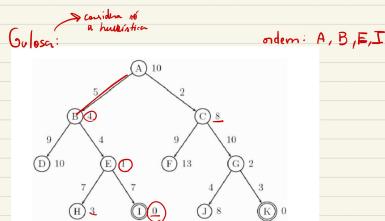
$$\begin{cases} \text{(mesma)} \\ \text{árvore} \end{cases} \quad \left\{ \begin{array}{l} \text{largura} = A, B, C, D, E, F, G, H, I \\ \text{profundidade} = A, B, P, E, H, I \end{array} \right.$$

Custo Uniforme - considera só o menor custo acumulado (também não verifica heurística).

Custo Uniforme: \rightarrow custo acumulado
A, C, B, E, F, G, D, K



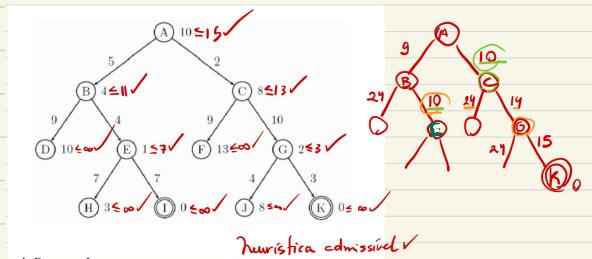
Gulosa - considera só a escolha da menor heurística. O objetivo é minimizar $h(n)$ a cada passo (mesmo que aheurística não seja admissível).



A* - considera a escolha da menor soma de $g(n) + h(n)$

A^* \rightarrow considera o custo acumulado + a heurística do nó

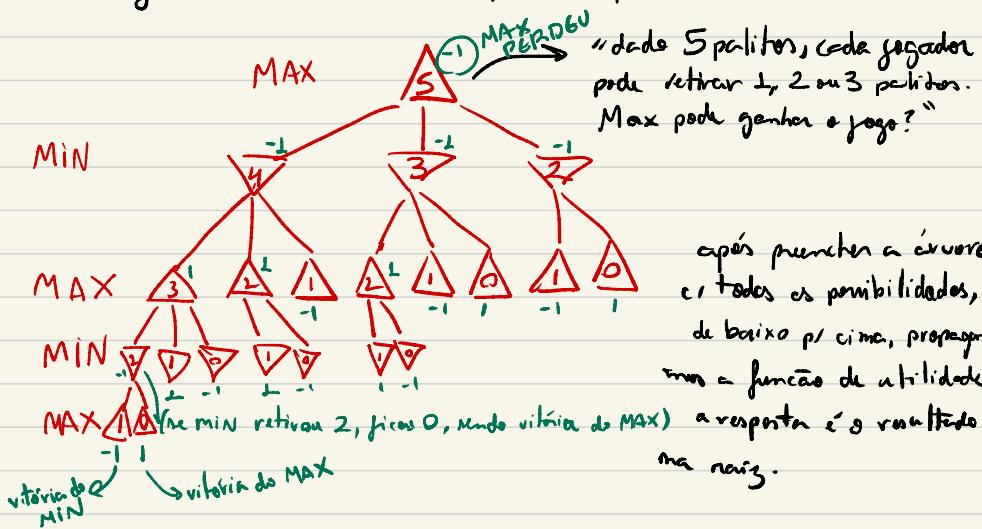
ordem: A, B, C, E, G, K



A* é o + rápido (p/ uma dada heurística, menor alg. vai expandir menos nós p/ encontrar a solução ótima)

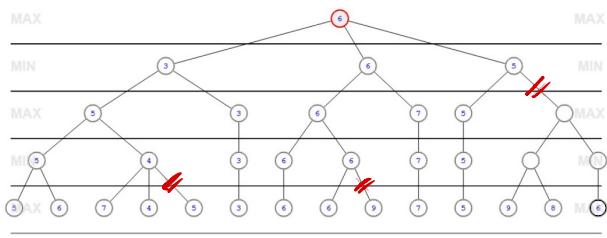
MINIMAX

- "jogo" com revezamento entre MIN e MAX
- melhor estratégia p/ jogos determinísticos
- gerarmos a árvore inteira até os estados terminais.



Poda alpha-beta: usado p/ melhorar a eficiência do MINIMAX, pedindo onde a busca seria irrelevante.

"não vale a pena piorar se já achou algo melhor"



Nó MIN (β) e há ancestrais MAX (α) = qualquer ancestral α é $\geq \beta$ que β deve ser min? → se sim, poda os demais ramos da min.

Nó MAX (α) e há ancestrais MIN (β) = qualquer ancestral β é $\leq \alpha$ que α deve ser max? → se sim, poda os demais ramos do max.

Heurística admissível: $\tilde{fute(n)}$

$$\forall n, h(n) < g(n)$$

para uma heurística ser admissível, a heurística deve ser menor que o custo real do nó até o objetivo (verificamos de baixo para cima na árvore),

Agrupamento

particionacional

hierárquica

por densidade

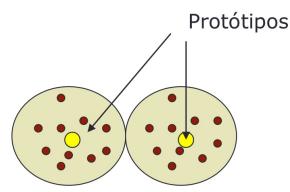
baseado em modelo

métodos que particionam
uma base de dados D de
n objetos em K grupos

* K-means = cada grupo é
representado pelo seu centro

Objetivo: atribuir clusters aos objetos dividindo-os, de forma que: maximize a similaridade dos objetos dentro de L cluster e minimize a similaridade entre clusters distintos.

o cluster é um conjunto dos objetos que são mais próximos ao protótipo que define o cluster (o centróide)



Um cluster é um conjunto de objetos no qual cada objeto está mais próximo ao protótipo que define o cluster do que dos protótipos de quaisquer outros clusters.

Em geral: Protótipo = centróide

Para avaliar se as instâncias estão no mesmo grupo:

Manhattan

$$d(x, y) = |x_1 - y_1| + |x_2 - y_2| + \dots + |x_p - y_p|$$

Minkowski

$$d(x, y) = \sqrt[m]{(x_1 - y_1)^m + (x_2 - y_2)^m + \dots + (x_p - y_p)^m}$$

Distância Euclidiana

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_p - y_p)^2}$$

$$\text{Manhattan} \rightarrow d(x_1, x_2) = |x_2 - x_1| + |y_2 - y_1|$$

\downarrow
 x_1, y_1 x_2, y_2

$$\text{Euclidiana} \rightarrow \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Algoritmo K-Means

Selecionar um número p/ K (ex=3)

1. Selecione k pontos como centroides iniciais

>Selectórios

Repetir

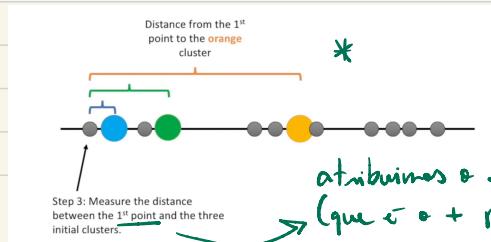
1. * Forme k clusters associando cada objeto a seu centroide mais próximo
2. Recalcule o centroide de cada cluster

→ (refazer tudo com pontos iniciada diferentes)

Até que Centroides não apresentarem mudanças

* os valores devem estar na mesma escala (normalizar)

* outliers são um problema (se possível, removê-los).



atribuimos o 1º ponto ao cluster azul
(que é o + próximo a ele)

* how similar
to each other they
are.

Criterios de Avaliação (internos):

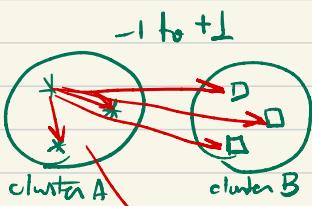
Silhouette - avalia a coesão e separação dos clusters, se baseando na diferença entre a distância média dos pontos pertencentes ao cluster + próximo p/ os pontos de L群组o

↳ intervalo $-1 \sim 1$

(cohesion* and separation)

$$\text{SilhouetteIndex} = \frac{1}{n} \sum_{i=1}^n S_i$$

value médio entre
todos os coeficientes
silhuetas de cada ponto



$$\frac{\min(\text{mean}(d_{\text{intra}})) - \text{mean}(d_{\text{inter}})}{\max(b, a)}$$

separation (b) - cohesion (a)

dos + próximos os pontos, e quanto eles
estão perto dele?

Sum of Squared Error (SSE) - p/ cada ponto, o erro é a dis-
tância ao grupo + próximo

$$SSE = \sum_{i=1}^k \sum_{p \in C_i} \text{dist}(p, c_i)^2$$

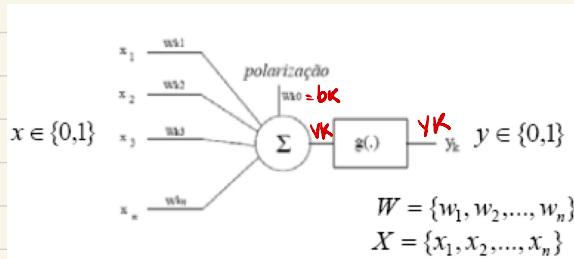
Redes Neurais

Unidades de processamento simples que armazenam conhecimento experimental (não tem capacidade de aprendizado).
- forças de conexão entre neurônios, pesos sinápticos, não servem p/ armazenar o conhecimento adquirido.

conectadas entre si

A aprendizagem: O objetivo é ajustar os pesos da rede p/ minimizar a medida de erro no conjunto de treinamento

McCulloch - Pitts:



$\left\{ \begin{array}{l} x_i \text{ é a excitação de entrada na sinapse } i; \\ y_k \text{ é a resposta (ou saída) do neurônio } k; \\ w_{ki} \text{ é o peso sináptico da entrada } i \text{ do neurônio } k; \\ g(.) \text{ é a função de ativação do neurônio.} \end{array} \right.$

* a func. de ativação deve ficar ativa p/ as respostas corretas (+ pos x de L) e inativa p/ as incorretas, e deve ser não-linear

- cada entrada tem um peso (intensidade)
- a unidade calcula uma soma ponderada das entradas
- aplica uma função de ativação a essa soma.
- w_{k0} (ou bias) é uma entrada fixa (-1 ou 1).

$$y_k = \begin{cases} 1 & \text{se } v_k \geq 0 \\ 0 & \text{se } v_k < 0 \end{cases}$$

somatório

capacidade de aprendizado pés puros serem ajustáveis.

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

Rosenblatt: Propôs o Perceptron, um neurônio de McCulloch, função Limiar e Aprendizado Supervisionado.

→ um único perceptron pode separar somente conjuntos de exemplo linearmente separáveis.

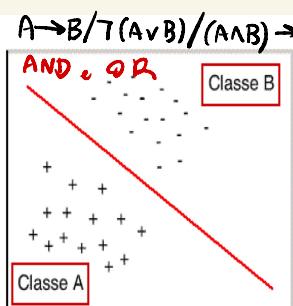


Fig. 1: Classes lineramente separáveis

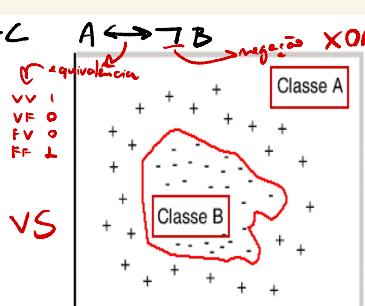
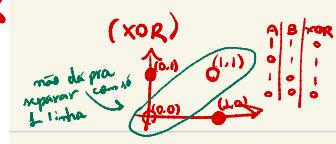


Fig. 2: Classes não linearmente separáveis



Treinamento c/ algoritmo Perceptron:

Algoritmo 7.1 Algoritmo de treinamento da rede perceptron

Entrada: Um conjunto de n objetos de treinamento

Saída: Rede perceptron com valores dos pesos ajustados

- ```

1 Inicializar pesos da rede com valores baixos
2 repita
3 para cada objeto x_i do conjunto de treinamento faça
4 Calcular valor da saída produzida pelo neurônio, $\hat{f}(x_i)$
5 Calcular erro = $y_i - \hat{f}(x_i)$
6 se erro > 0 então
7 Ajustar pesos do neurônio utilizando Equação 7.2
8 fim
9 fim
10 até erro = 0;

```

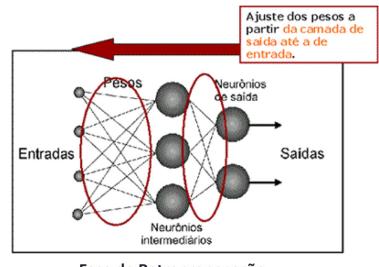
$$7.2: \quad w_j(t+1) = w_j(t) + \eta x_i^j (y_i - \hat{f}(\mathbf{x}_i))$$

$$peso(n+1) = peso(n) + (\text{taxa de aprendizagem} \times \text{entrada} \times \text{erro})$$

$\downarrow$  com parâmetro

**Para resolver problemas mais linearmente separáveis, adicionar-se-á uma ou mais camadas intermédias de neurônios e uma camadaativa de saída. (MLP - Multi Layer Perceptron)**

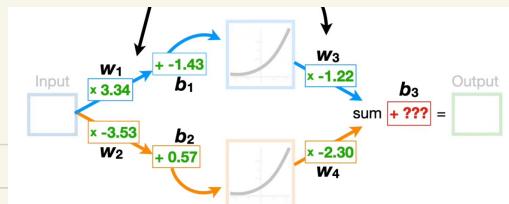
para treinar uma MLP, é necessário o Backpropagation, algoritmo para conexão de pesos.



# Backpropagation

## Atualização dos pesos:

- ↳ atualiza-se os pesos a cada instância classificada incorretamente (Stochastic gradient descent)
- ↳ atualiza-se os pesos ao final de todos os instâncias apresentadas (Batch gradient descent)



**Algoritmo 7.2** Algoritmo de treinamento *back-propagation*

```

Entrada: Um conjunto de n objetos de treinamento
Saída: Rede MLP com valores dos pesos ajustados
1 Inicializar pesos da rede com valores aleatórios
2 Inicializar $erro_{total} = 0$
3 repita
4 para cada objeto x_i do conjunto de treinamento faça
5 para cada camada da rede, a partir da primeira camada intermediária faça
6 para cada neurônio n_{jl} da camada atual faça
7 Calcular valor da saída produzida pelo neurônio, \hat{f}
8 fim
9 fim
10 Calcular $erro_{parcial} = y - \hat{f}$
11 para cada camada da rede, a partir da camada de saída faça
12 para cada neurônio n_{jl} da camada atual faça
13 Ajustar pesos do neurônio utilizando Equação 7.3
14 fim
15 fim
16 Calcular $erro_{total} = erro_{total} + erro_{parcial}$
17 fim
18 até $erro_{total} < \xi$:

```

$$w_{jl}(t+1) = w_{jl}(t) + \eta x^j \delta_l$$

heurísticos pr mº de neurônios:

$$q = \frac{p + M}{2}$$

Entrada → média dos entradas  
Valor médio

$$q = \sqrt{p \cdot M}$$

Entradas → Entradas  
Nardos → Nardos  
Raiz quadrada

$$q = 2p + 1$$

Entradas → Entradas  
Kolmogorov

A **redução de dimensionalidade** é uma técnica em mineração de dados e aprendizado de máquina que visa reduzir o número de variáveis (ou dimensões) em um conjunto de dados enquanto preserva ao máximo as informações relevantes.

# CNN

→ faz todo o processo num modelo único (já seleciona os característicos + importantes p/ aprendizagem).

→ funciona em 4 etapas:

1- operador de convolução

2- pooling

3- flattening

4- rede neural densa

(multiplica a

matriz de imagem  
pela matriz do Kernel)

1- Pega a imagem e multiplica os "pixels" por um detector de características (Kernel)

→ a matriz do Kernel é menor, então ela vai dando shift em cada coluna da imagem p/ multiplicar ela inteira.

→ depois, para esse mapa de características p/ uma função de ativação (ReLU)  $\rightarrow R(z) = \max(0, z) \rightarrow$  valores negativos de joga p/ 0, por exemplo.

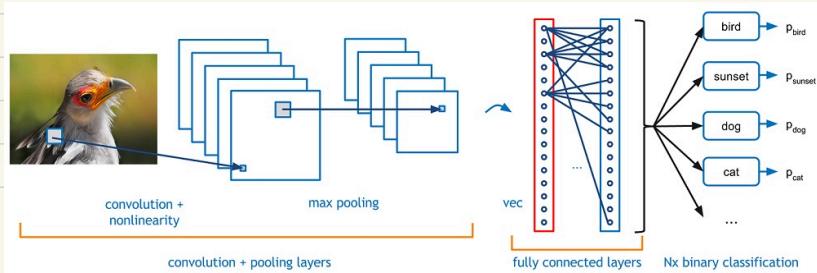
→ reduz + aianda os  
característicos

2- Pooling seleciona as características + relevantes da imagem (reduz overfitting + ruídos). Pode-se usar o máximo, o mínimo, a média dos valores, etc. Ela seleciona quais características vc quer.

$$\begin{matrix} 0 & 1 \\ 0 & 2 \end{matrix} \rightarrow 2$$

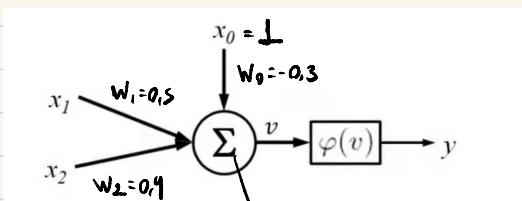
3- Pega os matrizes e transforma em vetores (transforma em forma linear)

4- pega os valores dos vetores nos entradas das Redes Neurais Densas



# Exercício (Perceptron)

## Perceptron



| $E_1$ | $E_2$ | $S_p$ | $S_F$ |
|-------|-------|-------|-------|
| 0     | 0     | -0,3  | 0     |
| 0     | 1     | 0,1   | 1     |
| 1     | 0     | 0,2   | 1     |
| 1     | 1     | 0,6   | 1     |

$\left. \right\} OR$

$$x_1 \cdot w_1 \\ x_2 \cdot w_2 \quad \left. \right\} x_1 \cdot w_1 + x_2 \cdot w_2 + x_0 \cdot w_0$$

$\downarrow -0,3$   
bias peso de bias

$$0 \cdot 0 = 0 \cdot 0,5 + 0 \cdot 0,4 + 1 \cdot -0,3 = \underline{-0,3} \quad ( < 0 = 0 )$$

$$0 \cdot 1 = 0 \cdot 0,5 + 1 \cdot 0,4 + 1 \cdot -0,3 = 0,1 \quad (> 0 = 1)$$

$$1 \cdot 0 = 1 \cdot 0,5 + 0 \cdot 0,4 + 1 \cdot -0,3 = 0,2 \quad (> 0 = 1)$$

$$1 \cdot 1 = 1 \cdot 0,5 + 1 \cdot 0,4 + 1 \cdot -0,3 = 0,6 \quad (> 0 = 1)$$

\* Imagine que os resultados de 0 e 1 e 1 e 0 estavam errados (e na verdade eu queria fazer um AND). Podemos fazer o treinamento do Perceptron

1- P/ cada peso  $w_i \Rightarrow w_i \leftarrow w_i + \eta \cdot (y_{verdadeiro} - y) \cdot x_i$   
 ↓ taxa de aprendizagem

2- Atualize o bias  $b \leftarrow b + \eta \cdot (y_{verdadeiro} - y)$

↓ usar aprendizados pequenos!

3- Passe por todos os exemplos de cunhado (uma época) e continue iterando até que todos os erros sejam atingidos.

$1 \cdot 0 \Rightarrow y_{true} = 0 ; y = 1$  (erro)  $\Rightarrow$  atualização dos pesos:

$$w_1 = 0,5 + (0,1 \cdot 1 \cdot 1) = 0,4 \quad \left. \right\} (0-1)$$

$$w_2 = 0,4 + 0,1 \cdot 1 \cdot 0 = 0,4 \quad \left. \right\} \text{novos pesos}$$

$$w_3 = -0,3 + 0,1 \cdot 1 \cdot -1 = -0,4 \quad \left. \right\} \text{novos pesos}$$

↳ continue esse processo até que o perceptron aprenda a função AND.

# Tabela de Diferenças:

| Característica            | McCulloch e Pitts            | Perceptron                                      | MLP com Backpropagation                                                     |
|---------------------------|------------------------------|-------------------------------------------------|-----------------------------------------------------------------------------|
| Capacidade de aprendizado | Não aprende (pesos fixos)    | Aprende ajustando os pesos                      | Aprende ajustando pesos em várias camadas por gradiente descendente         |
| Tipo de saída             | Binária (0 ou 1)             | Binária (0 ou 1)                                | Contínua ou categórica (via funções de ativação como sigmoid ou softmax)    |
| Propósito original        | Simular lógica booleana      | Reconhecimento de padrões lineares              | Reconhecimento de padrões não lineares e complexos                          |
| Arquitetura               | Apenas 1 neurônio            | Apenas 1 neurônio                               | Múltiplas camadas de neurônio (entrada, ocultas e saída)                    |
| Resolução de problemas    | Operações lógicas simples    | Apenas padrões linearmente separáveis (AND, OR) | Padrões linearmente e não linearmente separáveis (e.g., XOR) ( $\times$ OR) |
| Algoritmo de aprendizado  | Nenhum                       | Regra de aprendizado do Perceptron              | Backpropagation (baseado em gradiente descendente)                          |
| Funções de ativação       | Limiar (hard limit)          | Limiar (hard limit)                             | Funções não lineares como sigmoid, ReLU, ou tanh                            |
| Limitação                 | Pesos fixos, sem aprendizado | Não resolve problemas não lineares              | Pode sofrer overfitting, exige mais dados e poder computacional             |