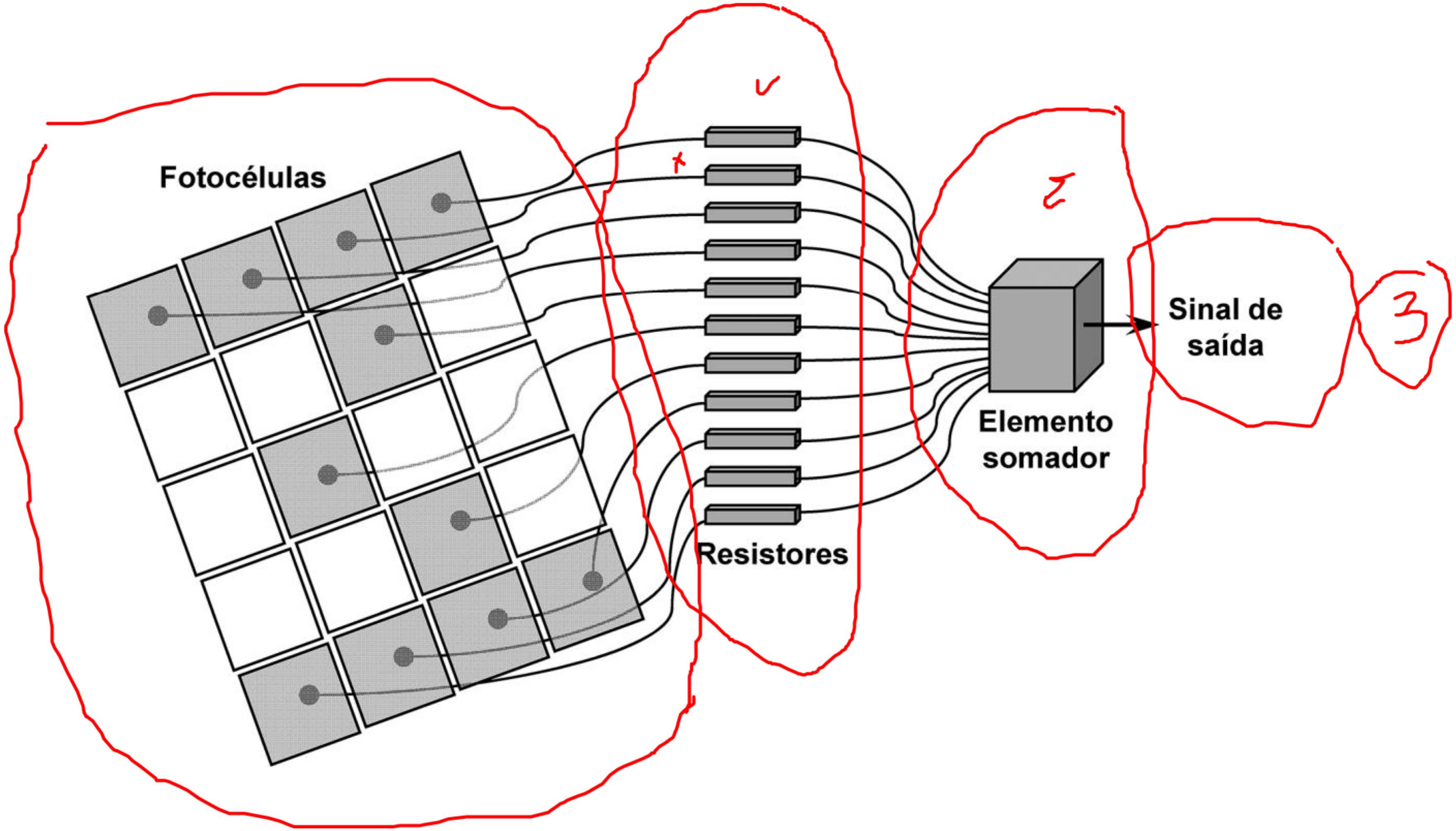
Rede Perceptron

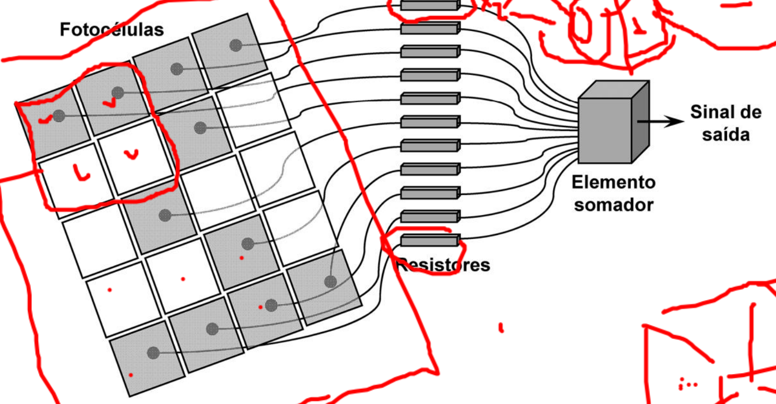
Residência de IA

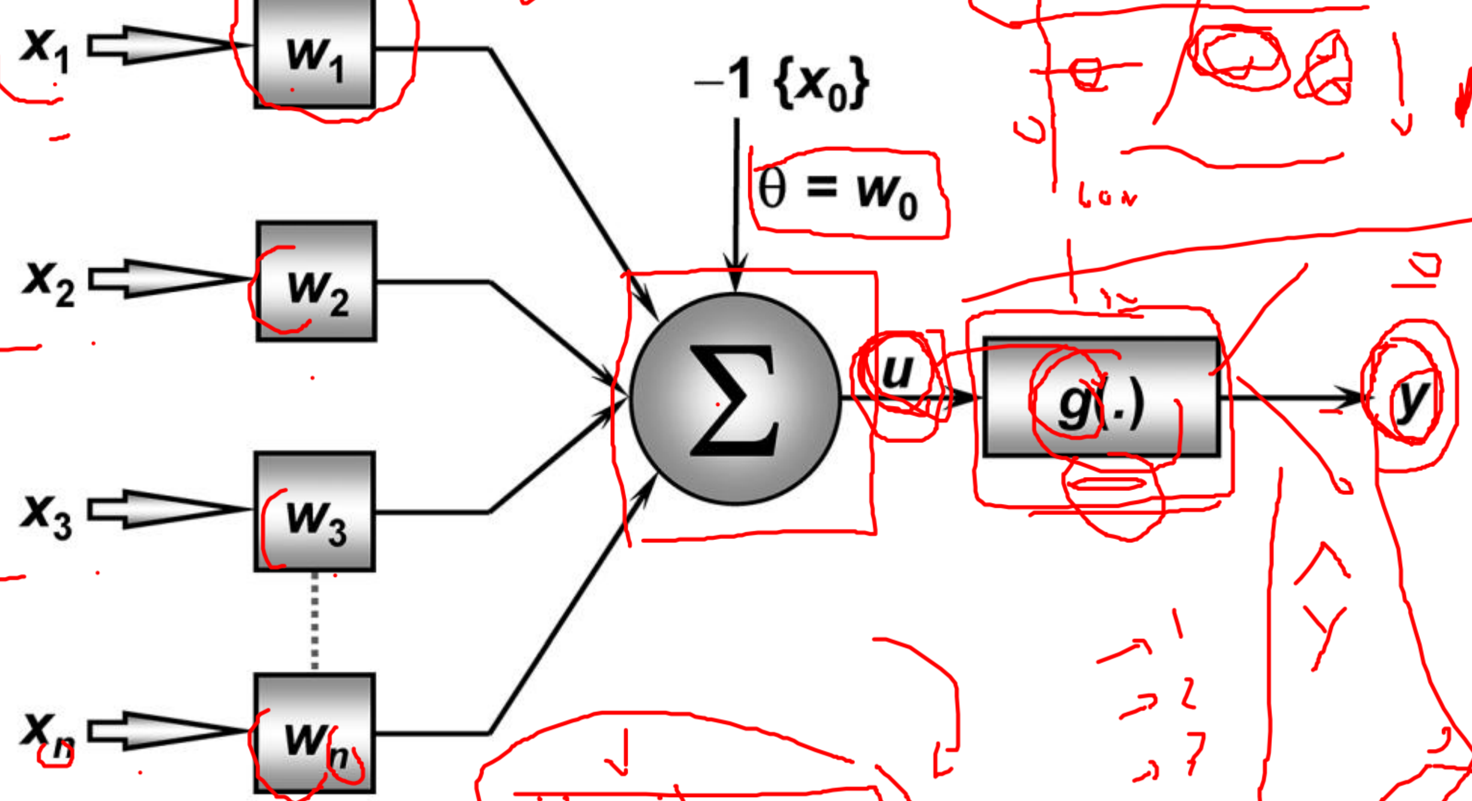
Perceptron

3



Perceptron



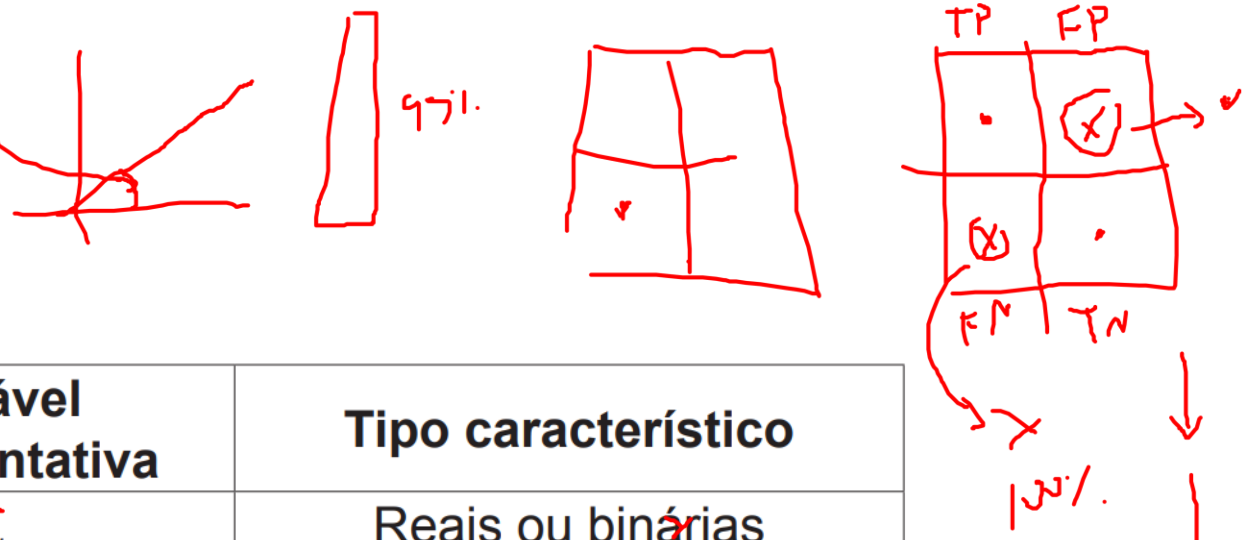


$$\begin{cases} u = \sum_{i=1}^{n} w_i \cdot x_i - \theta \\ y = g(u) \end{cases}$$

PANN

Perceptron

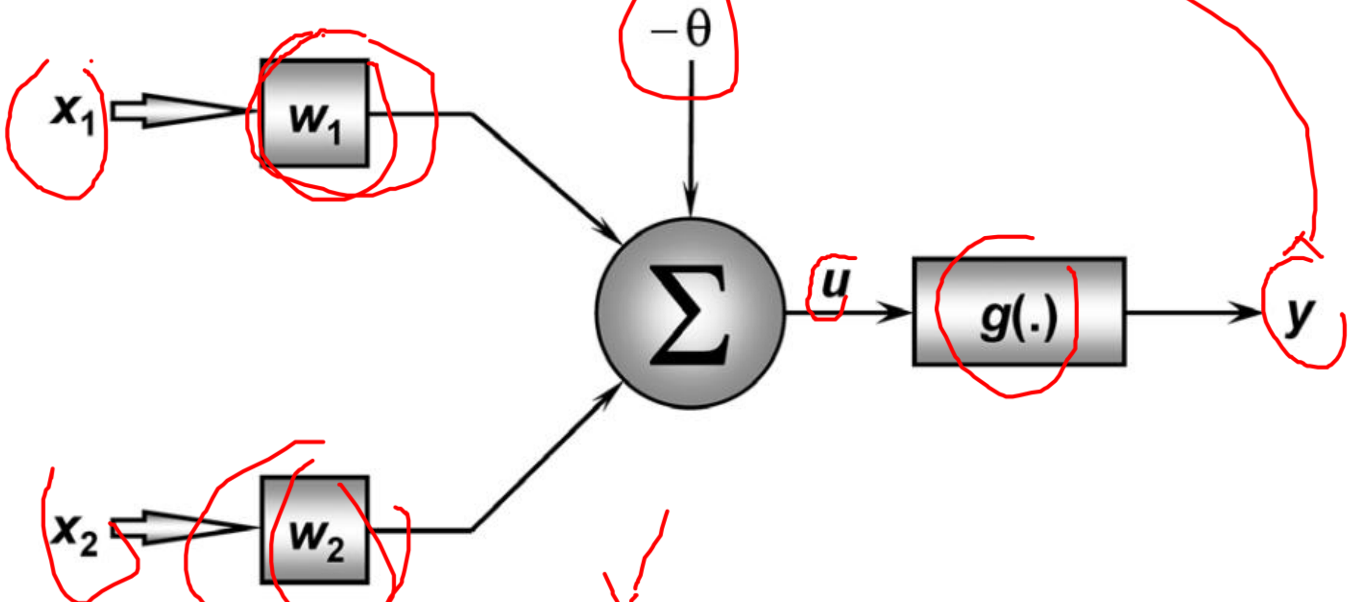
$y = b + xw$



Parâmetro	Variável representativa	Tipo característico
Entradas	x_i (*i*-ésima entrada)	Reais ou binárias (advindas externamente)
Pesos sinápticos	w_i (associado a x_i)	Reais (iniciados aleatoriamente)
Limiar	θ	Real (iniciado aleatoriamente)
Saída	y	Binária
Função de ativação	$g(.)$	Degrau ou degrau bipolar
Processo de treinamento	-----	Supervisionado
Regra de aprendizado	-----	Regra de Hebb

Perceptron

$\hat{y} = \sum_{i=1}^{n}$



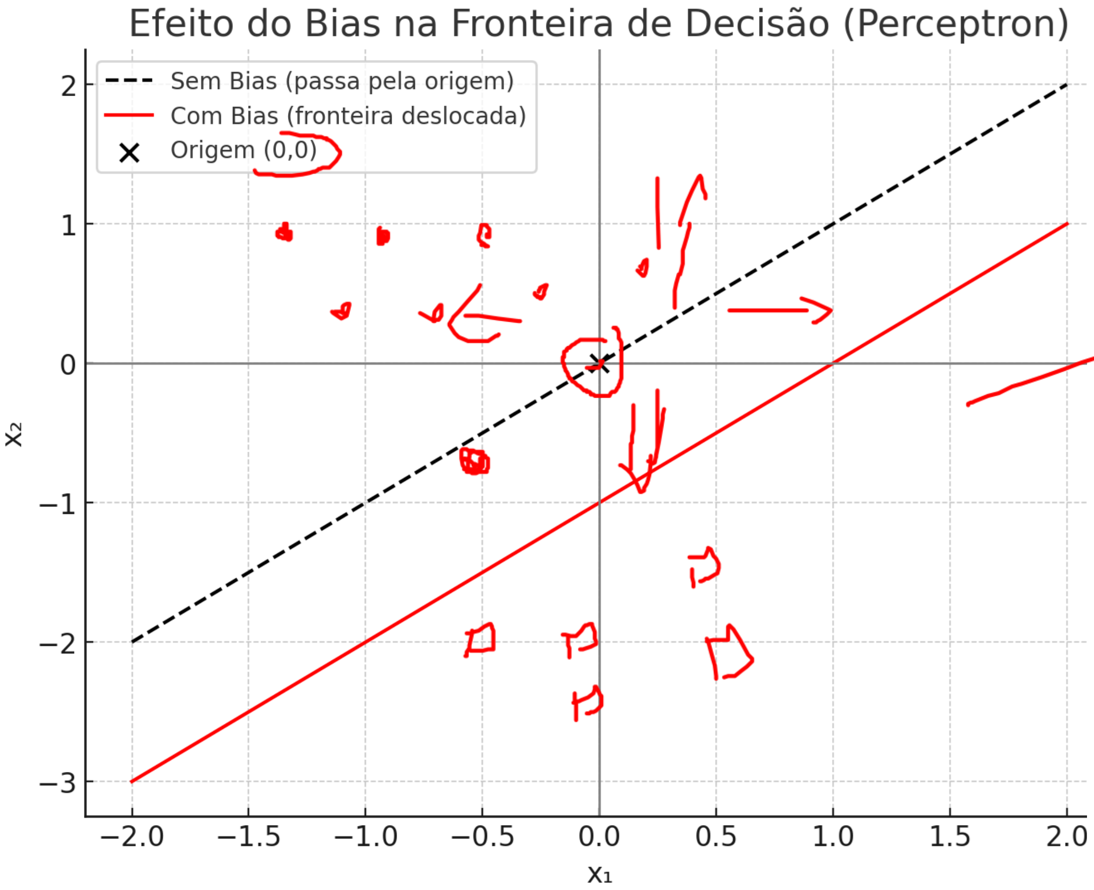
$$y = \begin{cases} 1, & \text{se } \sum w_i \cdot x_i - \theta \geq 0 \Leftrightarrow w_1 \cdot x_1 + w_2 \cdot x_2 - \theta \geq 0 \\ -1, & \text{se } \sum w_i \cdot x_i - \theta < 0 \Leftrightarrow w_1 \cdot x_1 + w_2 \cdot x_2 - \theta < 0 \end{cases}$$

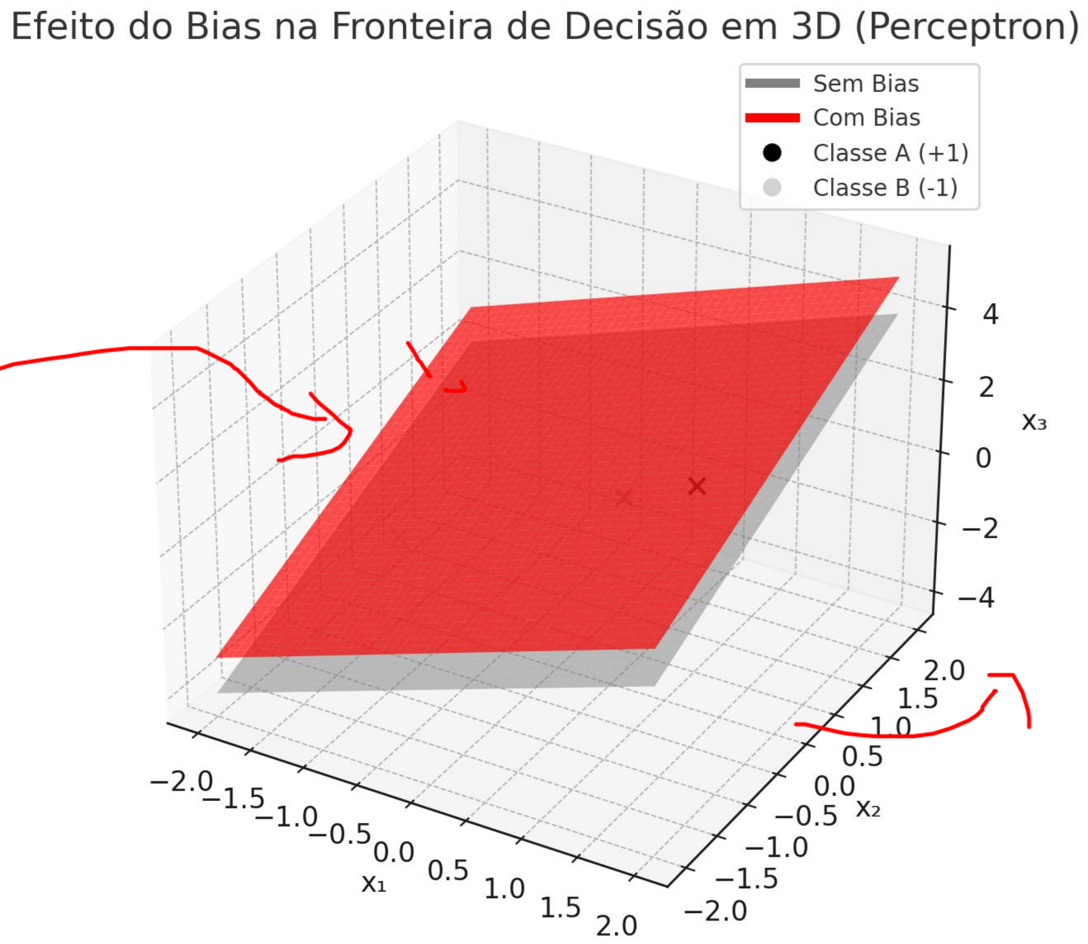
Bias

- Sem o bias, o modelo só consegue gerar hiperplanos (retas, planos, etc.) que passam pela origem (0,0,...,0). **Isso limita muito o que ele pode aprender**. Com o bias, o hiperplano pode se deslocar no espaço, permitindo separar dados que não estão alinhados com a origem.
- Por exemplo:
  - Sem bias → reta
    - y=2x sempre passa pela origem.
  - Com bias → reta
    - y=2x+3 pode se mover para cima ou para baixo, ajustando o ponto de corte.
- Conclusão: o bias é o que permite flexibilidade no posicionamento da fronteira de decisão.

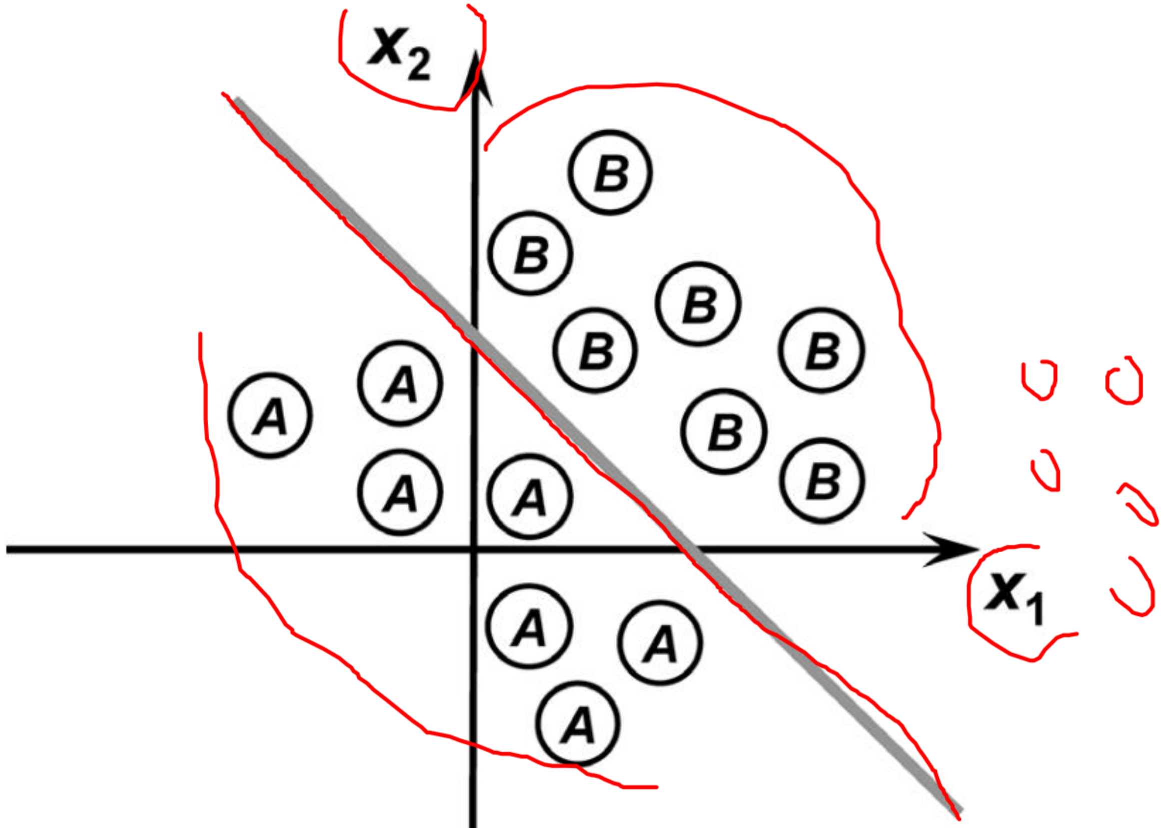
Bias

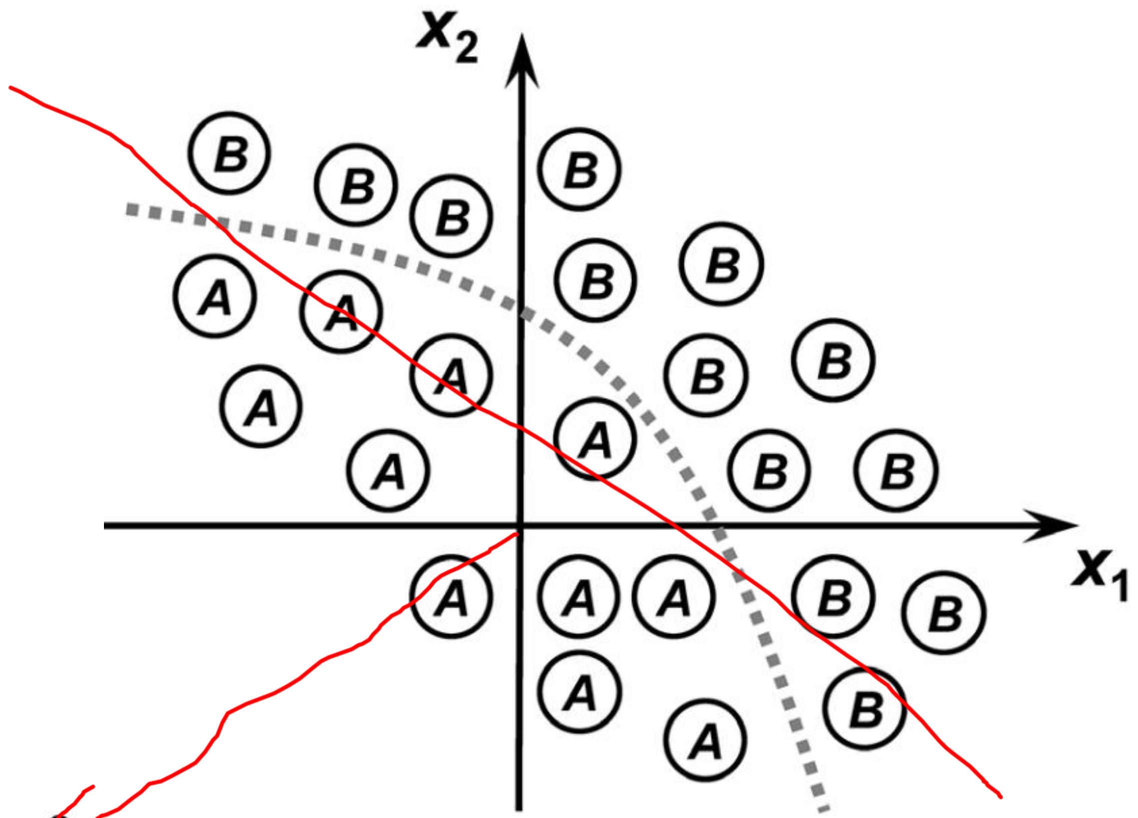
Redes modernas não precisam explicitamente do bias porque outras camadas já cumprem o mesmo papel.





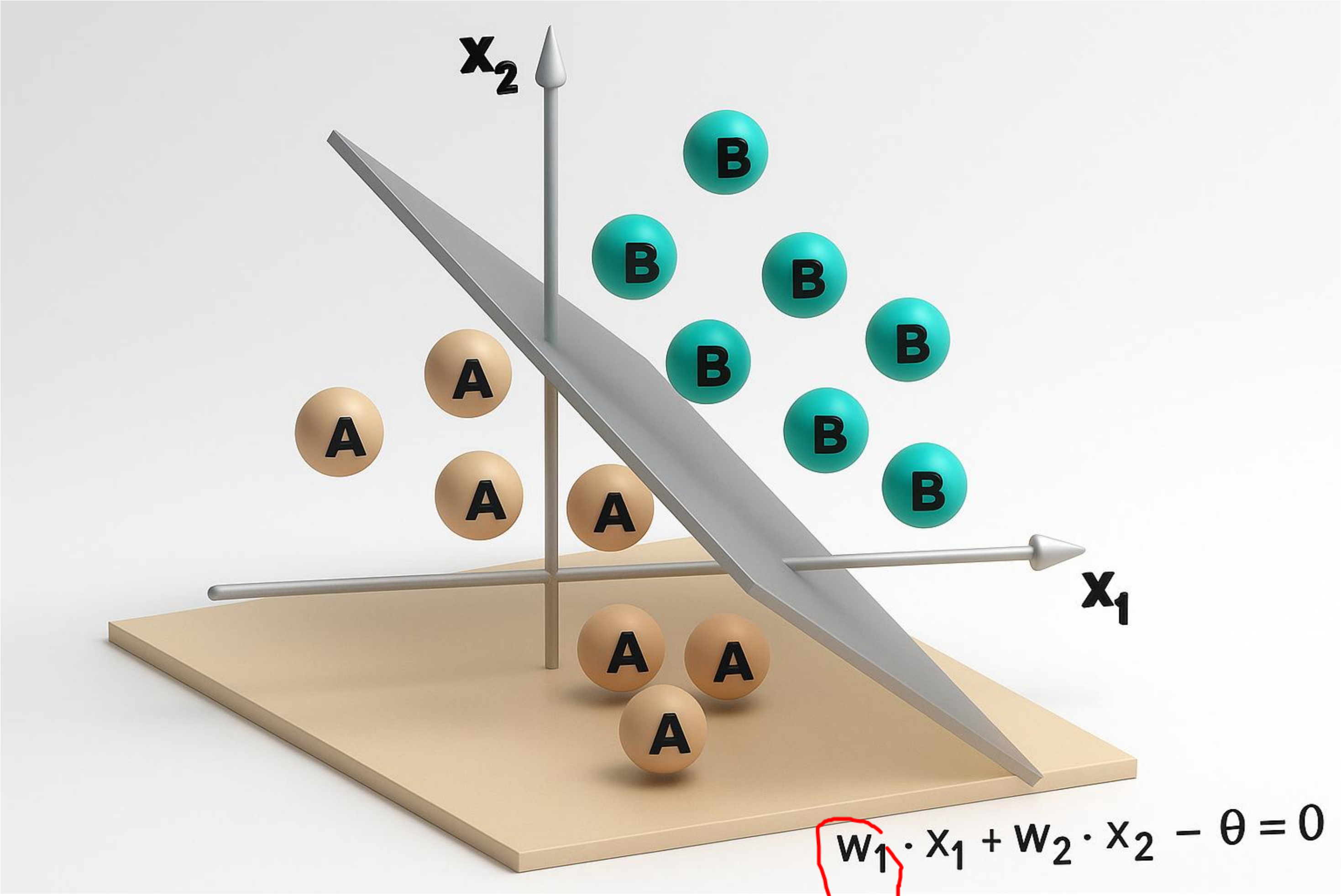
Separabilidade

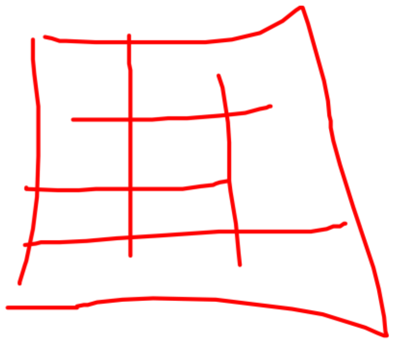




$$w_1 \cdot x_1 + w_2 \cdot x_2 - \theta = 0$$

Separabilidade





$+ w_3 \cdot x_3$

Hiperparâmetros

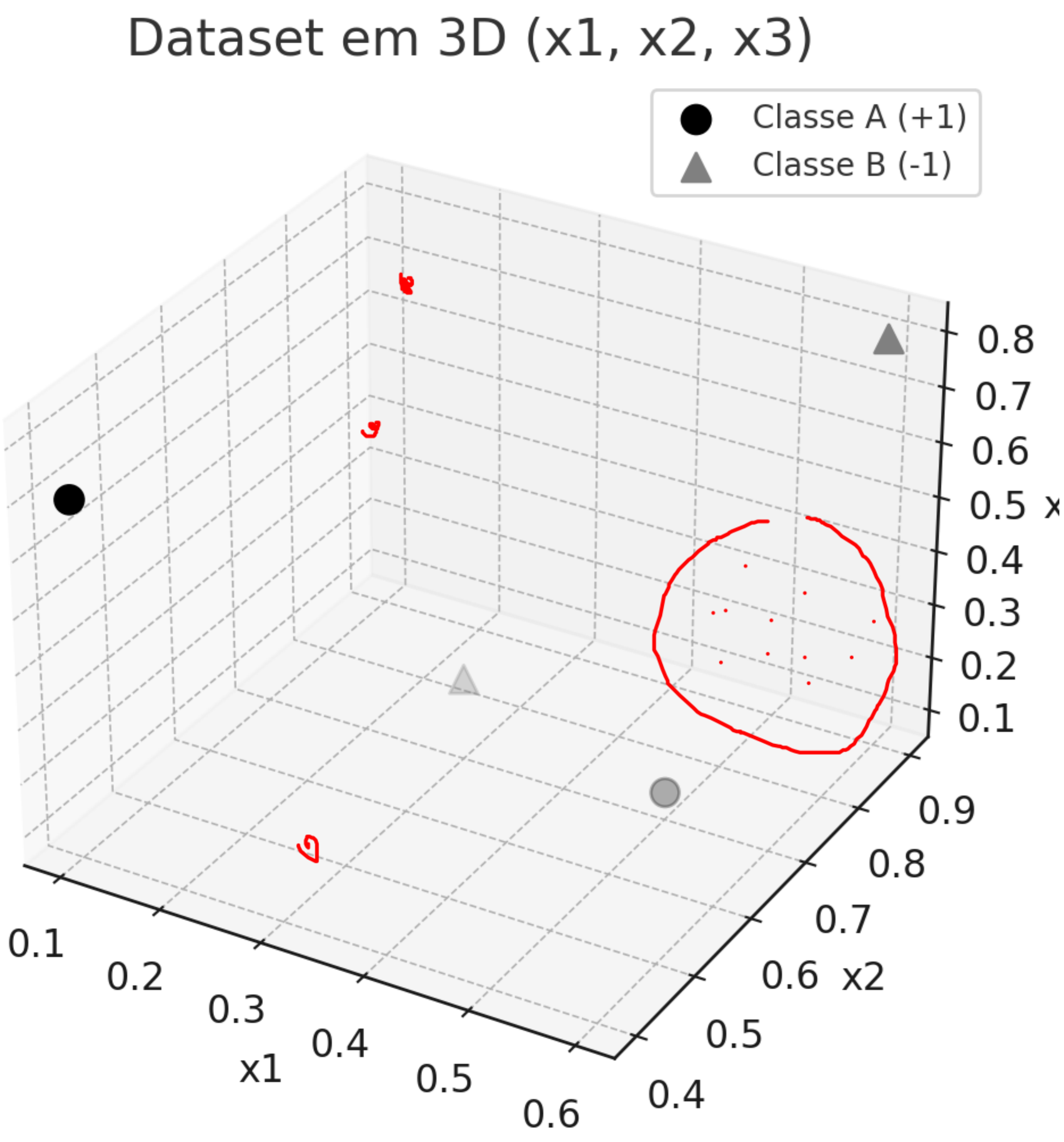
- **Taxa de aprendizado** (η): controla o tamanho do ajuste dos pesos.
- **Época** (epoch): uma passagem completa sobre o conjunto de treinamento.
- **Convergência**: ponto em que o algoritmo encontra pesos que classificam corretamente todas as amostras (se possível).

Treinamento

$$\begin{array}{c} \\ \Omega^{(\boldsymbol{x})} = \end{array} \begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{cccc} \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \boldsymbol{x}^{(3)} & \boldsymbol{x}^{(4)} \\ \end{array}$$

$$\Omega^{(\boldsymbol{x})} = \begin{array}{c|cccc} & \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \boldsymbol{x}^{(3)} & \boldsymbol{x}^{(4)} \\ \hline x_0 & -1 & -1 & -1 & -1 \\ x_1 & 0{,}1 & 0{,}3 & 0{,}6 & 0{,}5 \\ x_2 & 0{,}4 & 0{,}7 & 0{,}9 & 0{,}7 \\ x_3 & 0{,}7 & 0{,}2 & 0{,}8 & 0{,}1 \end{array};$$

$$\Omega^{(\boldsymbol{d})} = \begin{array}{cccc} d^{(1)} & d^{(2)} & d^{(3)} & d^{(4)} \\ \hline 1 & -1 & -1 & 1 \end{array}$$



Treinamento

Início {Algoritmo *Perceptron* – Fase de Treinamento}

<1> Obter o conjunto de amostras de treinamento { $\boldsymbol{x}^{(k)}$ };

<2> Associar a saída desejada { $d^{(k)}$ } para cada amostra obtida;

<3> Iniciar o vetor $\boldsymbol{w}$ com valores aleatórios pequenos;

<4> Especificar a taxa de aprendizagem { η };

<5> Iniciar o contador de número de épocas { $época \leftarrow 0$ };

<6> Repetir as instruções:

- <6.1> $erro \leftarrow$ "inexiste";
- <6.2> Para todas as amostras de treinamento { $\boldsymbol{x}^{(k)}$, $d^{(k)}$ }, fazer:
  - <6.2.1> $u \leftarrow \boldsymbol{w}^T \cdot \boldsymbol{x}^{(k)}$;
  - <6.2.2> $y \leftarrow \text{sinal}(u)$;
  - <6.2.3> Se $y \neq d^{(k)}$
    - <6.2.3.1> Então $\begin{cases} \boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \cdot (d^{(k)} - y) \cdot \boldsymbol{x}^{(k)} \\ erro \leftarrow \text{"existe"} \end{cases}$
- <6.3> $época \leftarrow época + 1$;

Até que: $erro$ = "inexiste"

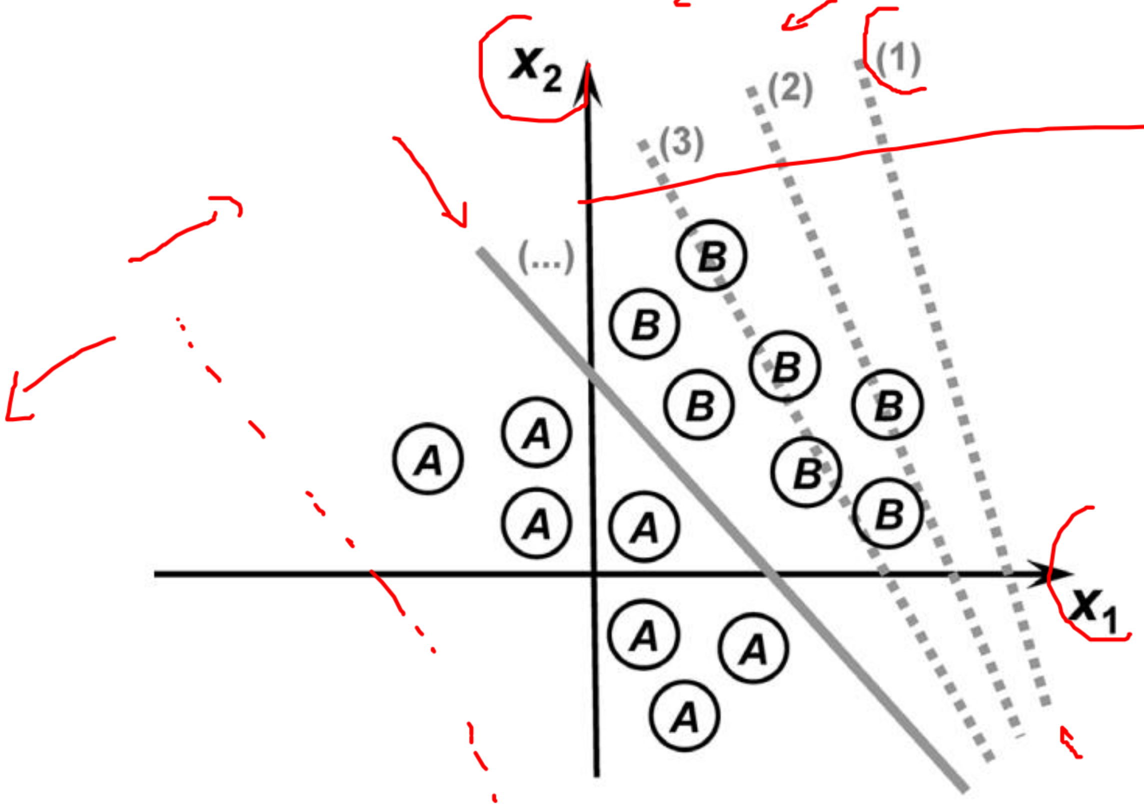
Fim {Algoritmo *Perceptron* – Fase de Treinamento}

```python
# <1> e <2> Conjunto de amostras e saídas desejadas
X = np.array([
    [-1.0, -1.0, -1.0, -1.0],  # x0 (viés)
    [ 0.1,  0.3,  0.6,  0.5],  # x1
    [ 0.4,  0.7,  0.9,  0.7],  # x2
    [ 0.7,  0.2,  0.8,  0.1],  # x3
], dtype=float)

d = np.array([ 1,  1, -1,  1], dtype=int)  # saídas desejadas

# <3> Inicializar pesos com valores aleatórios pequenos
rng = np.random.default_rng(42)
w = rng.uniform(-0.05, 0.05, size=(X.shape[0],))

# <4> Taxa de aprendizagem
eta = 0.1

# Função sinal
def sinal(u):
    return 1 if u >= 0 else -1

# <5> Inicializar contador de épocas
epoca = 0
historico = []
```

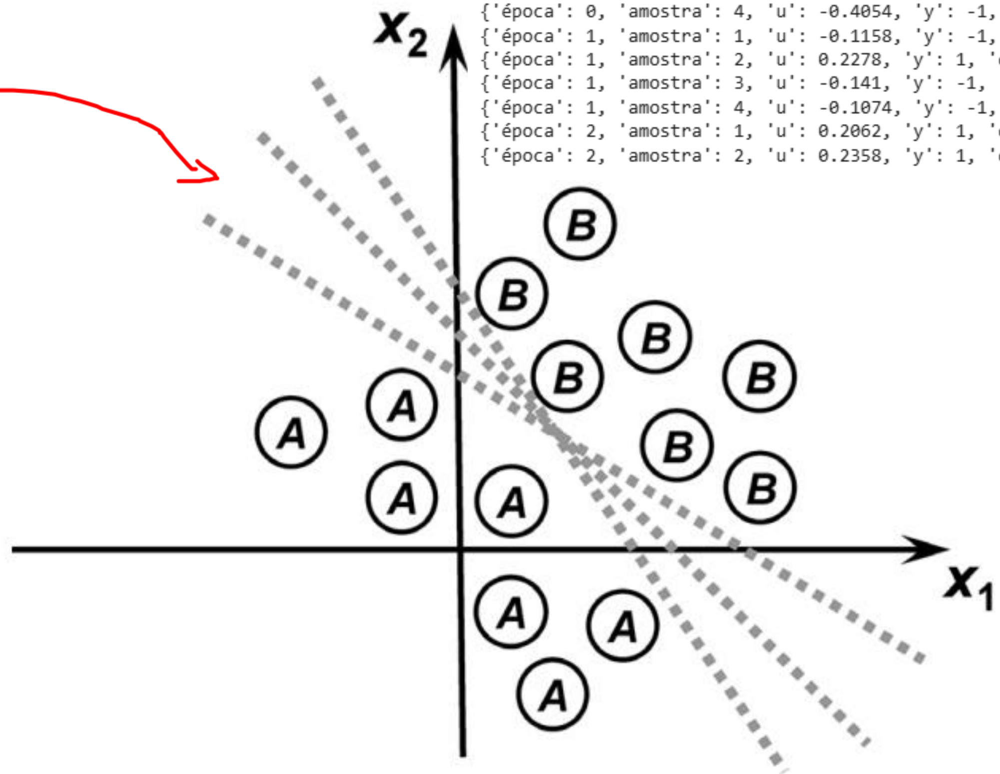
```
Treinamento concluído.
Épocas executadas: 102
Pesos finais w*:
[-0.3726044   1.65388784 -1.72414021  0.2197368 ]

Classificação final das amostras:
x^1: u=0.0022, y=1, d=1
x^2: u=-0.2942, y=-1, d=-1
x^3: u=-0.0110, y=-1, d=-1
x^4: u=0.0146, y=1, d=1

Resumo das primeiras 10 atualizações:
{'época': 0, 'amostra': 1, 'u': 0.0002, 'y': 1, 'd': np.int64(1)}
{'época': 0, 'amostra': 2, 'u': -0.0002, 'y': -1, 'd': np.int64(-1)}
{'época': 0, 'amostra': 3, 'u': 0.017, 'y': 1, 'd': np.int64(-1)}
{'época': 0, 'amostra': 4, 'u': -0.4054, 'y': -1, 'd': np.int64(1)}
{'época': 1, 'amostra': 1, 'u': -0.1158, 'y': -1, 'd': np.int64(1)}
{'época': 1, 'amostra': 2, 'u': 0.2278, 'y': 1, 'd': np.int64(-1)}
{'época': 1, 'amostra': 3, 'u': -0.141, 'y': -1, 'd': np.int64(-1)}
{'época': 1, 'amostra': 4, 'u': -0.1074, 'y': -1, 'd': np.int64(1)}
{'época': 2, 'amostra': 1, 'u': 0.2062, 'y': 1, 'd': np.int64(1)}
{'época': 2, 'amostra': 2, 'u': 0.2358, 'y': 1, 'd': np.int64(-1)}
```

```python
# <6> Loop de treinamento
while True:
    erro_existe = False  # <6.1> erro ← "inexiste"

    # <6.2> Para todas as amostras de treinamento
    for k in range(X.shape[1]):
        xk = X[:, k]
        dk = d[k]

        # <6.2.1> u ← w^T . x^(k)
        u = float(np.dot(w, xk))

        # <6.2.2> y ← sinal(u)
        y = sinal(u)

        w_antes = w.copy()

        # <6.2.3> Se y ≠ d^(k)
        if y != dk:
            w = w + eta * (dk - y) * xk
            erro_existe = True

        historico.append({
            "época": epoca,
            "amostra": k + 1,
            "u": u,
            "y": y,
            "d": dk,
            "w_antes": w_antes,
            "w_depois": w.copy()
        })

    # <6.3> época ← época + 1
    epoca += 1

    # Até que erro = "inexiste"
    if not erro_existe:
        break
```

Treinamento




```
Treinamento concluído.
Épocas executadas: 102
Pesos finais w*:
[-0.3726044   1.65388784 -1.72414021  0.2197368 ]

Classificação final das amostras:
x^1: u=0.0022, y=1, d=1
x^2: u=-0.2942, y=-1, d=-1
x^3: u=-0.0110, y=-1, d=-1
x^4: u=0.0146, y=1, d=1

Resumo das primeiras 10 atualizações:
{'época': 0, 'amostra': 1, 'u': 0.0002, 'y': 1, 'd': np.int64(1)}
{'época': 0, 'amostra': 2, 'u': -0.0002, 'y': -1, 'd': np.int64(-1)}
{'época': 0, 'amostra': 3, 'u': 0.017, 'y': 1, 'd': np.int64(-1)}
{'época': 0, 'amostra': 4, 'u': -0.4054, 'y': -1, 'd': np.int64(1)}
{'época': 1, 'amostra': 1, 'u': -0.1158, 'y': -1, 'd': np.int64(1)}
{'época': 1, 'amostra': 2, 'u': 0.2278, 'y': 1, 'd': np.int64(-1)}
{'época': 1, 'amostra': 3, 'u': -0.141, 'y': -1, 'd': np.int64(-1)}
{'época': 1, 'amostra': 4, 'u': -0.1074, 'y': -1, 'd': np.int64(1)}
{'época': 2, 'amostra': 1, 'u': 0.2062, 'y': 1, 'd': np.int64(1)}
{'época': 2, 'amostra': 2, 'u': 0.2358, 'y': 1, 'd': np.int64(-1)}
```



Treinamento

- O Perceptron aprende porque ajusta seus pesos toda vez que erra, movendo o limite de decisão na direção correta para corrigir o erro.
  - Calcula: $u = w \cdot x$
  - Decide: $y = \text{sinal}(u)$
  - Compara com a saída desejada d
  - Se errou $(y \neq d)$: $w \leftarrow w + \eta(d - y)x$

Situação	**d**	**y**	**d - y**	**Movimento do w**
Errou um positivo	+1	-1	+2	move **na direção de x**
Errou um negativo	-1	+1	-2	move **contra x**

Operação

Início {Algoritmo *Perceptron* – Fase de Operação}

<1> Obter uma amostra a ser classificada { $\boldsymbol{x}$ };

<2> Utilizar o vetor $\boldsymbol{w}$ ajustado durante o treinamento;

<3> Executar as seguintes instruções:

<3.1> $u \leftarrow \boldsymbol{w}^T \cdot \boldsymbol{x}$;

<3.2> $y \leftarrow \text{sinal}(u)$;

<3.3> Se $y = -1$

<3.3.1> Então: amostra $\boldsymbol{x} \in$ *{Classe A}*

<3.4> Se $y = 1$

<3.4.1> Então: amostra $\boldsymbol{x} \in$ *{Classe B}*

Fim {Algoritmo *Perceptron* – Fase de Operação}

```python
import numpy as np

# -------------------------------------------
# Função sinal(u): retorna 1 se u ≥ 0; caso contrário, -1
# -------------------------------------------
def sinal(u: float) -> int:
    return 1 if u >= 0 else -1

# ---------------------------------------------------------
# Algoritmo Perceptron — Fase de Operação (versão fiel)
# ---------------------------------------------------------
def perceptron_fase_operacao(x: np.ndarray, w: np.ndarray):
    # <3.1>
    u = float(np.dot(w, x))
    # <3.2>
    y = sinal(u)
    # <3.3> e <3.4>
    if y == -1:
        classe = "Classe A"
    elif y == 1:
        classe = "Classe B"
    else:
        # Não ocorre para sinal clássico, mas deixamos por completude
        classe = "Indefinido"
    return classe, y, u
```

```
x^1: u=0.0022, y=1, -> Classe B
x^2: u=-0.2942, y=-1, -> Classe A
x^3: u=-0.0110, y=-1, -> Classe A
x^4: u=0.0146, y=1, -> Classe B
```

Taxa de Aprendizagem

- A taxa de aprendizagem (ou learning rate, denotada por η) é um parâmetro fundamental no treinamento de algoritmos como o **Perceptron**, pois ela **controla o tamanho dos ajustes feitos nos pesos a cada iteração do aprendizado**.

$x_1 \rightarrow 0{,}17$ 1 $\hat{o}$

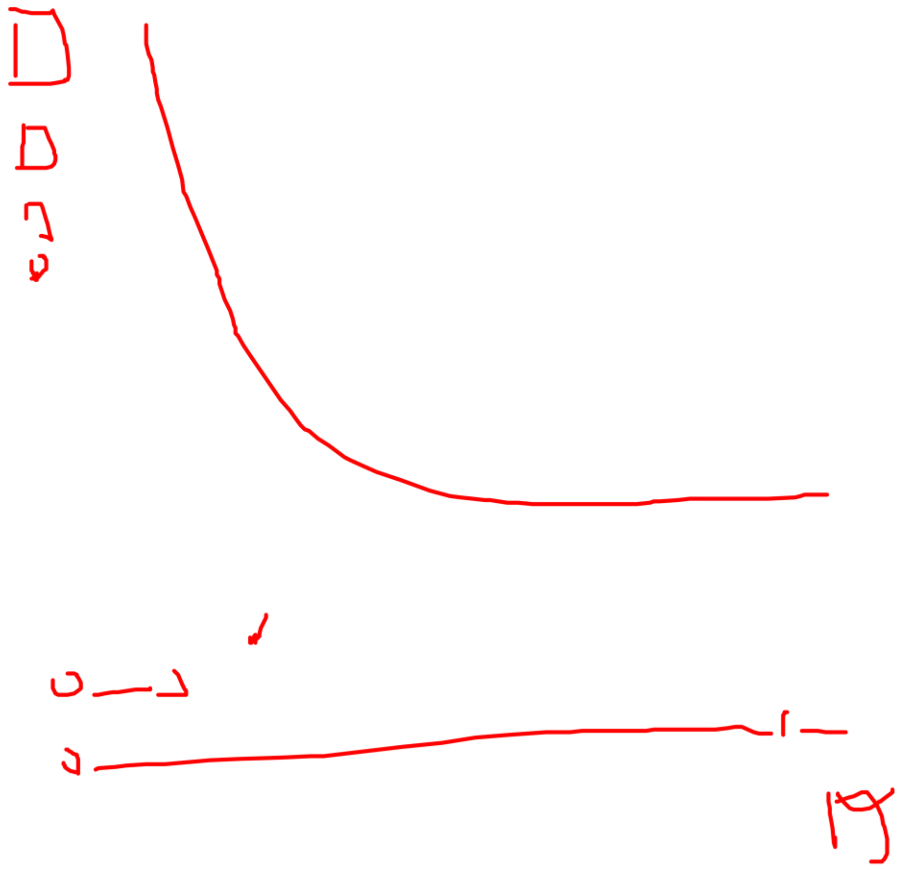
$x_2 \rightarrow 0.43$ 4

Taxa de Aprendizagem

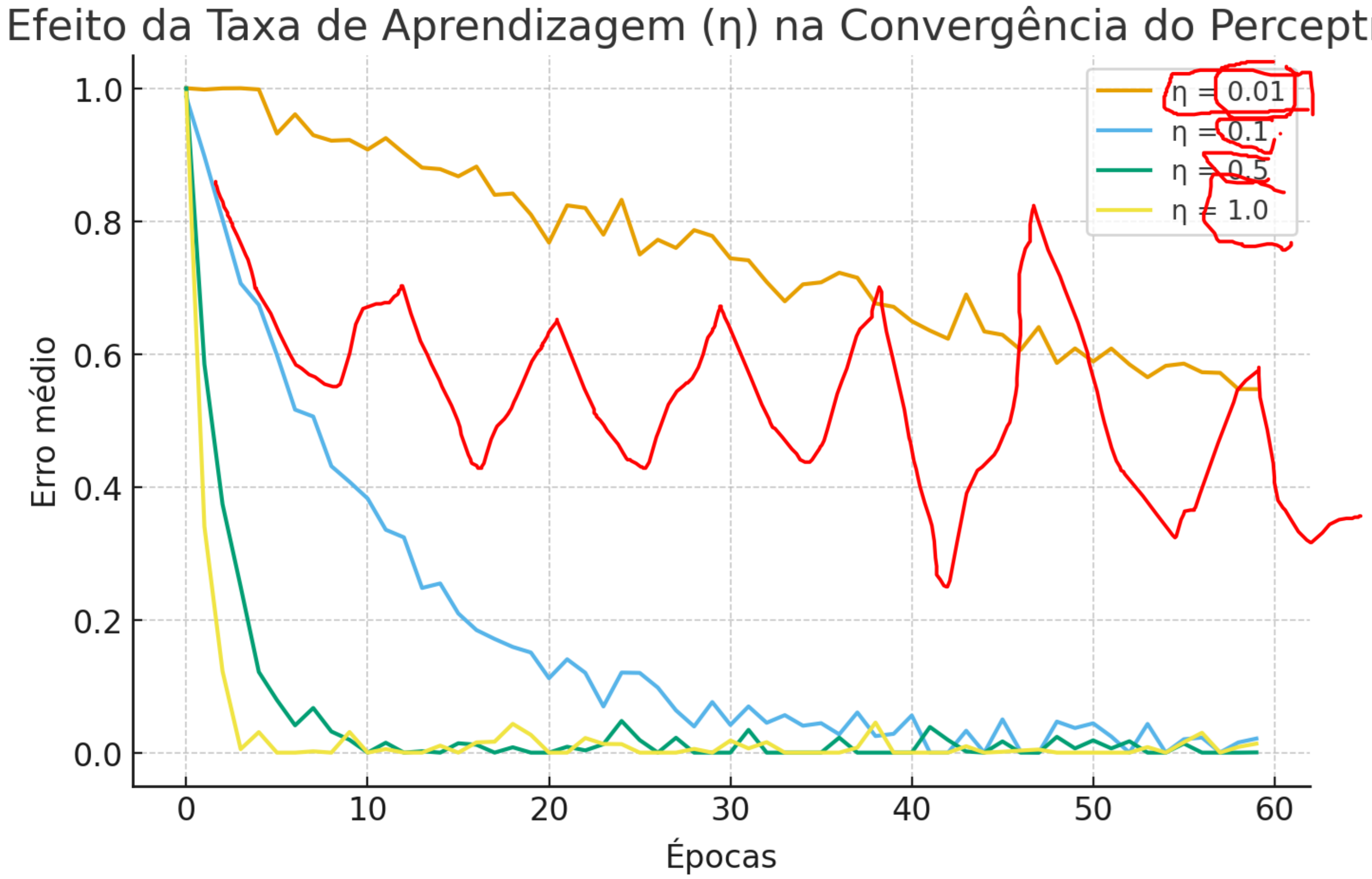
- Imagine que o Perceptron está “caminhando” em busca de uma solução (os pesos corretos).
- A taxa de aprendizagem define o tamanho do passo que ele dá em cada correção de erro:
  - Se η é grande, o Perceptron dá passos largos, **aprendendo mais rápido**, mas pode oscilar e nunca convergir.
  - Se η é pequena, os passos são curtos, **o aprendizado é mais estável**, mas pode demorar muito para chegar à solução.

Taxa de Aprendizagem

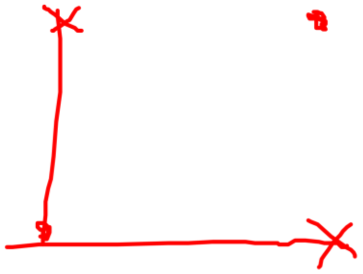
- Valores típicos em Perceptron:
- $\eta \in [0.01, 0.1]$
- Deve ser ajustada conforme:
  - o tamanho do erro,
  - a escala dos dados,
  - e a estabilidade desejada.
- Em redes neurais modernas, pode-se reduzir automaticamente η com o tempo (learning rate decay).

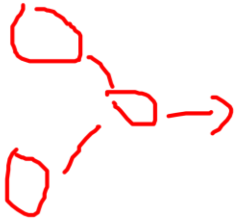


Taxa de Aprendizagem



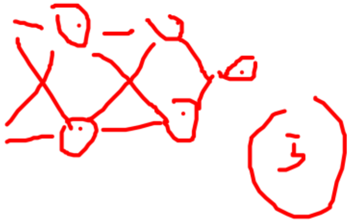
Limitações





- **Apenas separação linear:** Não aprende XOR ou padrões não lineares
- **Função degrau não diferenciável:** Não permite backpropagation
- **Não fornece probabilidades:** Classificação rígida e sem confiança
- **Sensível à taxa de aprendizado:** Pode divergir
- **Binário:** Não resolve problemas multiclasse facilmente
- **Sensível a ruído:** Oscila ou não converge

Perceptron é o átomo das redes neurais



- O Perceptron não é mais a rede, mas é a ideia que vive dentro de todos elas.

Conceito	**Introduzido no Perceptron**	**Mantido nas redes modernas**
Pesos w_i	Sim	Sim (milhões ou bilhões hoje)
Entrada com bias	Sim	Sim (às vezes substituído por normalização)
Combinação linear $u = \sum w_i\, x_i + b$	Sim	Sim (em toda camada linear ou convolucional)
Função de ativação $f(u)$	Sim (degrau)	Sim (sigmoid, tanh, ReLU, GELU, etc.)
Aprendizado supervisionado	Sim	Sim (com variantes modernas e gradientes)

Referências

- Silva, Ivan Nunes da; Spatti, Danilo Hernani. **Redes Neurais Artificiais para Engenharia e Ciências Aplicadas** – Curso Prático. 2ª edição. São Paulo: Editora Artliber, 2016. ISBN 978-85-88098-87-9.