

Introdução ao Deep Learning – Redes Neurais, Perceptron e MLP

Priscila Maia

priscila.maia@discente.ufg.br

Marcos Paulo Caetano

marcospaulo2@discente.ufg.br

Carlos Eduardo Rocha

carlos.rocha@discente.ufg.br

2024

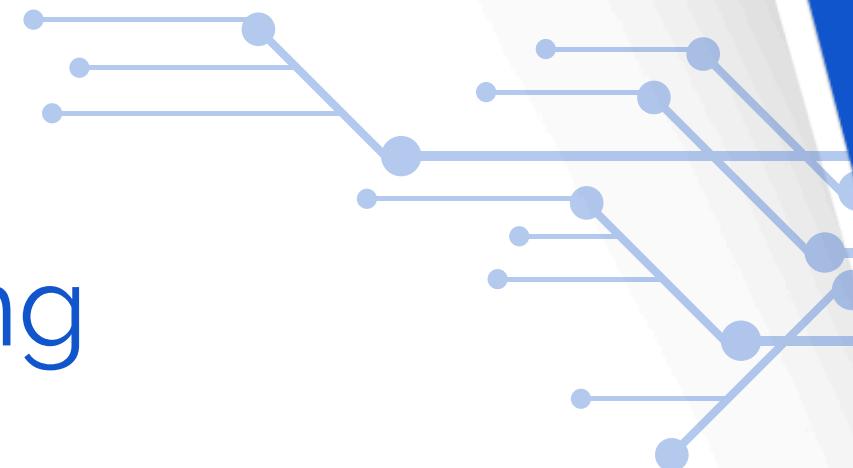
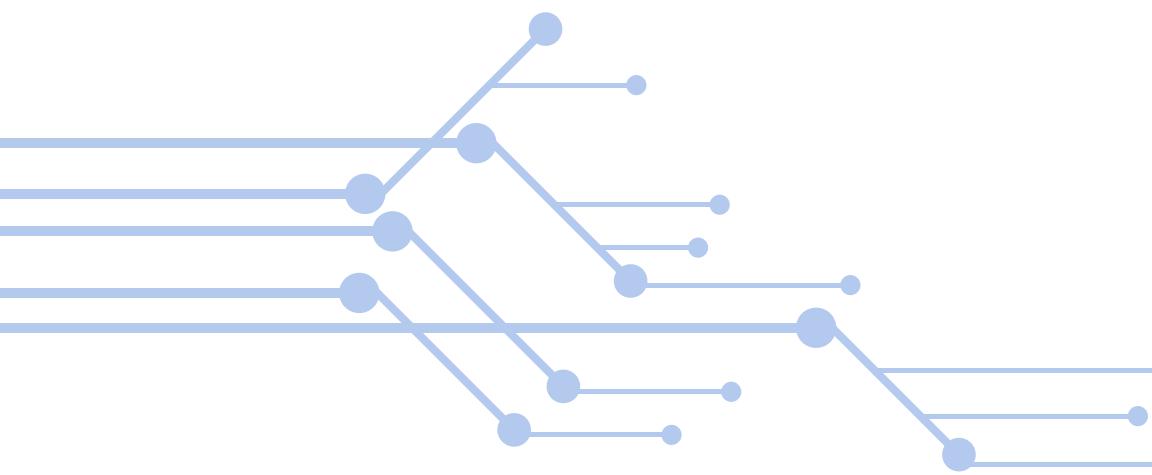
INF

INSTITUTO DE
INFORMÁTICA

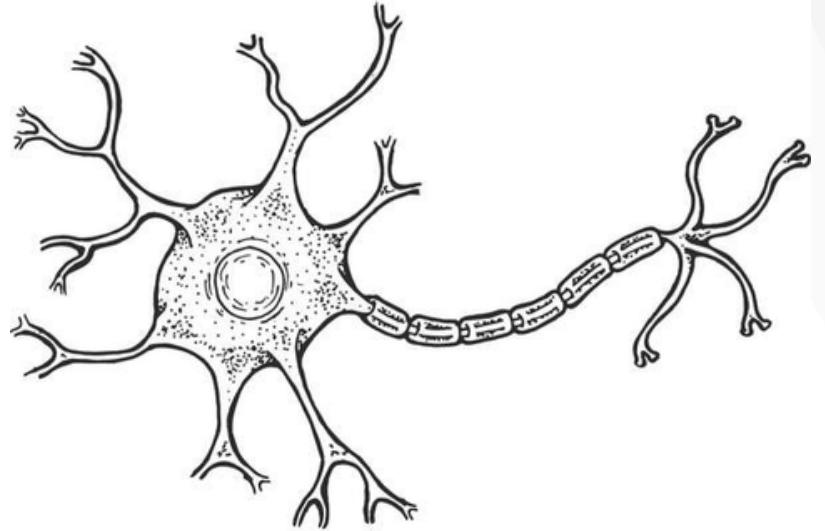


Sumário

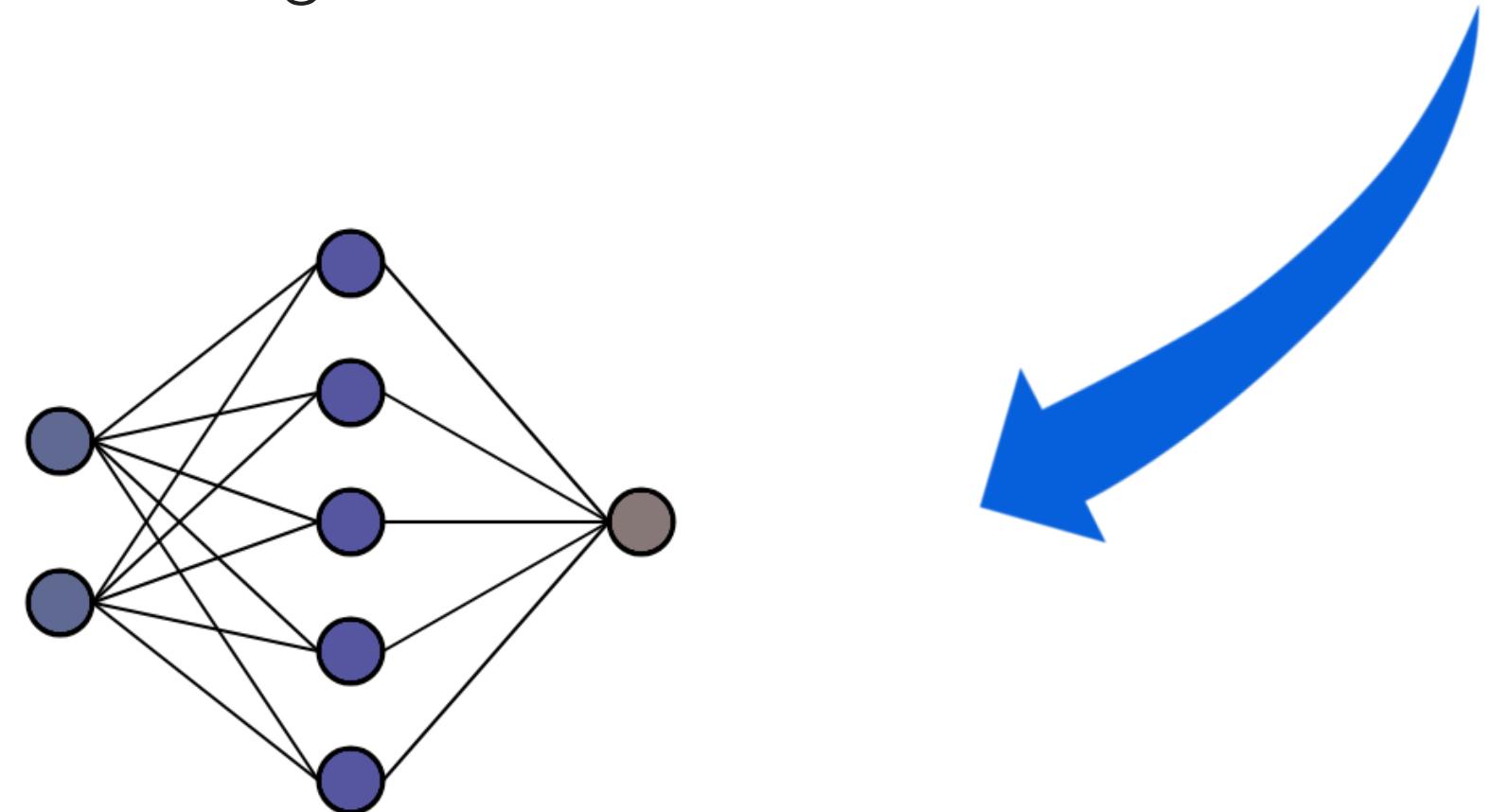
1. Introdução ao Deep Learning
2. Perceptron
3. Multilayer Perceptron (MLP)
4. Conceitos-chave
5. Projeto prático



Introdução ao Deep Learning

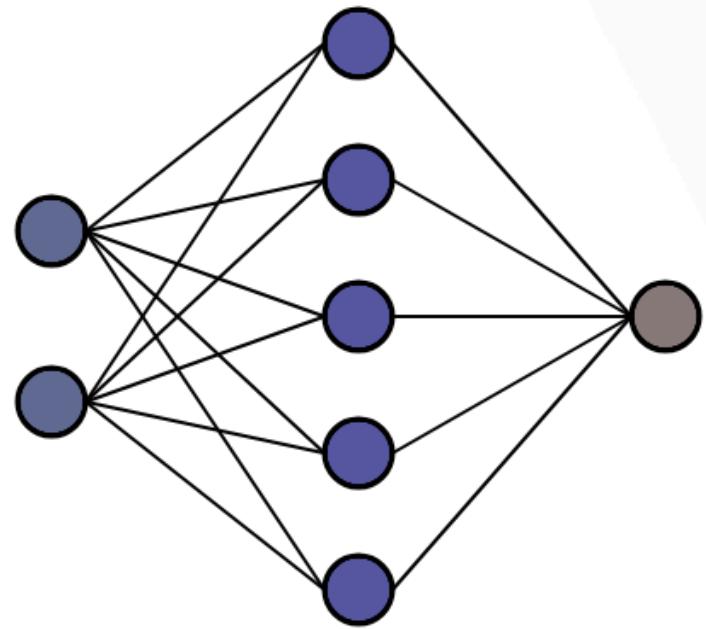


É uma classe de algoritmos de aprendizado de máquina que utiliza redes neurais artificiais para **modelar e entender padrões complexos** em grandes volumes de dados.



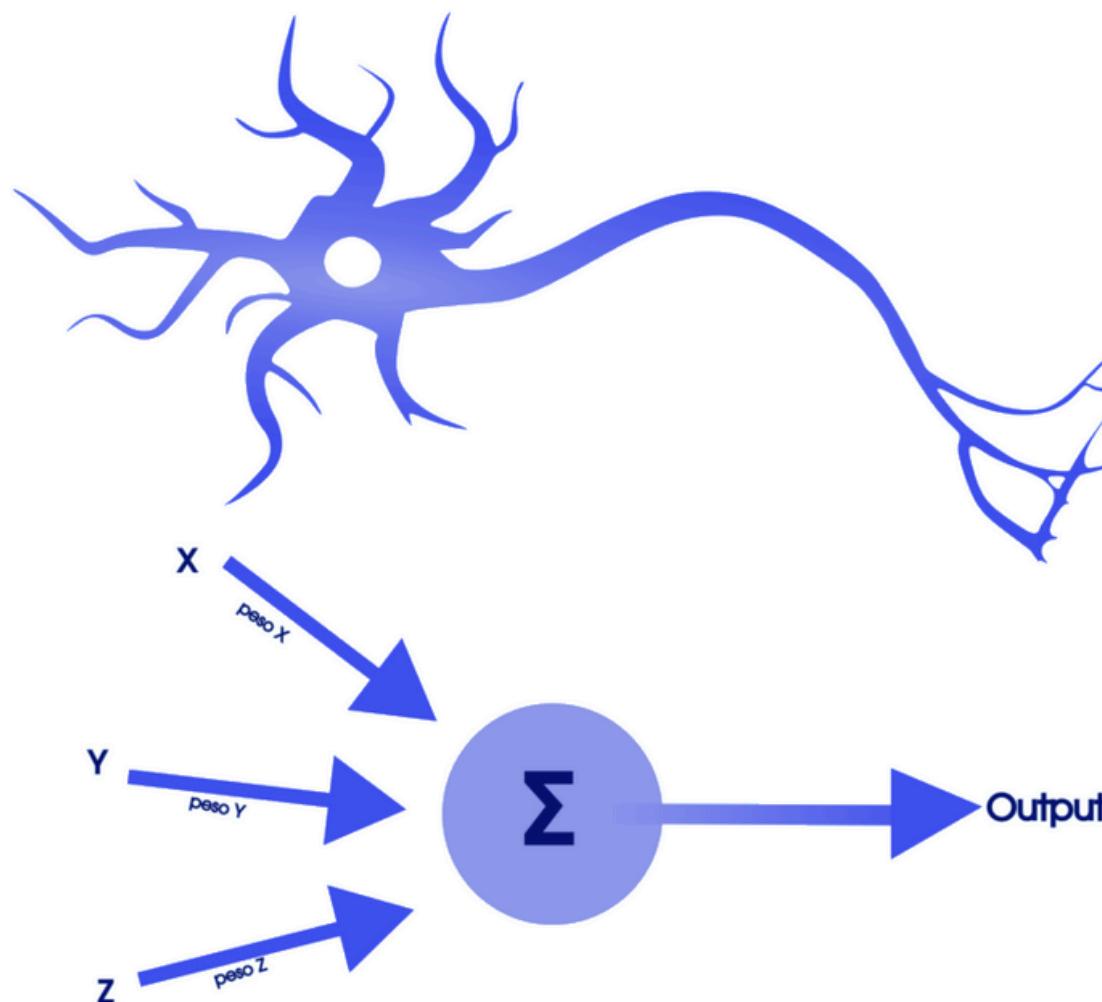
Mas o que são as redes neurais artificiais?

São nós interconectados, que chamamos de **neurônios artificiais** organizados em camadas.



Redes neurais artificiais:

- Nosso cérebro consegue fazer várias funções com mesmo "hardware"
 - E se tentassem imitar sua estrutura?
- Nosso cérebro é feito de neurônios
 - Dendrito -> Corpo -> Axônio
 - Camada de Entrada -> Camadas Ocultas -> Camada de Saída



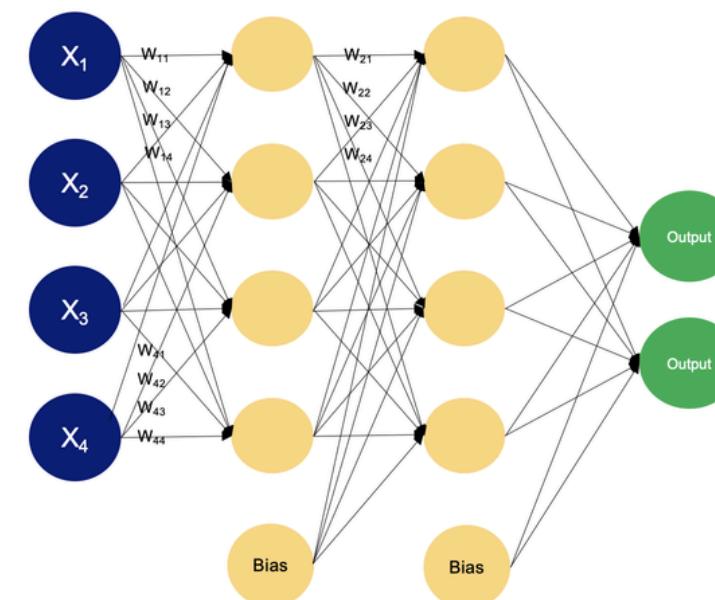
- **Camada de Entrada:** Onde os dados brutos são introduzidos na rede.
- **Camadas Ocultas:** Onde ocorrem os cálculos, ou seja, o processamento computacional, extraíndo características e padrões dos dados.
- **Camada de Saída:** Onde a rede fornece sua predição ou classificação com base no processamento realizado nas camadas ocultas.

Tipos:

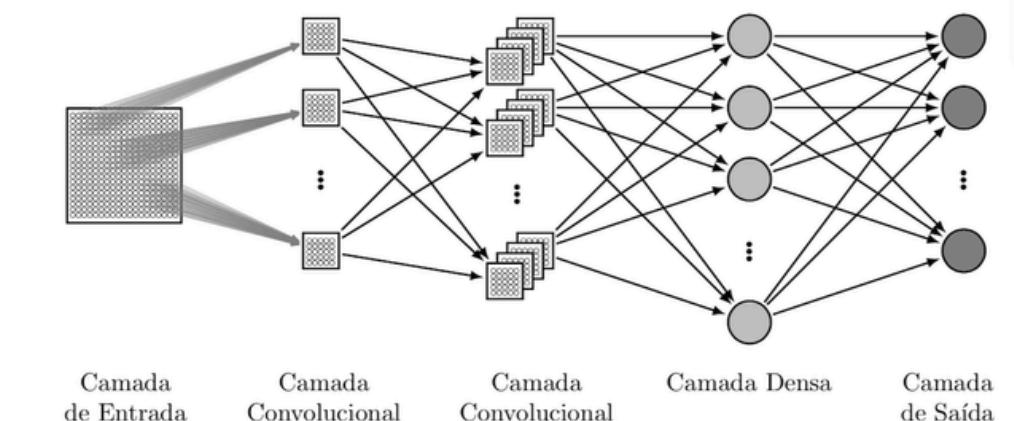
Há vários tipos de Redes Neurais, cada um projetado para tarefas e requisitos arquitetônicos específicos.

Redes Neurais Feedforward

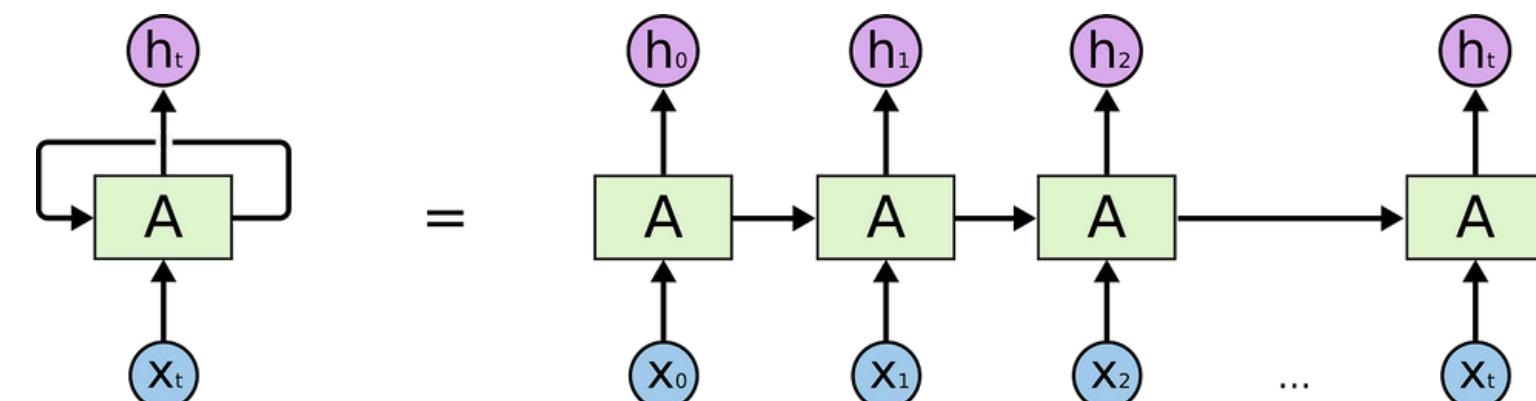
- Perceptron (neurônio)
- Perceptron Multicamadas (MLP)



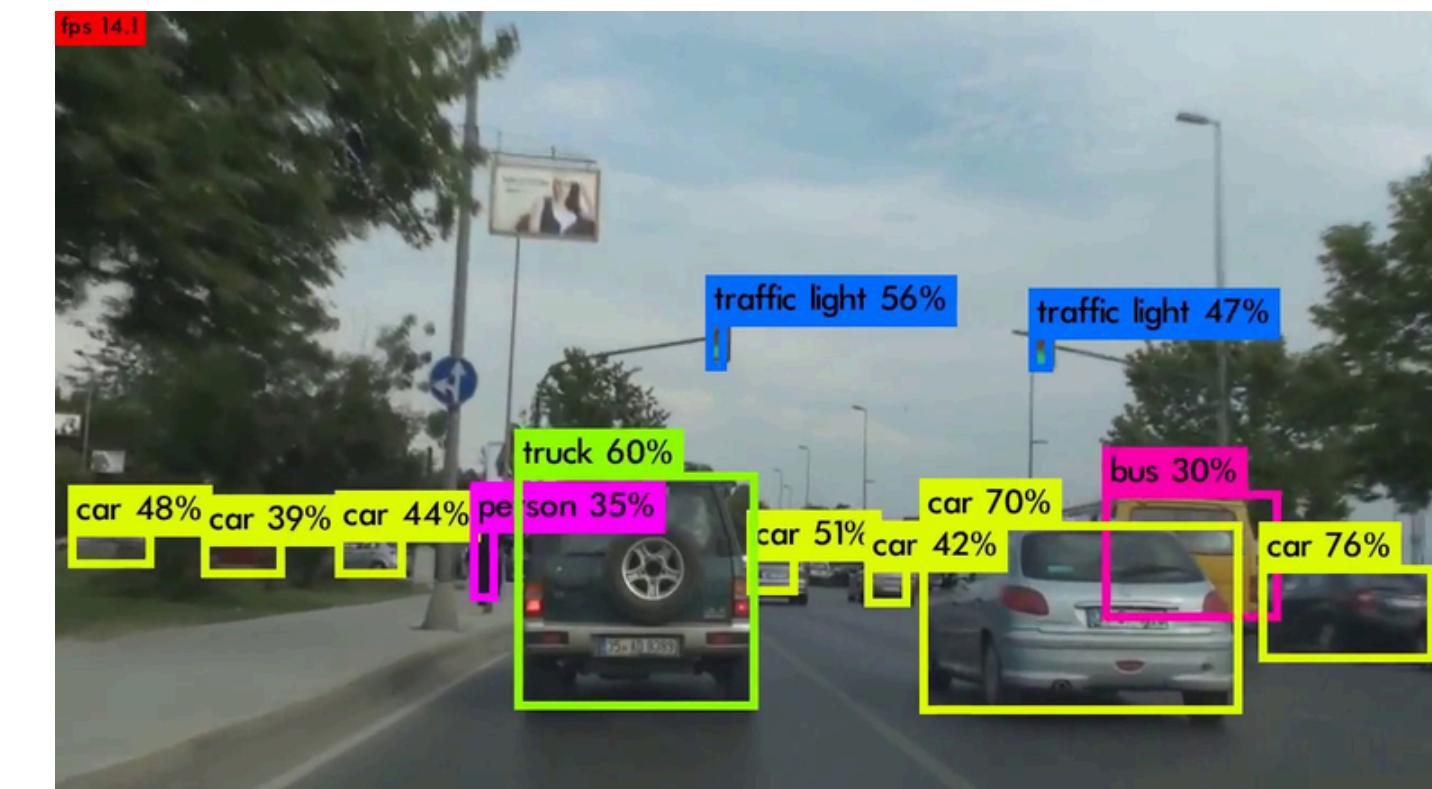
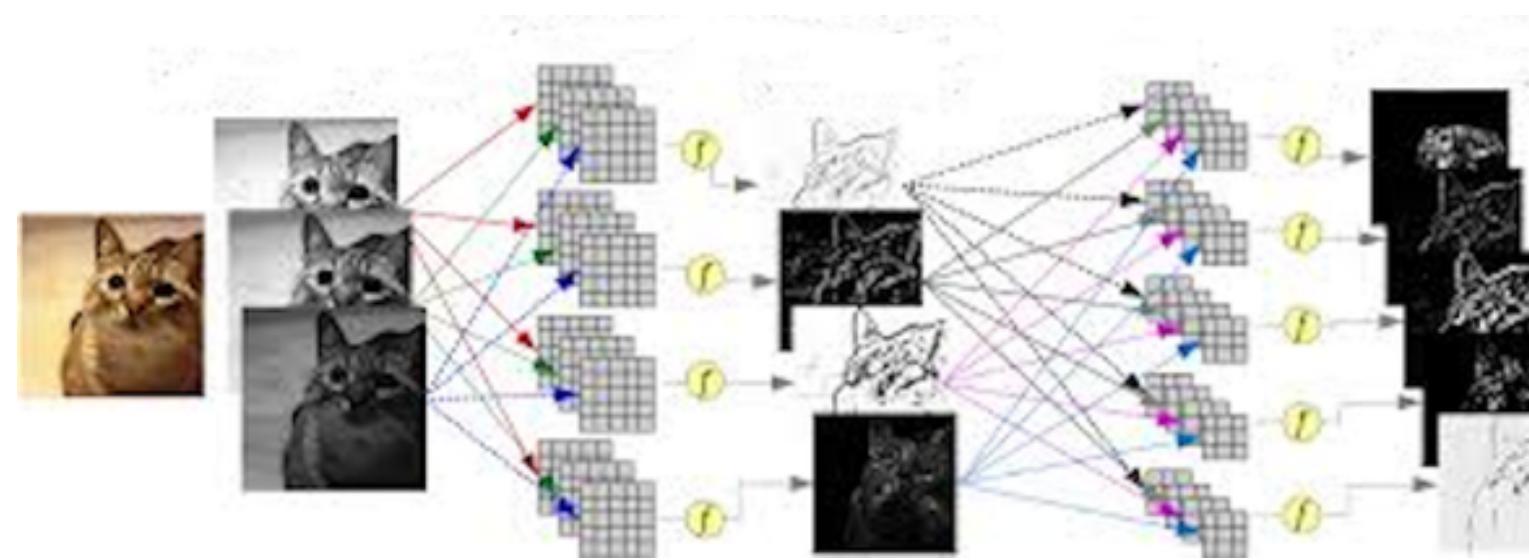
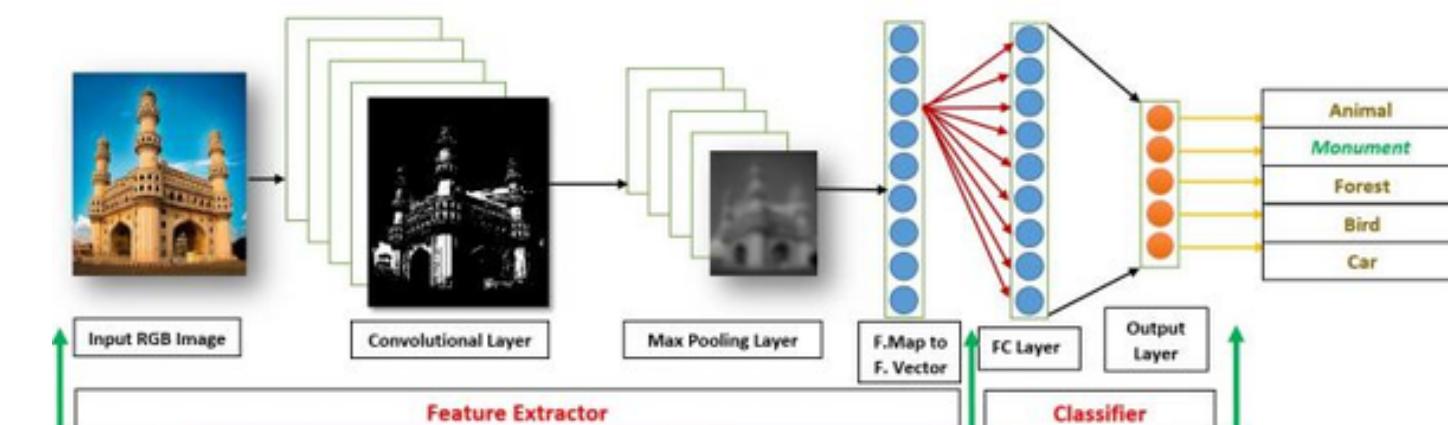
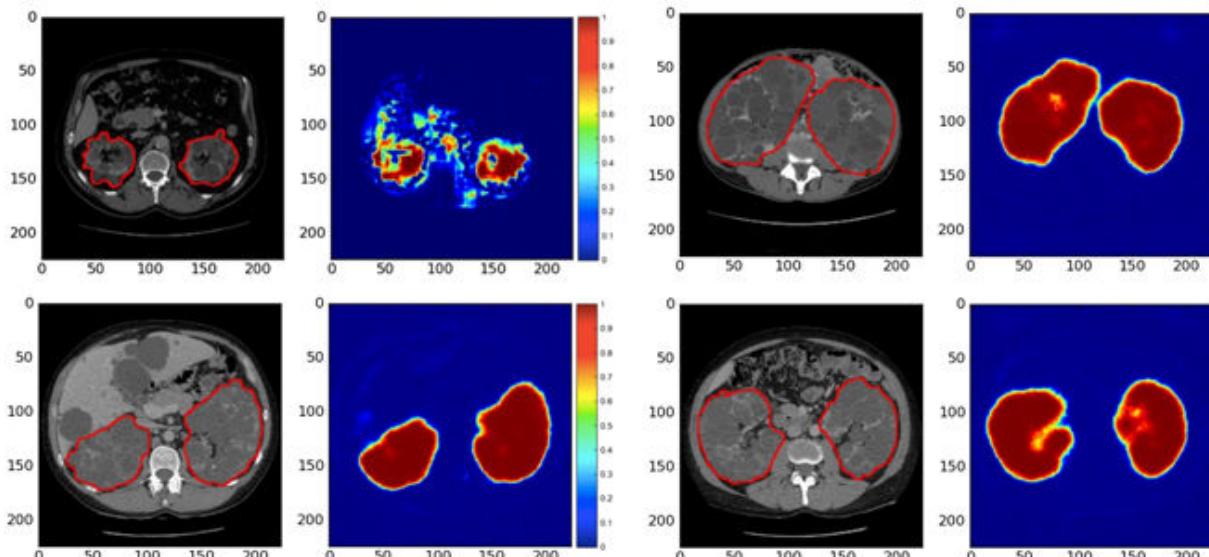
Redes Neurais Convolucionais (CNNs)



Redes Neurais Recorrentes (RNNs)



Importância no Contexto da Visão Computacional



Exemplos de aplicações reais

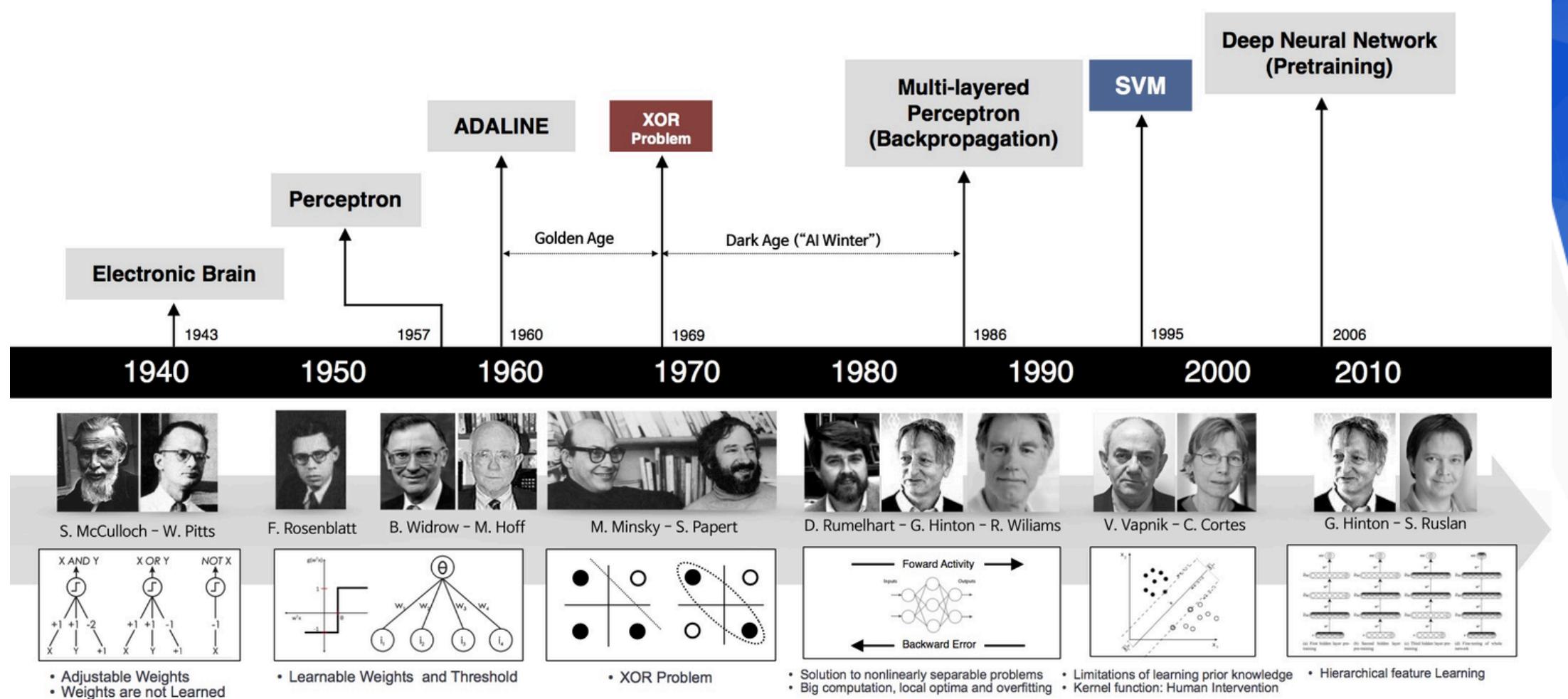
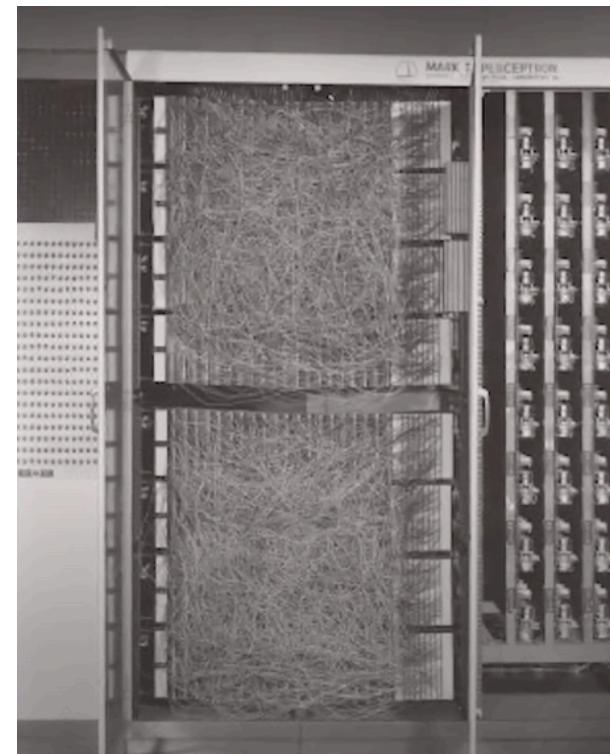
Conteúdo:

- Segurança e Vigilância:
 - Reconhecimento facial em câmeras de segurança.
 - Detecção de atividades suspeitas em vídeos.
- Saúde:
 - Diagnóstico por imagens médicas (como detecção de câncer em mamografias).
 - Identificação de doenças em raios-X e tomografias.
- Transporte:
 - Carros autônomos, reconhecendo objetos e placas de trânsito em tempo real.
 - Prevenção de acidentes com análise de comportamentos na estrada.
- Indústria:
 - Inspeção visual de qualidade em linhas de produção.
- Entretenimento:
 - Filtros de realidade aumentada em aplicativos de redes sociais.
 - Geração de imagens e vídeos realistas com redes adversariais (GANs).

Perceptron

História e origem

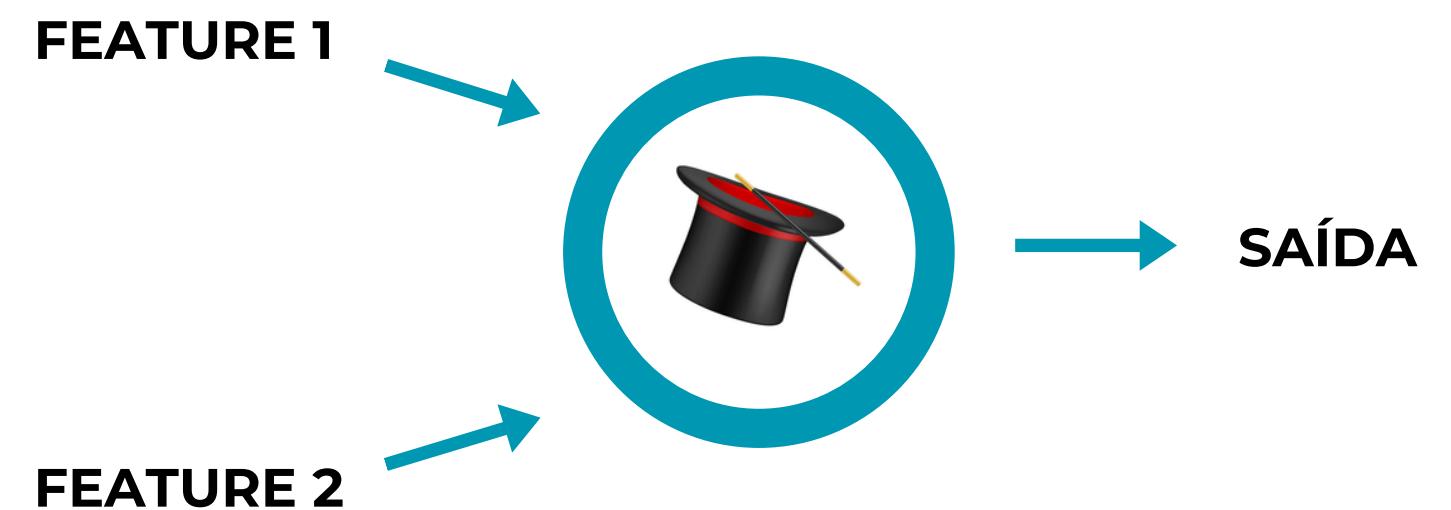
- Criação (1958): Desenvolvido por Frank Rosenblatt
- Objetivo: Inspirado no funcionamento dos neurônios biológicos, o modelo foi criado para simular a capacidade do cérebro de reconhecer padrões e aprender com eles.
- Primeira Implementação: Construído em hardware como o Mark I Perceptron, um computador analógico conectado a uma câmera, **capaz de identificar formas geométricas simples ou letras do alfabeto**.
- Impacto Inicial: Foi amplamente celebrado como um avanço revolucionário em IA, com manchetes proclamando que "as máquinas podem aprender sozinhas".



Perceptron

Consiste em uma única camada, que é a camada de entrada, com vários neurônios com seus próprios pesos; **não há camadas ocultas**. O algoritmo perceptron aprende os pesos para os sinais de entrada a fim de traçar um limite de decisão linear.

- Entradas: São os dados que alimentam o modelo.
- Pesos: Cada entrada tem um peso associado, representado por W_1 e W_2 . Esses pesos ajustam a importância de cada entrada no cálculo.
- Processamento: As entradas são multiplicadas pelos pesos e somadas:
- Função de Ativação: O resultado passa por uma função de ativação, que decide a saída final.
- Saída: Resultado



Perceptron

Determinar se um **programador está apto ou não apto para ser alocado a um projeto de programação**, com base em dois critérios principais:

FEATURE 1- Experiência com a linguagem do projeto:

X = 1 O programador tem experiência com a linguagem do projeto.

X = 0 O programador não tem experiência com a linguagem do projeto.

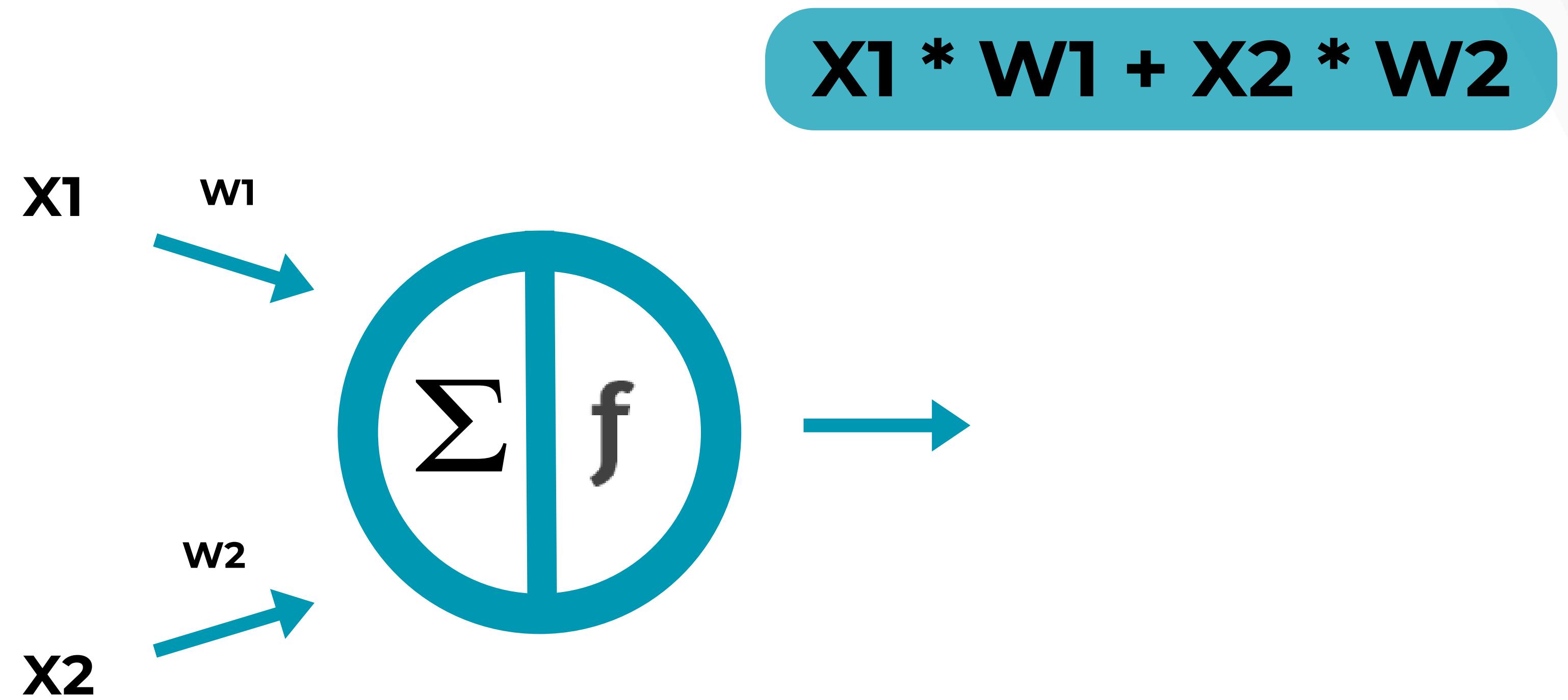
FEATURE 2 - Experiência prévia com outros projetos

X = 1 O programador tem experiência com projetos anteriores.

X = 0 O programador não tem experiência com projetos anteriores.

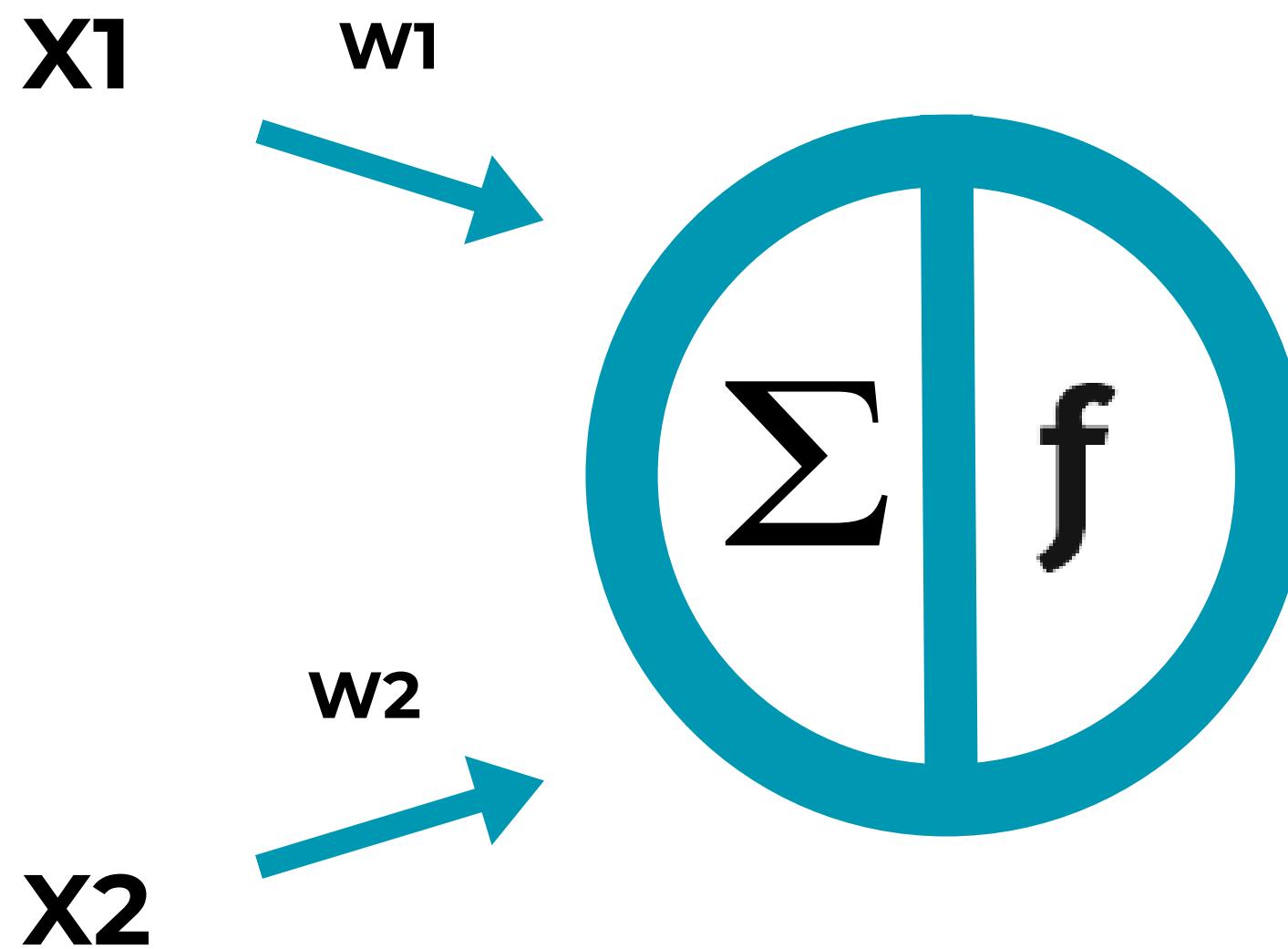


Perceptron



Perceptron

$$X_1 * W_1 + X_2 * W_2$$



A nossa função nesse exemplo é a função degrau (Step Function):

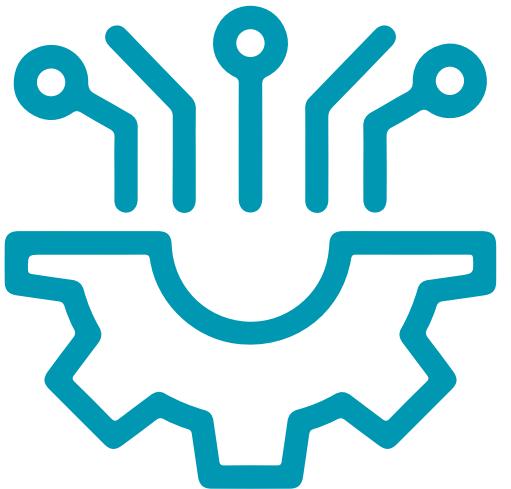
se > 0 então a saída é 1

se ≤ 0 então a saída é 0

Perceptron

Receita do perceptron:

- 1 - iniciar com um conjunto de dados supervisionado
- 2 - feed forward (perceptron faz as previsões)
- 3 - calcular os erros (loss) e atualizar os pesos



Perceptron

1- Dados supervisionados

Registro	X1	X2	Target
A	1	0	1
B	0	0	0
C	0	1	1
D	1	1	1

conceitos importantes

época:

número de vezes que o algoritmo processa o dataset inteiro

tamanho do batch:

número de exemplos de treinamento em um único batch antes do modelo atualizar os pesos

\hat{Y}

Perceptron

1- Dados supervisionados

Registro	X1	X2	Target
A	1	0	1
B	0	0	0
C	0	1	1
D	1	1	1

conceitos importantes

época:

número de vezes que o algoritmo processa o dataset inteiro

tamanho do batch:

número de exemplos de treinamento em um único batch antes do modelo atualizar os pesos

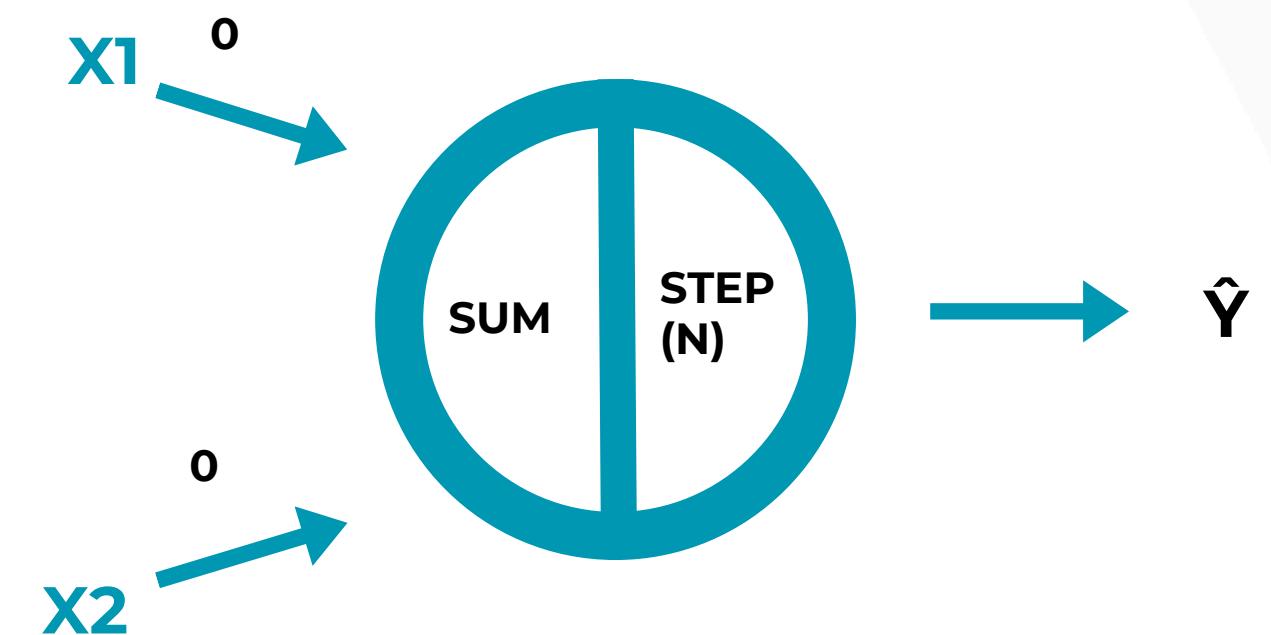
\hat{Y}



Perceptron

2- Feed Forward

Registro	X1	X2	Target
A	1	0	1
B	0	0	0
C	0	1	1
D	1	1	1



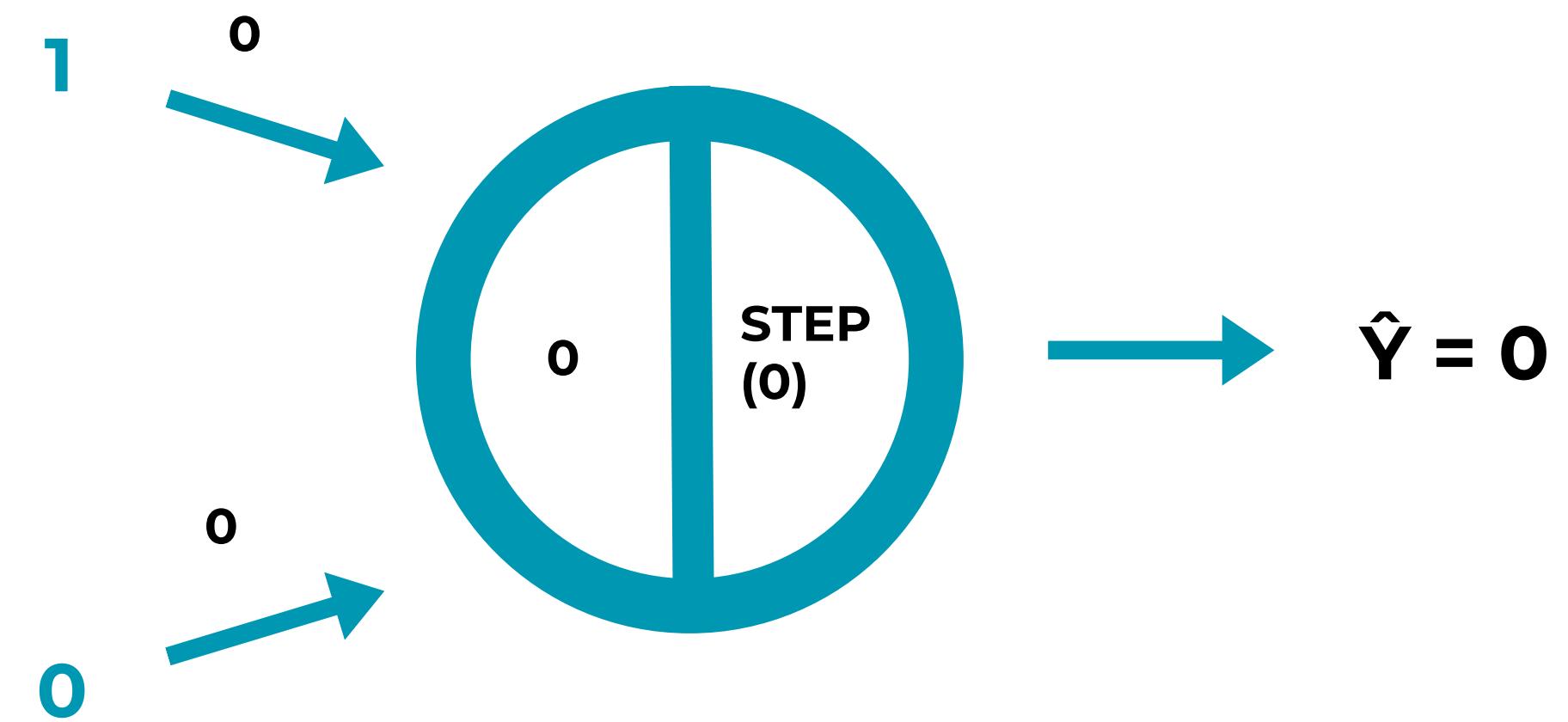
$$\hat{Y} = g(W_1 * X_1 + W_2 * X_2)$$

Perceptron

2- Feed Forward

ÉPOCA 1
AMOSTRA 1

Registro	X1	X2	Target
A	1	0	1



Perceptron

3- Calcular os erros (loss) e atualizar os pesos

ÉPOCA 1 AMOSTRA 1

erro = target - predição

$$\text{erro} = 1 - 0 \rightarrow \text{erro} = 1$$

novo peso = peso + learning rate * erro * input

$$w1 = 0 + 0.5 * 1 * 1 \rightarrow w1 = 0.5$$

$$w2 = 0 + 0.5 * 1 * 0 \rightarrow w2 = 0$$

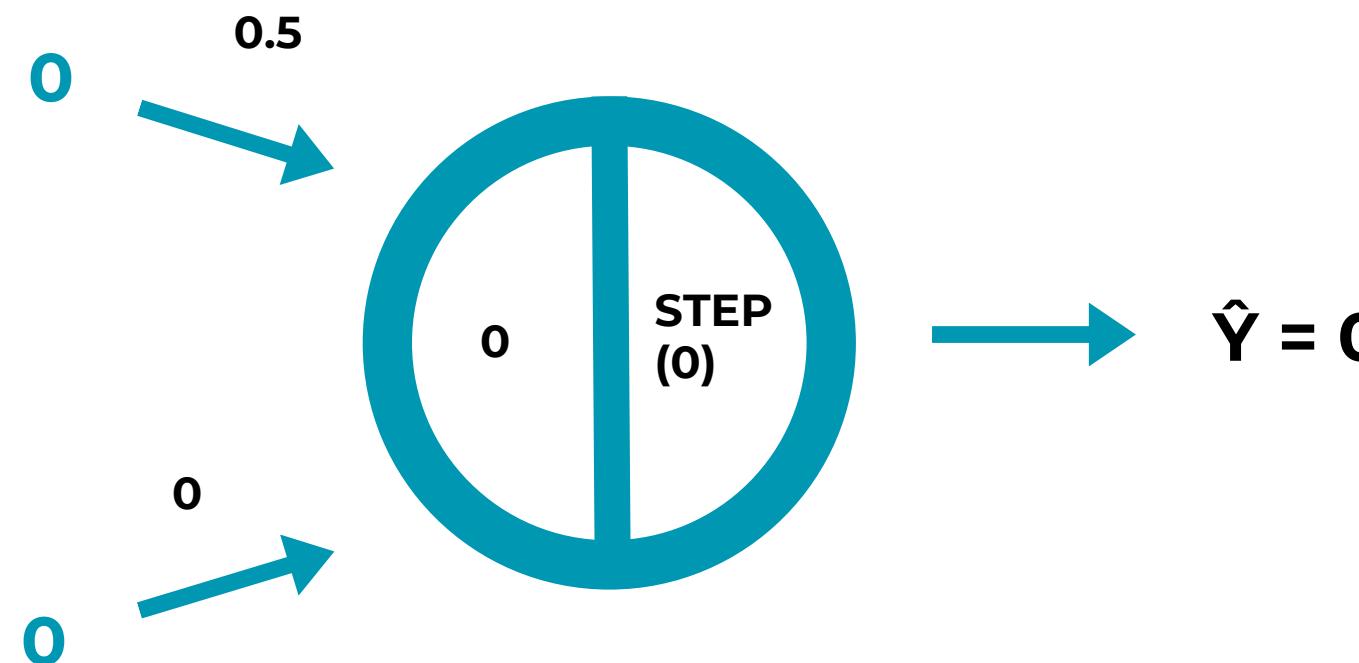
$$\Delta w_i = \eta(y_i - \hat{y}_i)x_i$$
$$w_i = w_i + \Delta w_i$$

Perceptron

2- Feed Forward

ÉPOCA 1
AMOSTRA 2

Registro	X1	X2	Target
B	0	0	0



3- Calcular os erros (loss) e atualizar os pesos

ÉPOCA 1
AMOSTRA 2

$$\text{erro} = \text{target} - \text{predição}$$

$$\text{erro} = 0 - 0 \rightarrow \text{erro} = 0$$

$$\text{novo peso} = \text{peso} + \text{learning rate} * \text{erro} * \text{input}$$

$$w_1 = 0.5 + 0.5 * 0 * 1 \rightarrow w_1 = 0.5$$

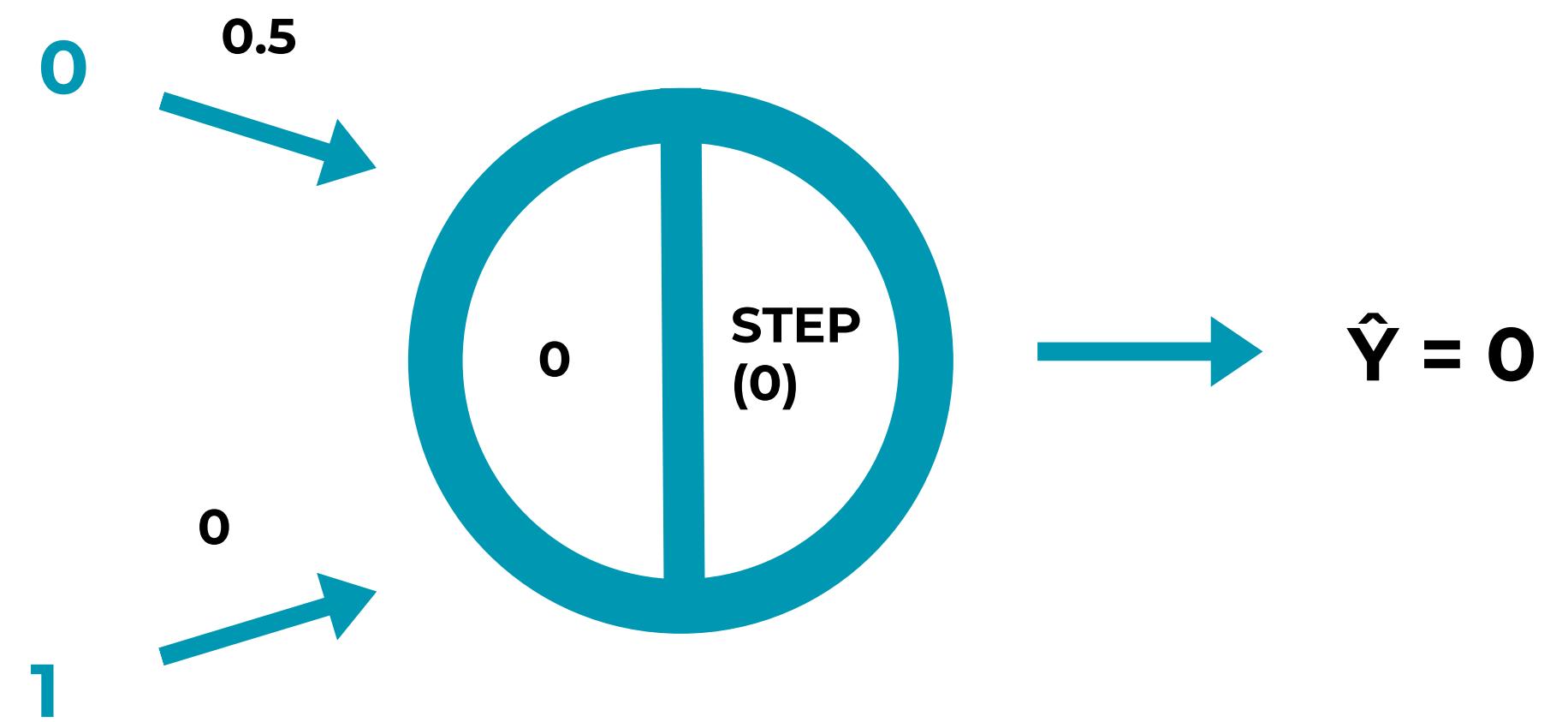
$$w_2 = 0 + 0.5 * 0 * 0 \rightarrow w_2 = 0$$

Perceptron

2- Feed Forward

**ÉPOCA 1
AMOSTRA 3**

Registro	X1	X2	Target
C	0	1	1



Perceptron

3- Calcular os erros (loss) e atualizar os pesos

**ÉPOCA 1
AMOSTRA 3**

erro = target - predição

$$\text{erro} = 1 - 0 \rightarrow \text{erro} = 1$$

novo peso = peso + learning rate * erro * input

$$w1 = 0.5 + 0.5 * 1 * 0 \rightarrow w1 = 0.5$$

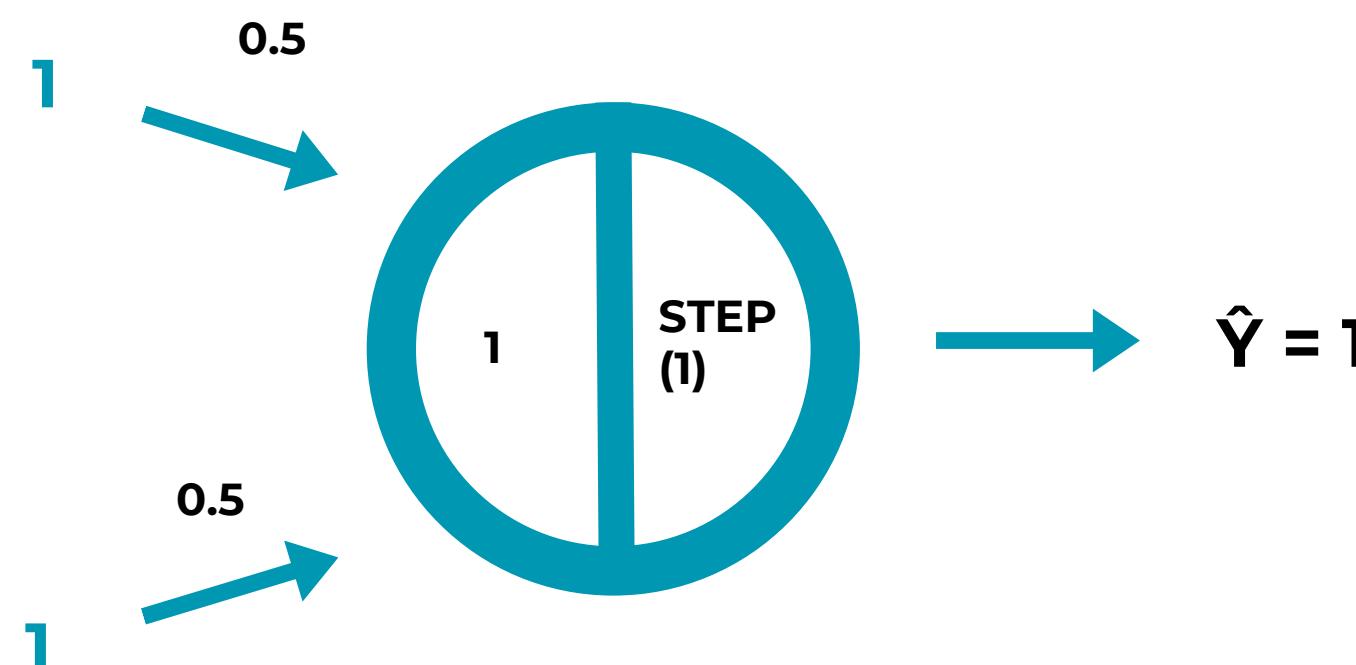
$$w2 = 0 + 0.5 * 1 * 1 \rightarrow w2 = 0.5$$

Perceptron

2- Feed Forward

ÉPOCA 1
AMOSTRA 4

Registro	X1	X2	Target
D	1	1	1



3- Calcular os erros (loss) e atualizar os pesos

ÉPOCA 1
AMOSTRA 4

$$\text{erro} = \text{target} - \text{predição}$$

$$\text{erro} = 1 - 1 \rightarrow \text{erro} = 0$$

$$\text{novo peso} = \text{peso} + \text{learning rate} * \text{erro} * \text{input}$$

$$w1 = 0.5 + 0.5 * 0 * 0 \rightarrow w1 = 0.5$$

$$w2 = 0.5 + 0.5 * 0 * 1 \rightarrow w2 = 0.5$$

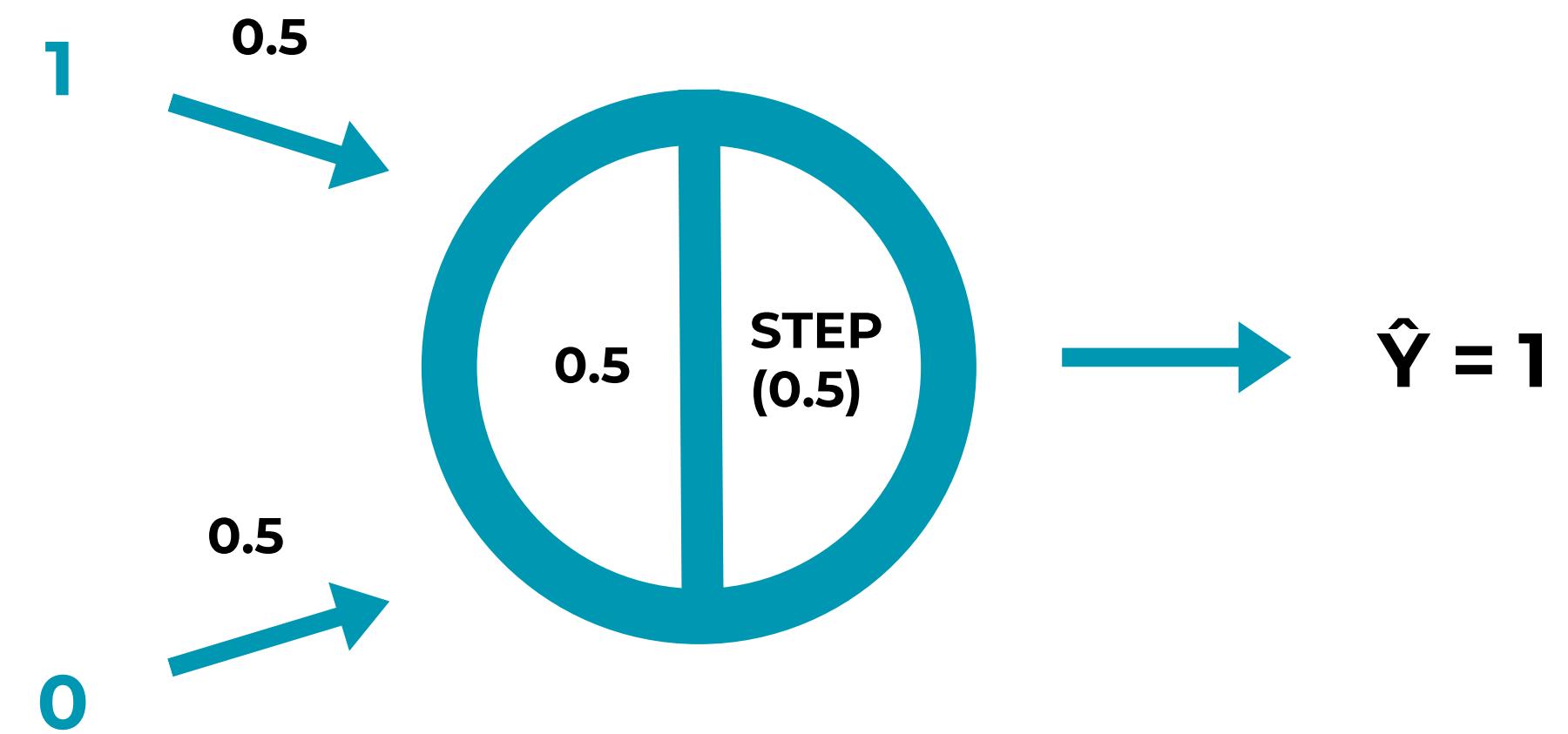
FIM DA ÉPOCA 1

Perceptron

2- Feed Forward

ÉPOCA 2
AMOSTRA 1

Registro	X1	X2	Target
A	1	0	1



Perceptron

2- Feed Forward

ÉPOCA 2
AMOSTRA 2

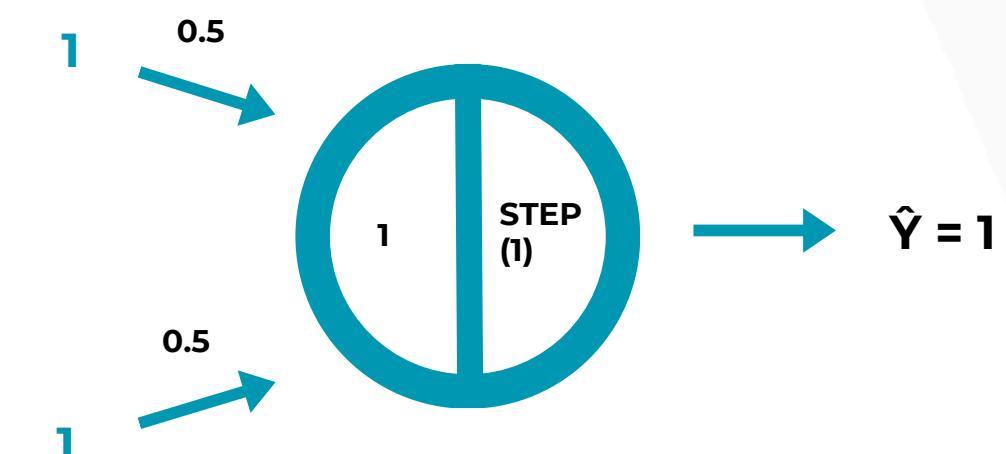
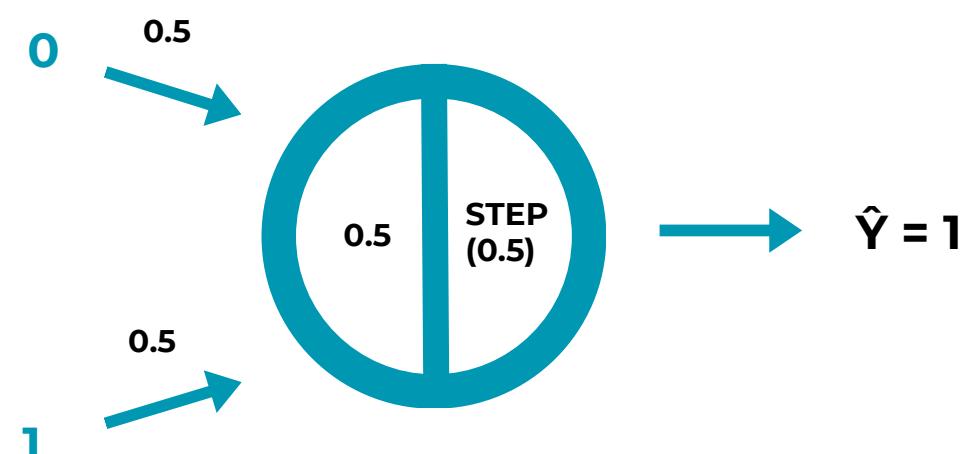
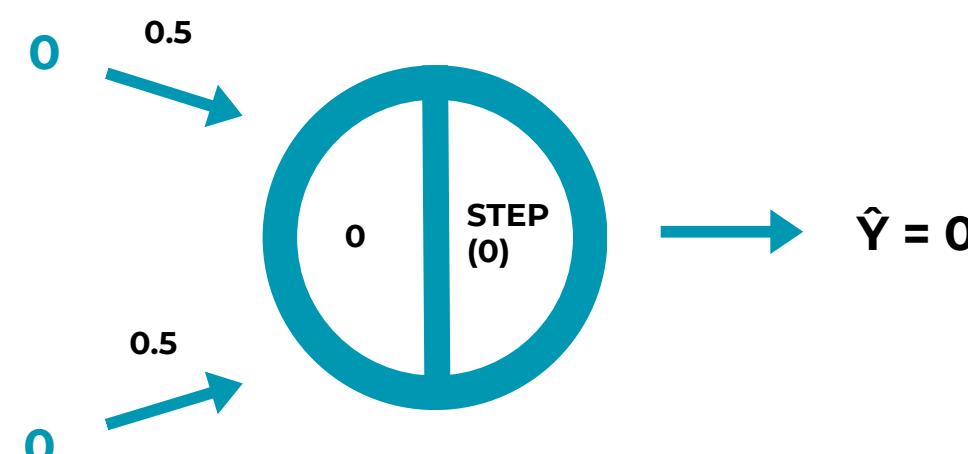
Registro	X1	X2	Target
B	0	0	0

ÉPOCA 2
AMOSTRA 3

Registro	X1	X2	Target
C	0	1	1

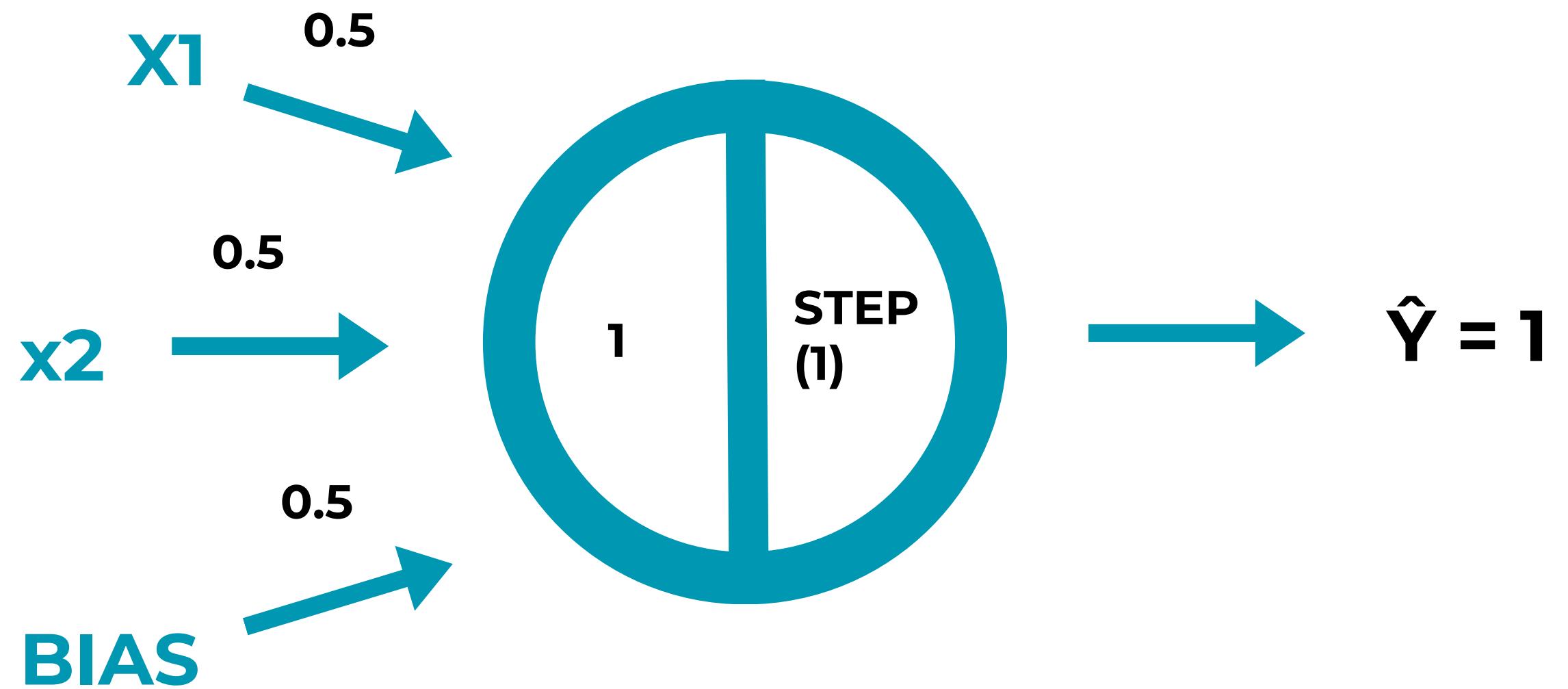
ÉPOCA 2
AMOSTRA 4

Registro	X1	X2	Target
D	1	1	1



FIM DA ÉPOCA 2

Bias

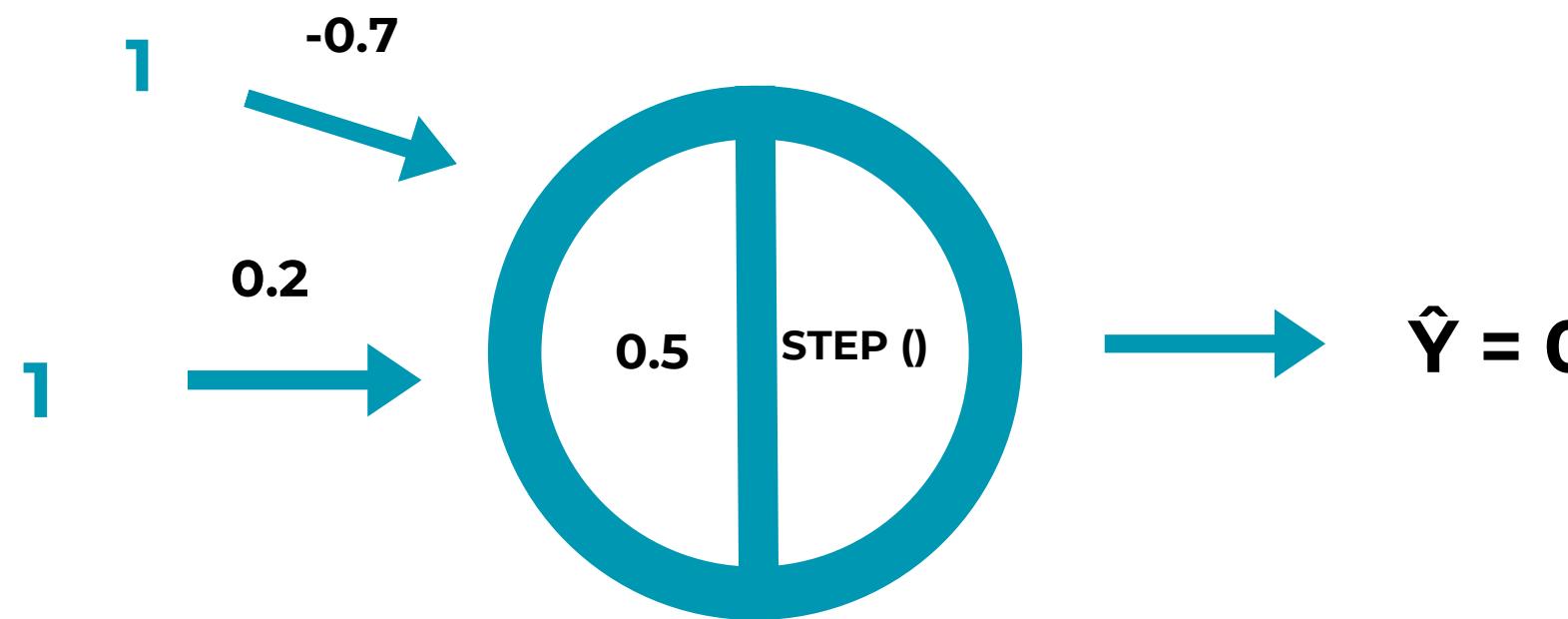


$$\hat{Y} = g(W_1 * X_1 + W_2 * X_2 + b)$$

Bias

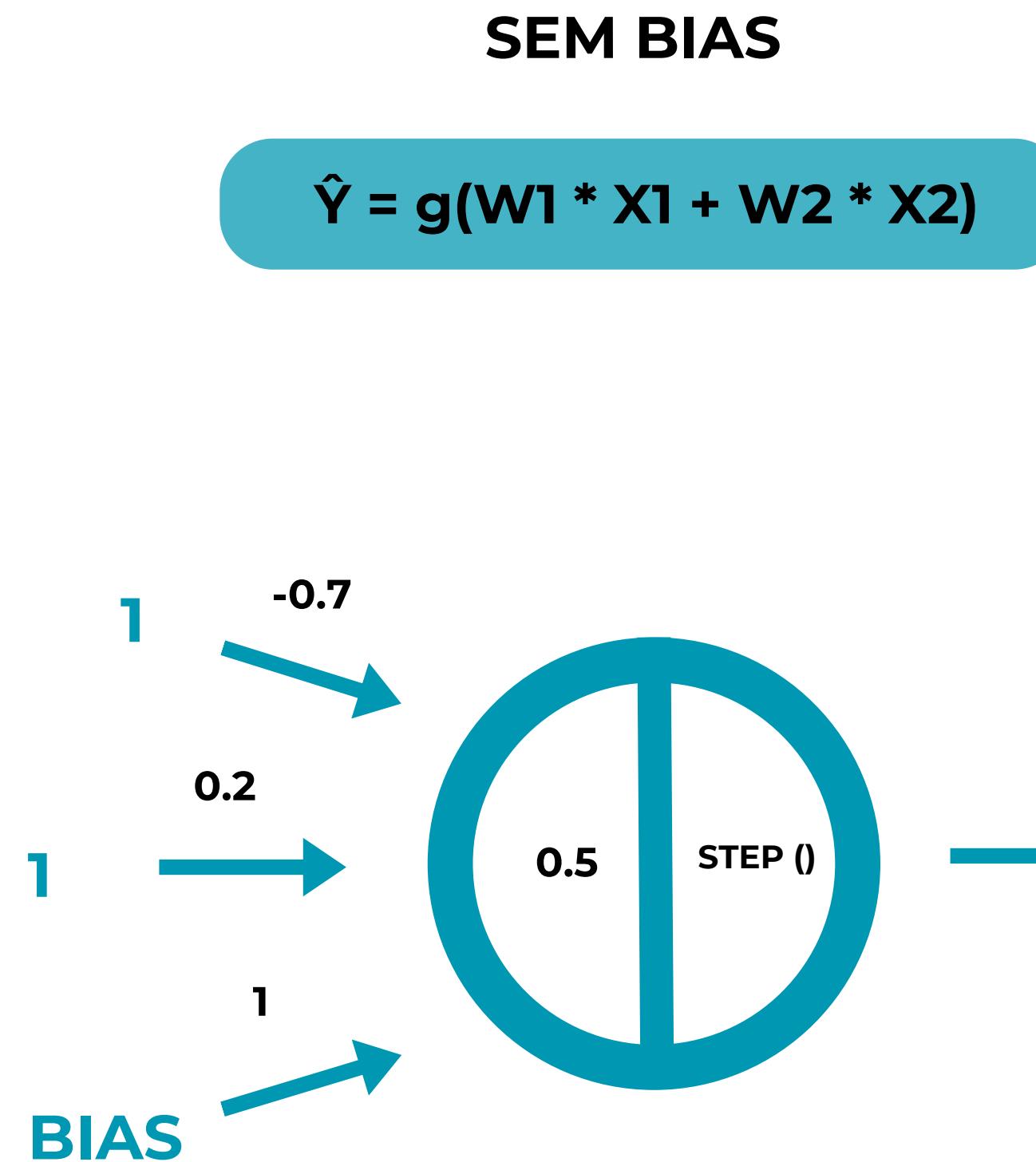
- cada neurônio pode ter bias
- pode ser estático ou aprendido (assim como os pesos)
- ajuda a determinar o limiar (threshold) de disparo
- resulta em mais flexibilidade para o modelo
- $b = b + (\eta \cdot E)$:
 - b : O bias novo, ou seja, o valor atualizado do bias após o ajuste.
 - b : O bias antigo (atual), ou seja, o valor do bias antes do ajuste.
 - η : O learning rate, que define o tamanho do passo dado no ajuste (mesmo valor usado para os pesos).
 - E : O erro calculado para a saída do modelo (valor desejado - valor predito pela rede)

Bias



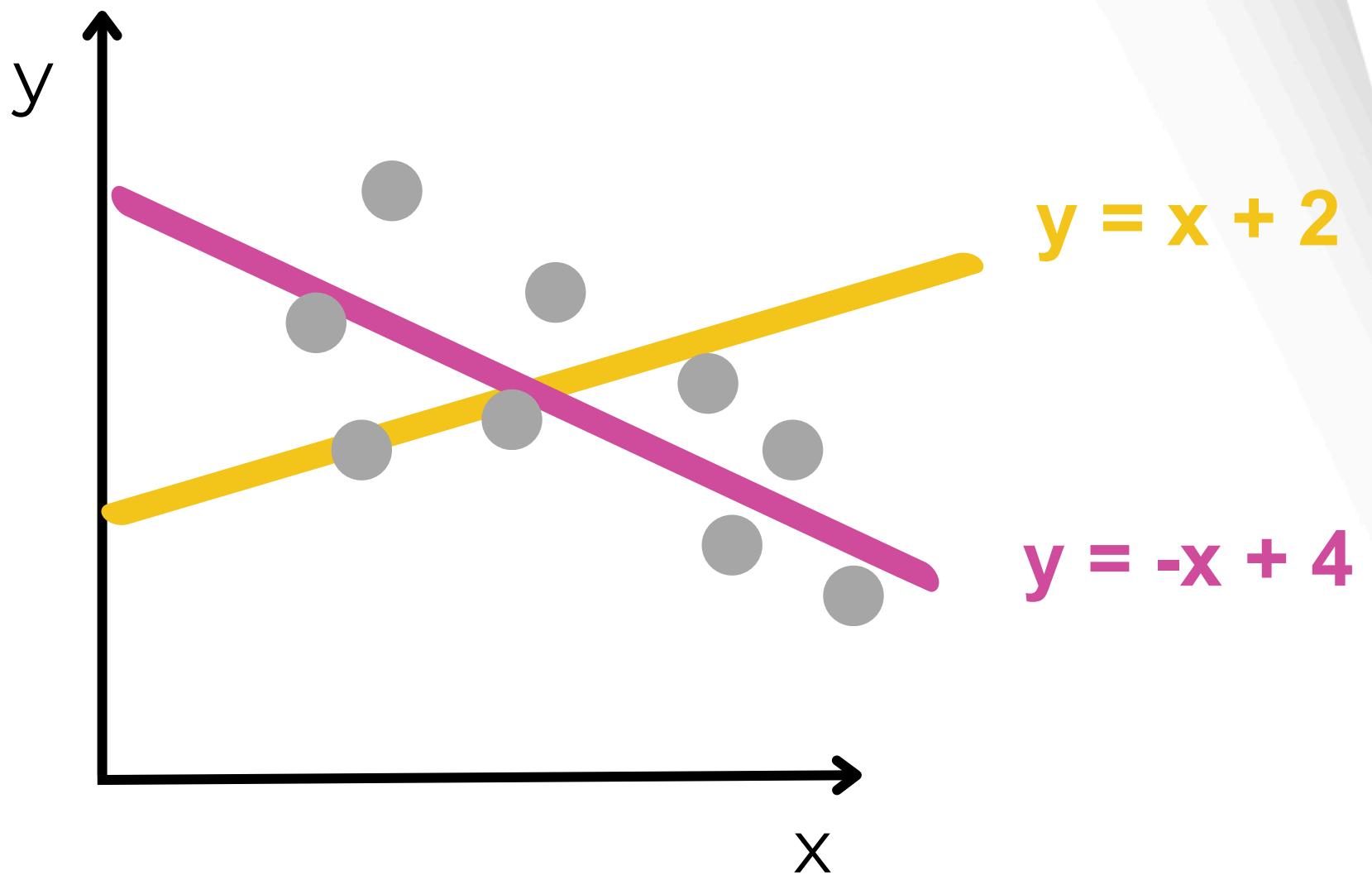
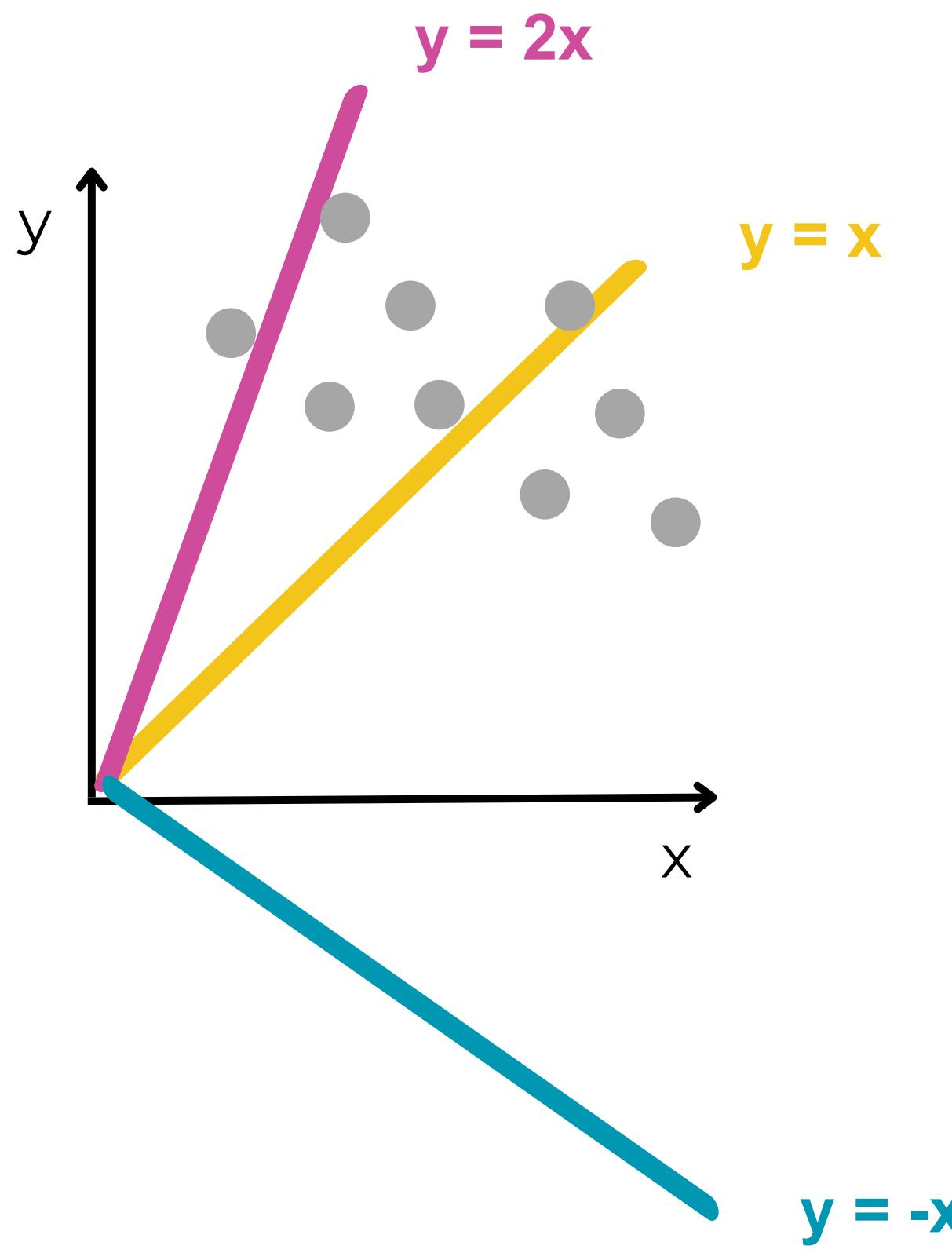
COM BIAS

$$\hat{Y} = g(W_1 * X_1 + W_2 * X_2 + b)$$



SEM BIAS

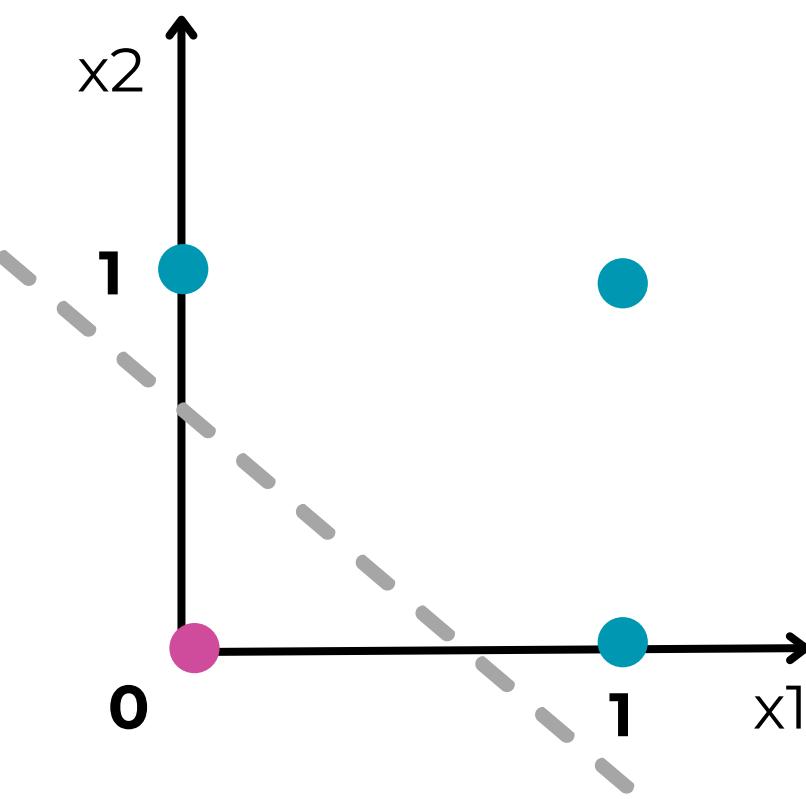
$$\hat{Y} = g(W_1 * X_1 + W_2 * X_2)$$



Limitações do Perceptron

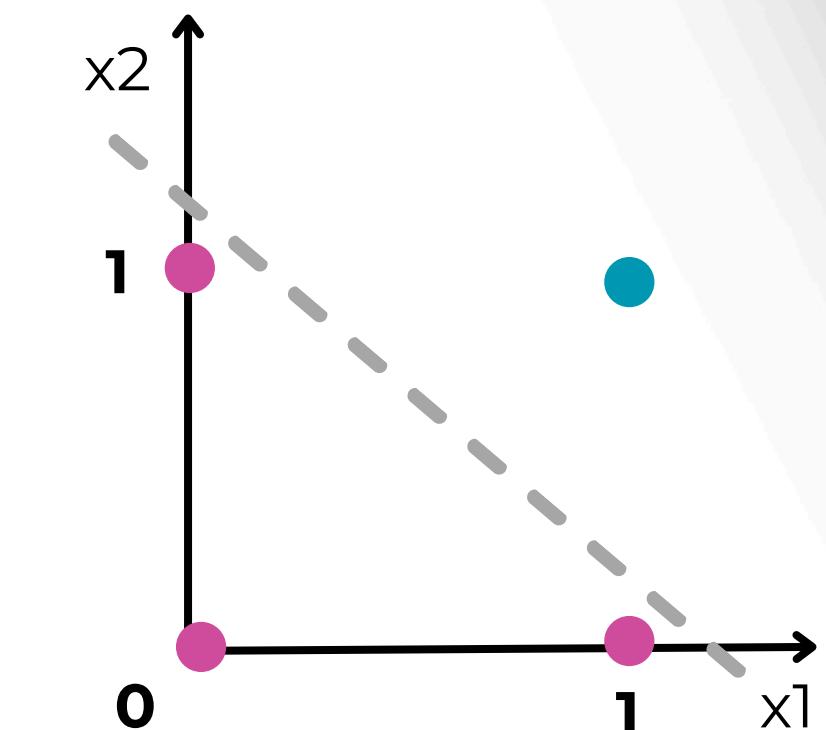
OR

X1	X2	0
1	0	1
0	0	0
0	1	1
1	1	1



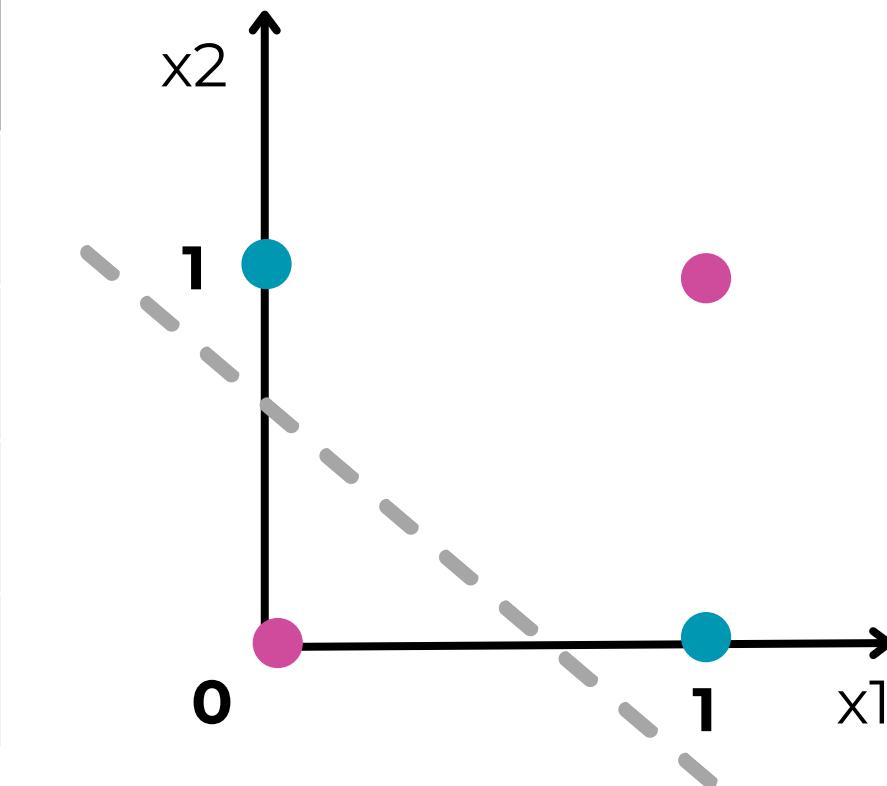
AND

X1	X2	0
1	0	0
0	0	0
0	1	0
1	1	1



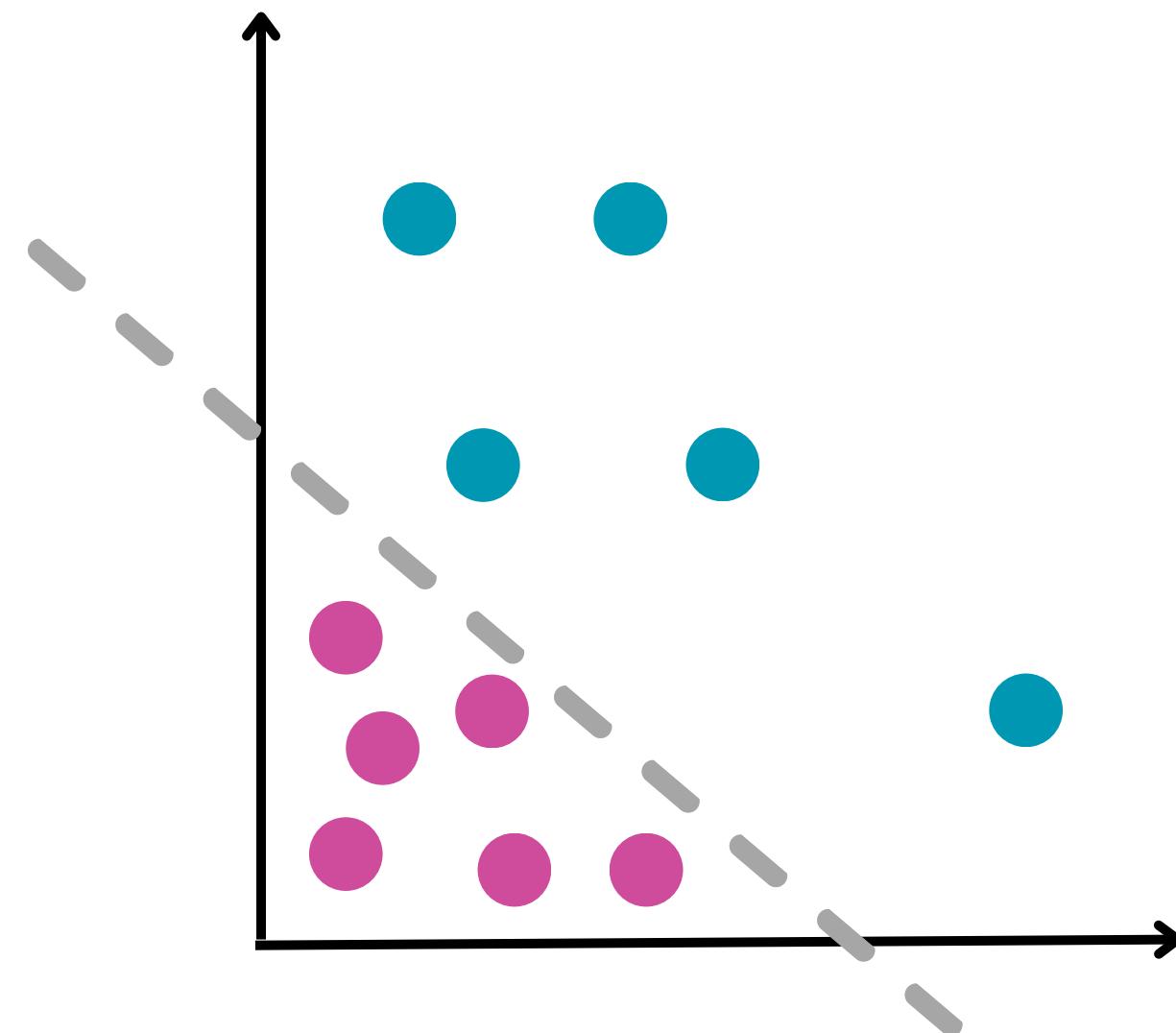
XOR

X1	X2	0
1	0	0
0	0	1
0	1	0
1	1	1

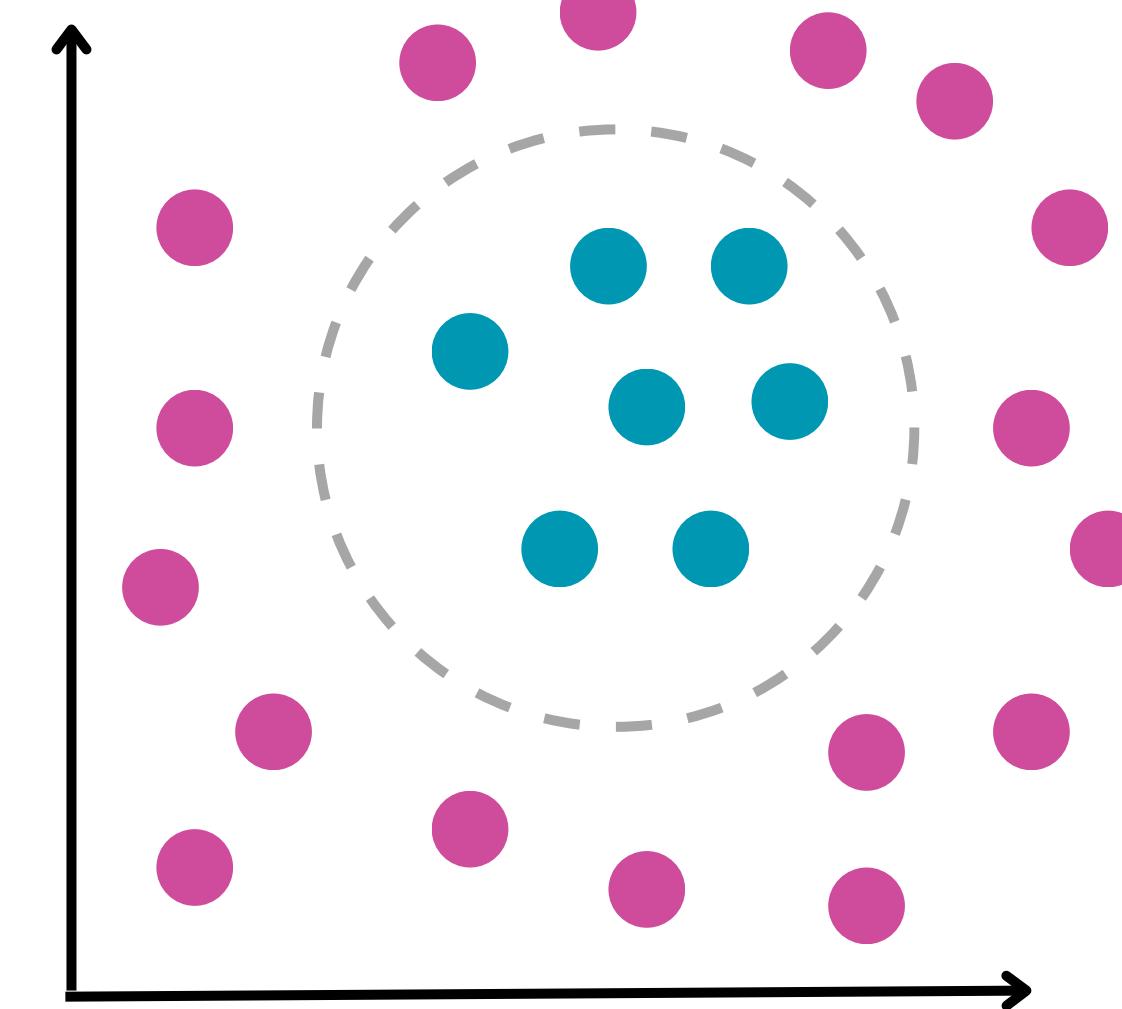


Limitações do Perceptron

linearmente separável



não linearmente separável



Primeiro Inverno da IA

- 1969:

O livro *Perceptrons* mostrou matematicamente que o perceptron simples era incapaz de resolver problemas não linearmente separáveis.

Minsky e Papert argumentaram que essas limitações tornavam o perceptron inadequado para muitos problemas práticos.

Isso desmotivou investimentos em pesquisa de redes neurais

- O inverno da IA (1970 a 1980):

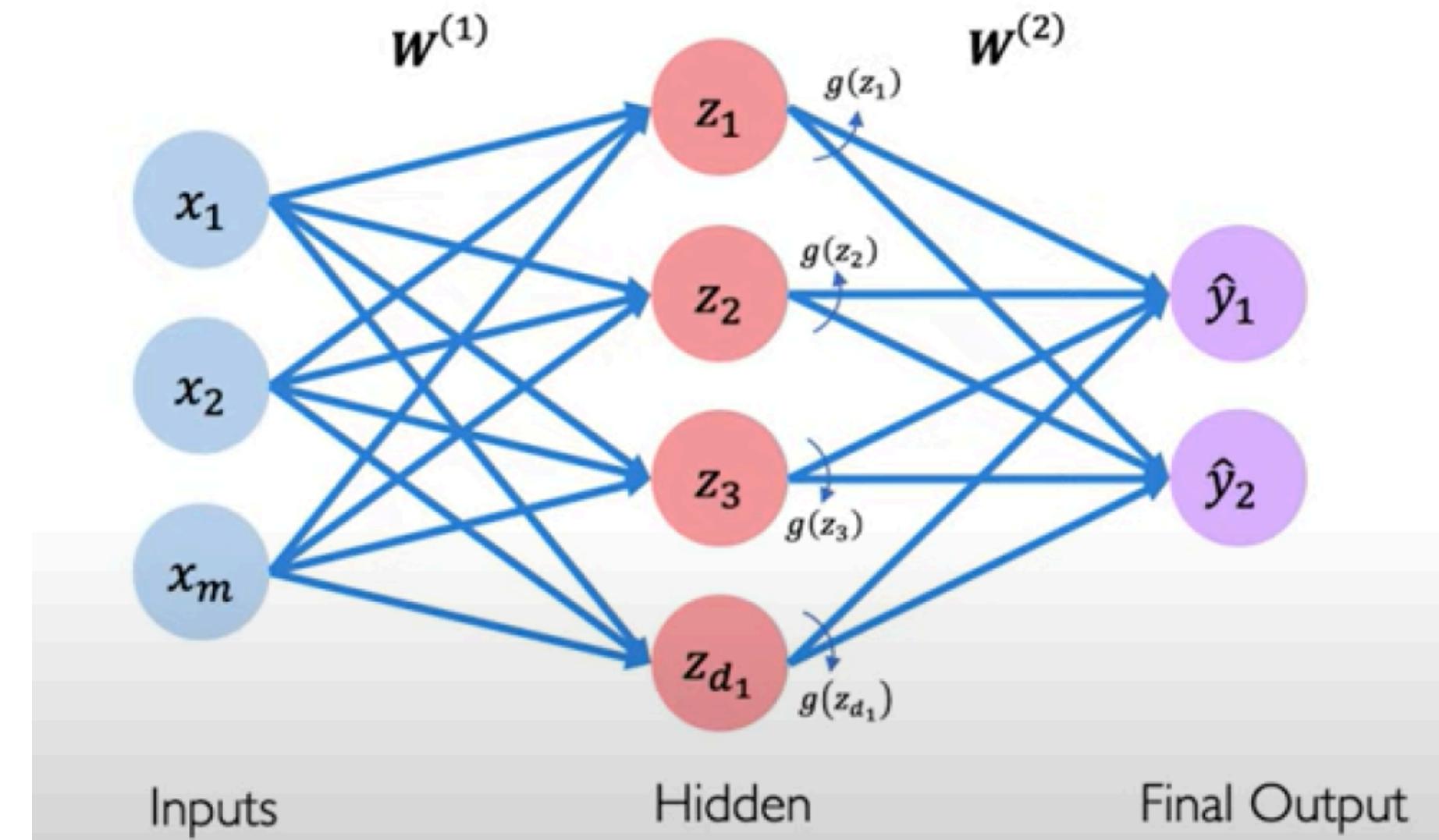
Com a retirada de financiamento, a pesquisa em redes neurais foi praticamente abandonada.

O foco mudou para sistemas baseados em regras, como sistemas especialistas, que dominavam a pesquisa em IA na época.



Perceptron Layer

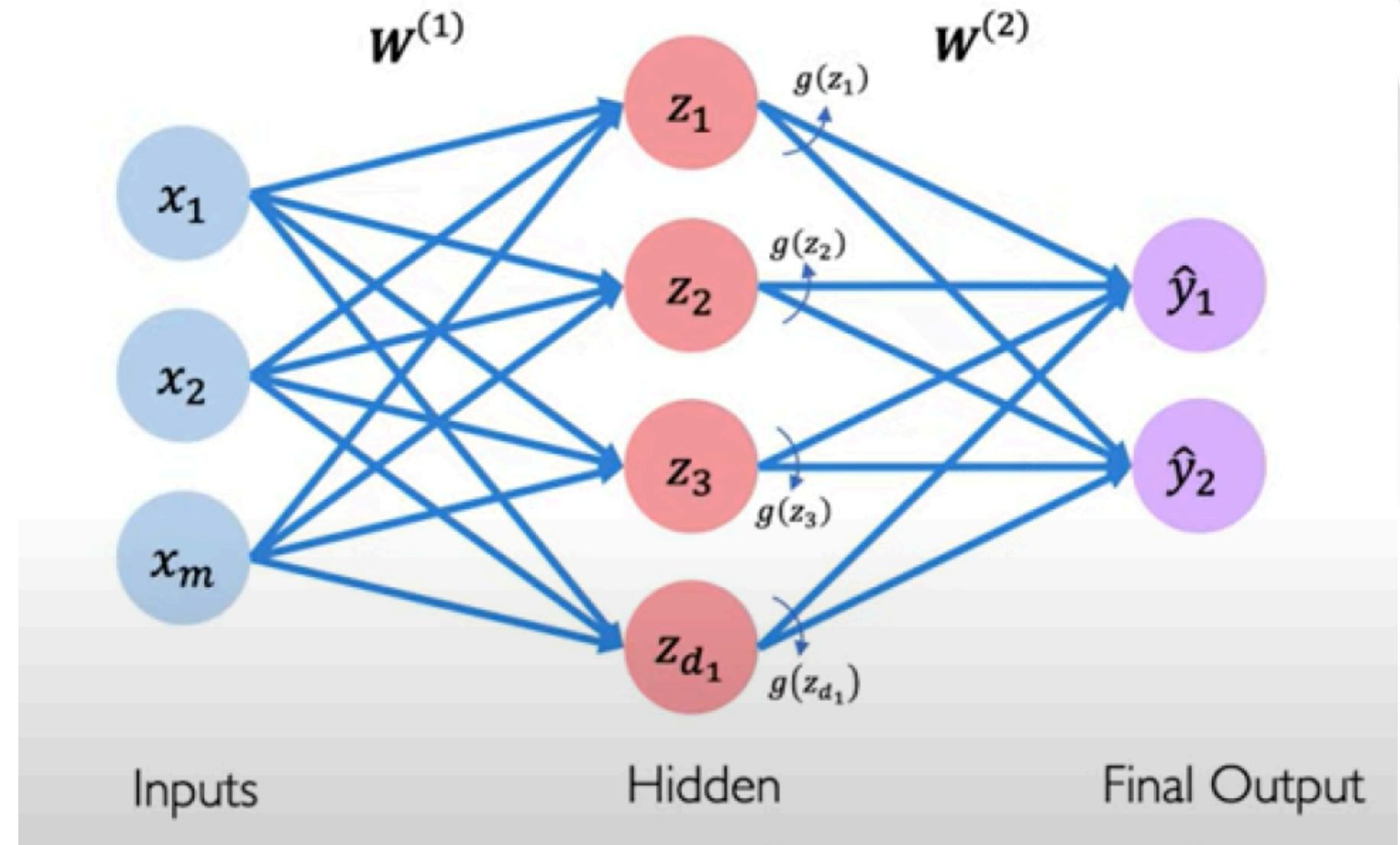
- Imagine se tivéssemos vários perceptrons pegando o mesmo input
 - Neurônios no cérebro fazem várias conexões
 - Hipótese: vários perceptrons juntos podem extrair várias características
- Cada perceptron (com pesos e bias único) podem ser alocados num layer



Perceptron Layer

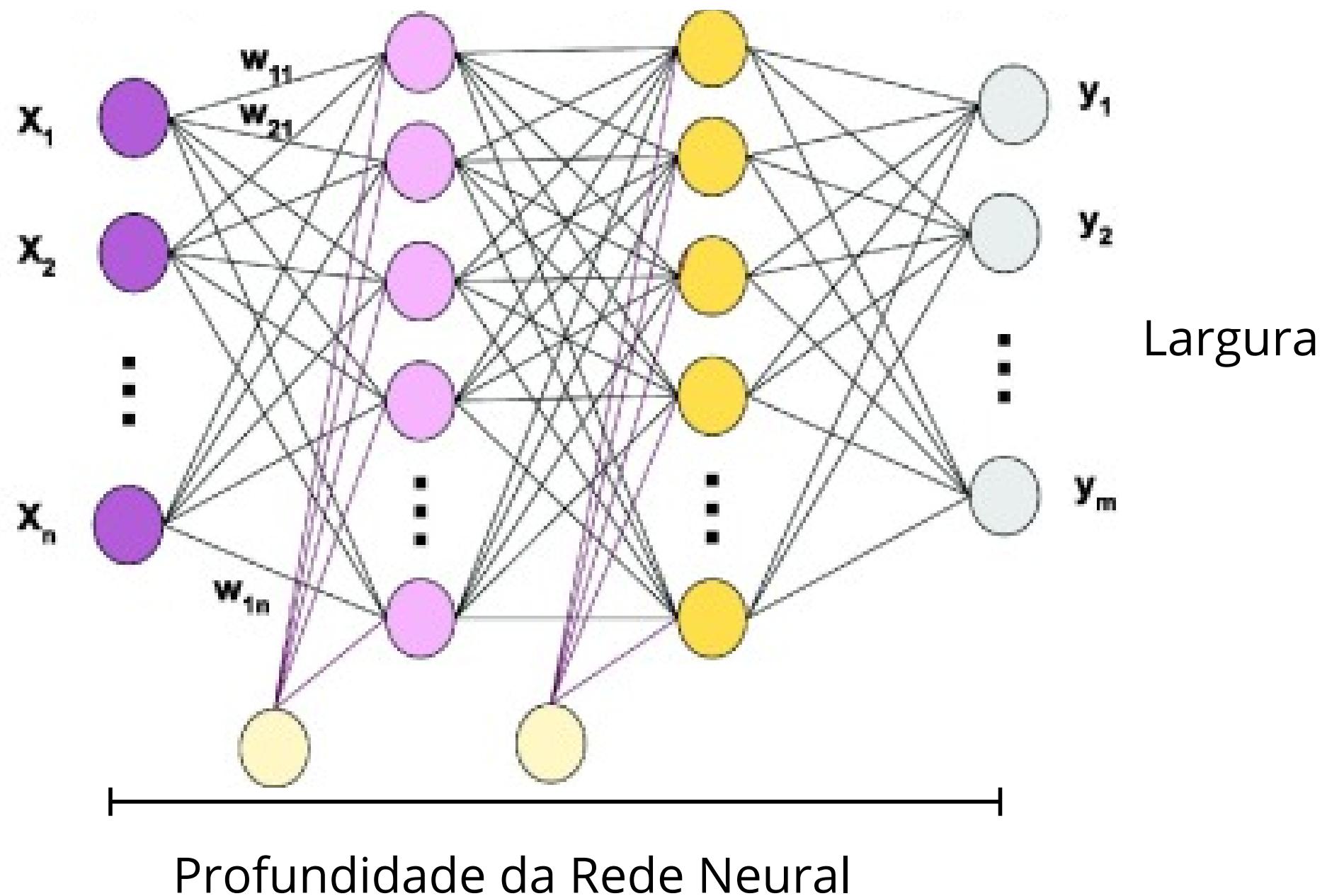
Cada neurônio será composto por:

- 1- Pesos
- 2- Bias
- 3- Função de ativação



$$f(\bar{x}, \bar{w}_1, \bar{w}_2, \dots, \bar{w}_n, b_1, b_2, \dots, b_n) = \begin{bmatrix} \text{ReLU}(\bar{w}_1^\top \bar{x} + b_1) \\ \text{ReLU}(\bar{w}_2^\top \bar{x} + b_2) \\ \vdots \\ \text{ReLU}(\bar{w}_n^\top \bar{x} + b_n) \end{bmatrix}$$

Perceptron Layer



- Largura: quantos neurônios podemos alocar
- Profundidade: quantidade de layers
- Teorema: uma rede neural com largura infinita pode aproximar qualquer função existente
- Como a largura fica muito grande, descobriram que o aumento de profundidade ajuda na resolução de problemas, por que ? Não sabemos ...

Funções de ativação

E se tirarmos a função de ativação ?

Antes

$$\begin{bmatrix} \text{ReLU}(\bar{w}_1^\top \bar{x} + b_1) \\ \text{ReLU}(\bar{w}_2^\top \bar{x} + b_2) \\ \vdots \\ \text{ReLU}(\bar{w}_n^\top \bar{x} + b_n) \end{bmatrix}$$

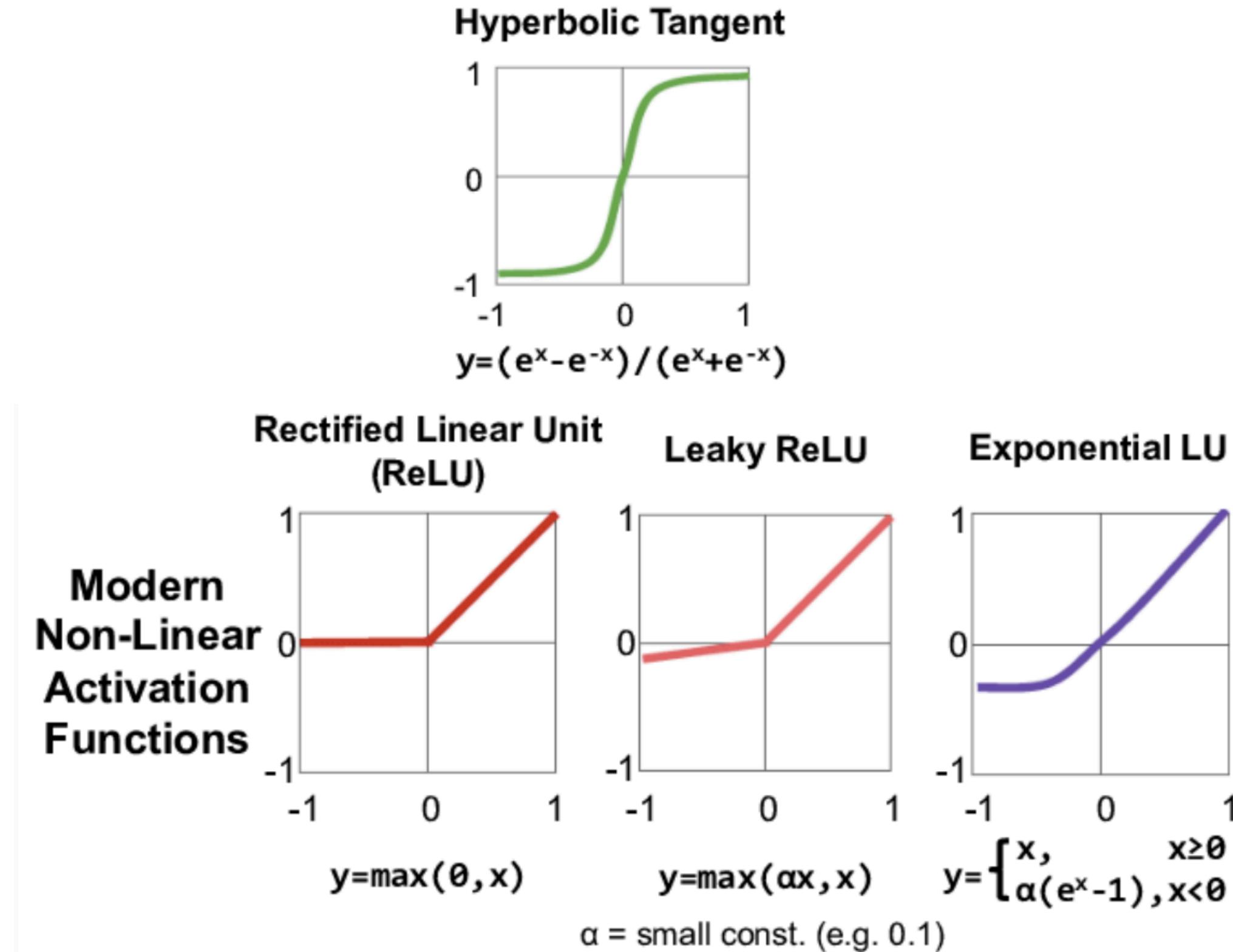


Depois

$$\begin{aligned} f(\bar{x}, W_1, W_2, \bar{b}_1, \bar{b}_2) &= W_2(W_1\bar{x} + \bar{b}_1) + \bar{b}_2 \\ &= W_2W_1\bar{x} + (W_2\bar{b}_1 + \bar{b}_2) \\ &= W_3\bar{x} + \bar{b}_3 \end{aligned}$$

Sem função de ativação, vai ser apenas um grande layer **linear**

Funções de ativação - exemplos



Função de Output

Classificação binária

Sigmoide $\frac{1}{1 + e^{-x}}$

Classificação multiclasse

Softmax
$$\frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

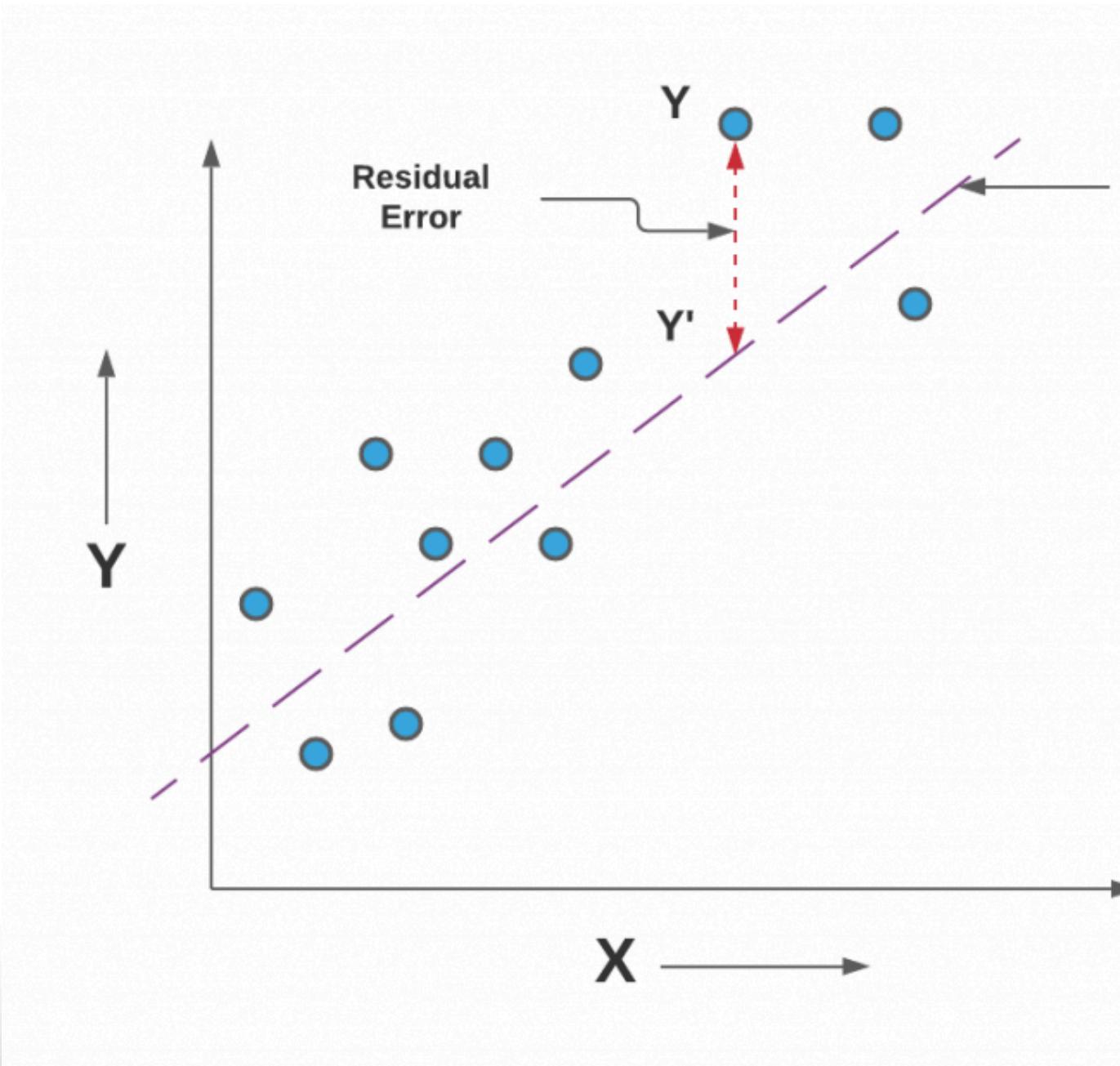
Regressão (valores contínuos)

$$F(x) = Wx + B$$

LOSS

- Quando inicializamos nossa rede com pesos e bias aleatórios, provavelmente nosso resultado de primeira vai ser muito ruim
 - Temos que achar um jeito de ajustar tudo para melhorar o resultado
- Loss
 - Busca para quantificar o quanto ruim está sendo nossa previsão comparada ao mundo real
- Detalhe: em aprendizado profundo precisamos que nossa Loss seja diferenciável, ou seja, existe a derivada da loss
 - Por que? Explicaremos mais para frente...
- Exemplo: cobrança de falta

LOSS - exemplo: MSE



$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\text{Error})^2$$

Mean Error Squared

$$(\hat{Y}_i - Y_i)^2$$

Para calcular o MSE, primeiro calculamos as diferenças quadradas entre cada valor real e previsto:

Squared Differences: $[(10-12)^2, (20-18)^2, (30-32)^2, (40-38)^2, (50-48)^2]$
= $[4, 4, 4, 4, 4]$

Em seguida, fazemos a média dessas diferenças quadradas para obter o MSE:

$$\begin{aligned}\text{MSE} &= (4 + 4 + 4 + 4 + 4) / 5 \\ &= 20 / 5 \\ &= 4\end{aligned}$$

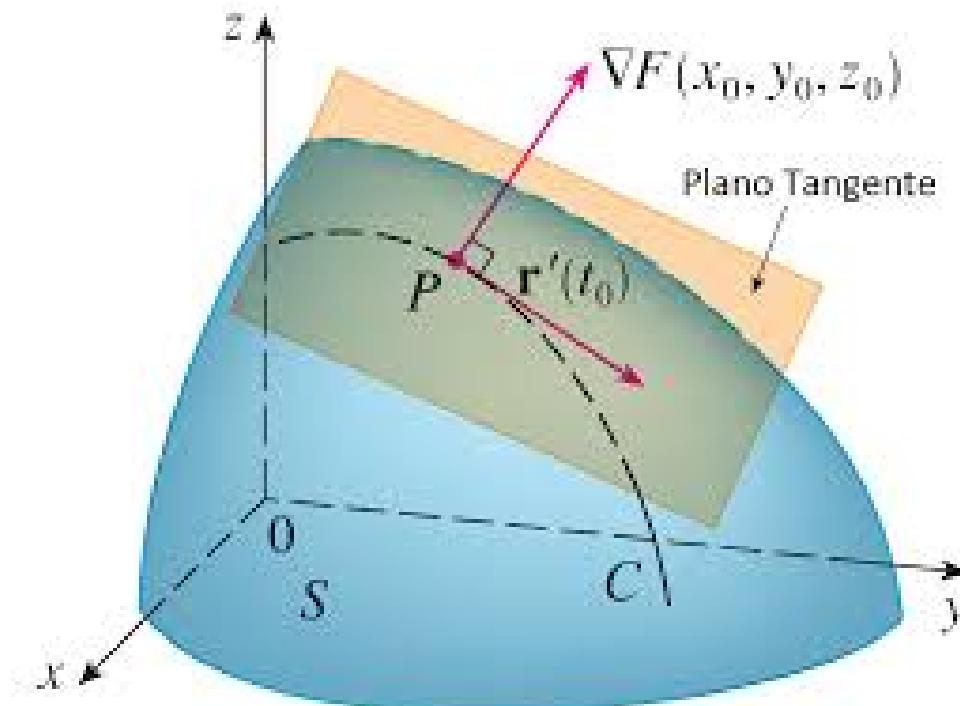
- Agora que conseguimos quantificar o quanto ruim nosso modelo está, em qual direção devo mudar os parâmetros para diminuir minha loss?

Gradiente Descendente

Gradiente Descendente

Vetor Gradiente: em determinado ponto, em qual direção minha função vai crescer mais rapidamente?

$$\nabla_{\theta} \text{MSE}(\theta) = \begin{bmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{bmatrix} = \frac{2}{m} X^T \cdot (X \cdot \theta - y)$$



Powered By Embed Fun

Vetor Gradiente

- n é o número de características;
- X^T é transposição da matriz de características.
- y é o vetor dos valores do alvo.

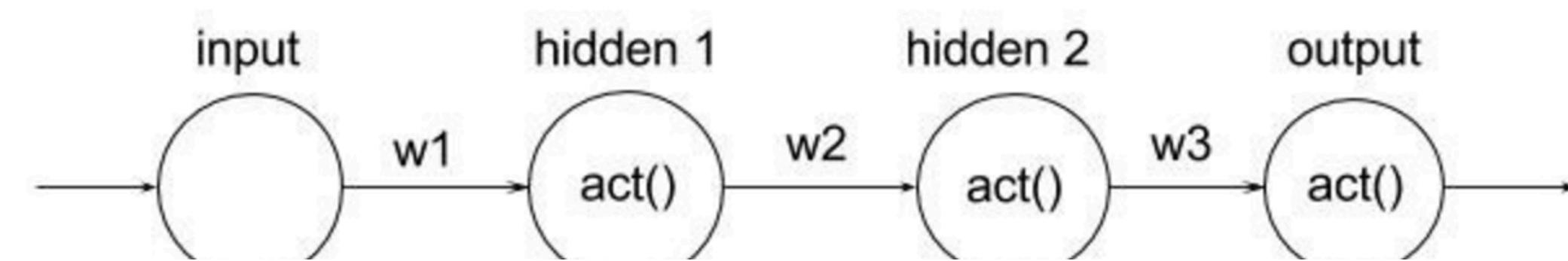
Atualização do Gradiente Descendente - exemplo MSE

O vetor gradiente aponta para o crescimento da função, como desejamos descer, basta caminharmos para o lado oposto (subtraindo o vetor de θ). A taxa de aprendizado é definida por η (Eta), multiplicamos o vetor gradiente por η para definir o tamanho do passo.

$$\theta^{(nextstep)} = \theta - \eta \nabla_{\theta} \text{MSE}(\theta)$$

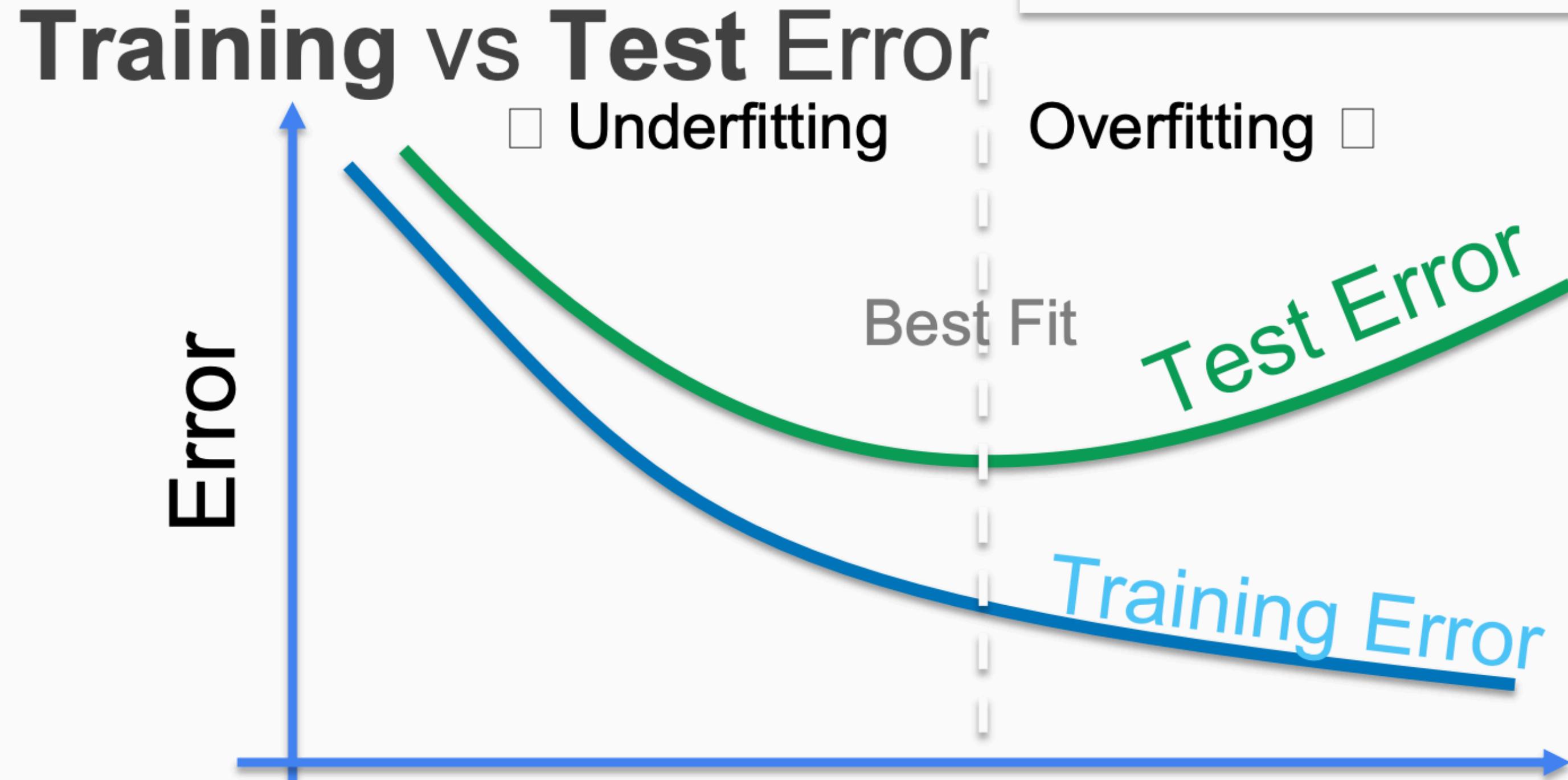
Backpropagation

- Uso da regra da cadeia para decompor o cálculo do gradiente da função de loss com respeito aos pesos da rede
- Isso é feito "de trás para frente", começando na saída e propagando os gradientes camada por camada para os pesos mais próximos da entrada.



$$\frac{\partial \text{error}}{\partial w1} = \frac{\partial \text{error}}{\partial \text{output}} * \frac{\partial \text{output}}{\partial \text{hidden2}} * \frac{\partial \text{hidden2}}{\partial \text{hidden1}} * \frac{\partial \text{hidden1}}{\partial w1}$$

Conceitos no treinamento



Obrigado!

INF
INSTITUTO DE
INFORMÁTICA



Referências

https://docs.google.com/presentation/d/1uLPQQxQ87SO0ExITiEs7BNNAbNL3SGy5t_XvLPcxr6w/edit#slide=id.g239c805b7de_0_174

<https://medium.com/@avinicius.adorno/redes-neurais-artificiais-418a34ea1a39>

Links de apoio:

- Notion da Universidade de Berkeley: [Deep Learning Fundamentals](#)
- Colab do FreeCodeCamp: [Implementando MLP](#)
- Curso NPTEL: [Lectures sobre Perceptron e MLP](#)