

# Redes Multi-Layer Perceptron propagation

## Conceitos Arquiteturais e Forward

*Discentes: João Pedro de Alcântara Lima  
José Marques Cardoso Souza*

# Introdução



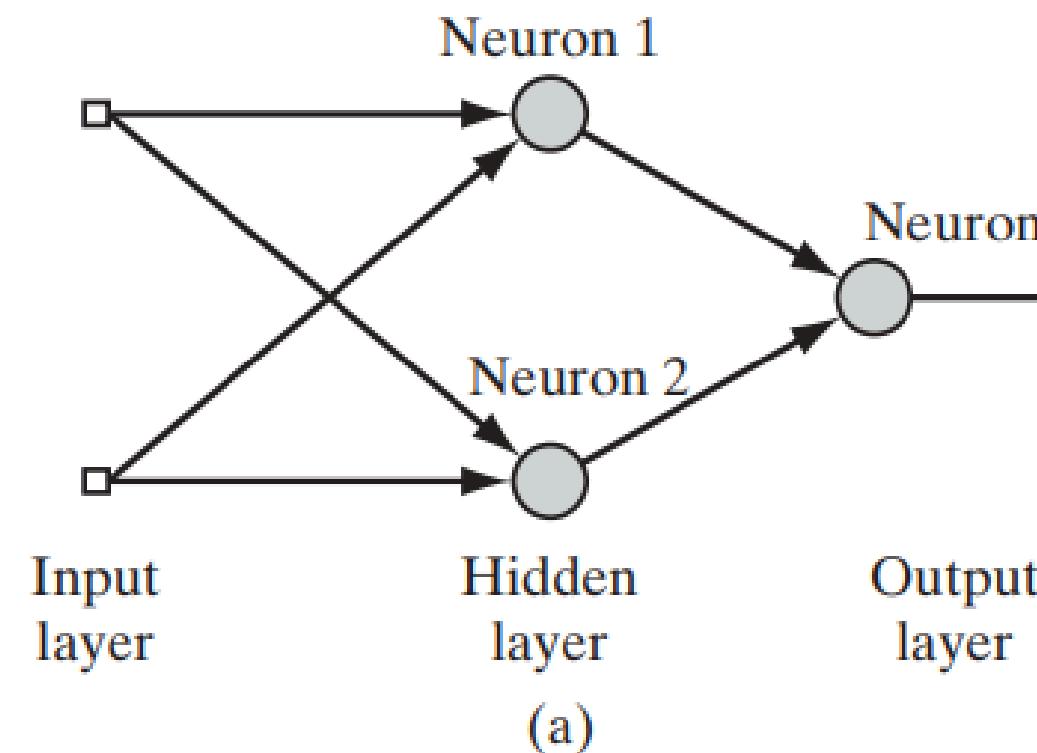
Redes Multi-Layer Perceptron (MLP) com base na literatura “*Neural Networks and Learning Machines* – Simon Haykin – Third Edition” são um tipo de rede neural artificial feedforward composta por múltiplas camadas de neurônios, incluindo:

- Camada de entrada: Recebe os dados de entrada da rede.
- Camadas ocultas: São as camadas intermediárias onde a rede realiza o aprendizado e extrai informações dos dados.
- Camada de saída: Produz a saída final da rede.

Elas são capazes de aprender relações complexas e não lineares entre entradas e saídas, tornando-as poderosas para tarefas como classificação, regressão e aproximação de funções.

# Motivação para desenvolvimento das Redes Multi-Layer Perceptron (MLP)

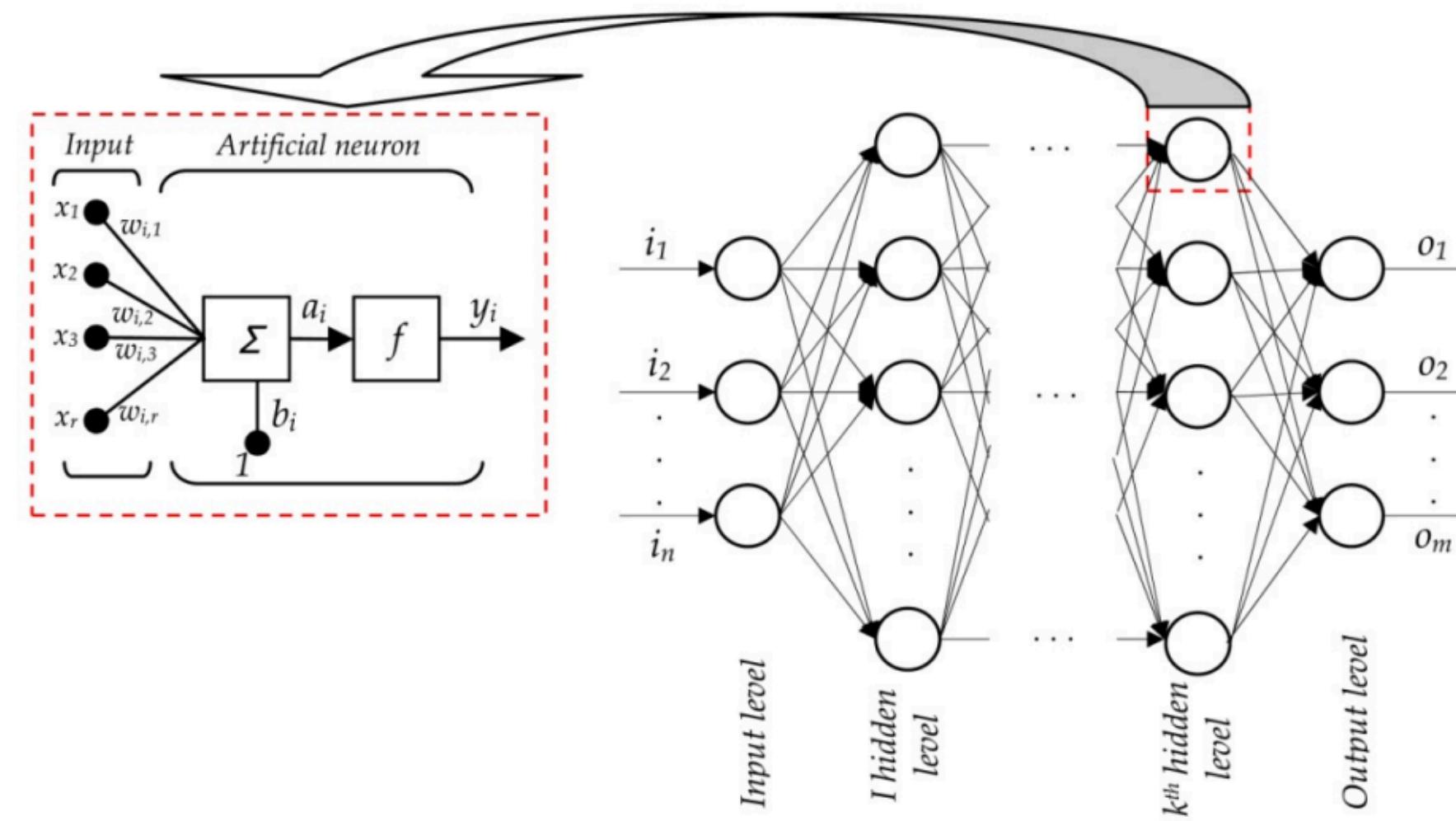
- Perceptron de camada única não resolve XOR (não linearmente separável).
- MLP com pelo menos uma camada oculta resolve.
- Foi o pivô da “crise das redes neurais” nos anos 1970, após o livro de Minsky e Papert (*Perceptrons*, 1969).
- O algoritmo de backpropagation foi redescoberto e popularizado por Rumelhart, Hinton e Williams em 1986.



**FIGURE 4.8** (a) Architectural graph of network for solving the XOR problem. (b) Signal-flow graph of the network.

# Representação da MLP

A RNA Multilayer Perceptron é uma generalização do Perceptron



- Evolução dos perceptrons simples
- Supera limitações do perceptron de Rosenblatt e do algoritmo LMS
- Aplicações em classificação e regressão

# Representação da MLP

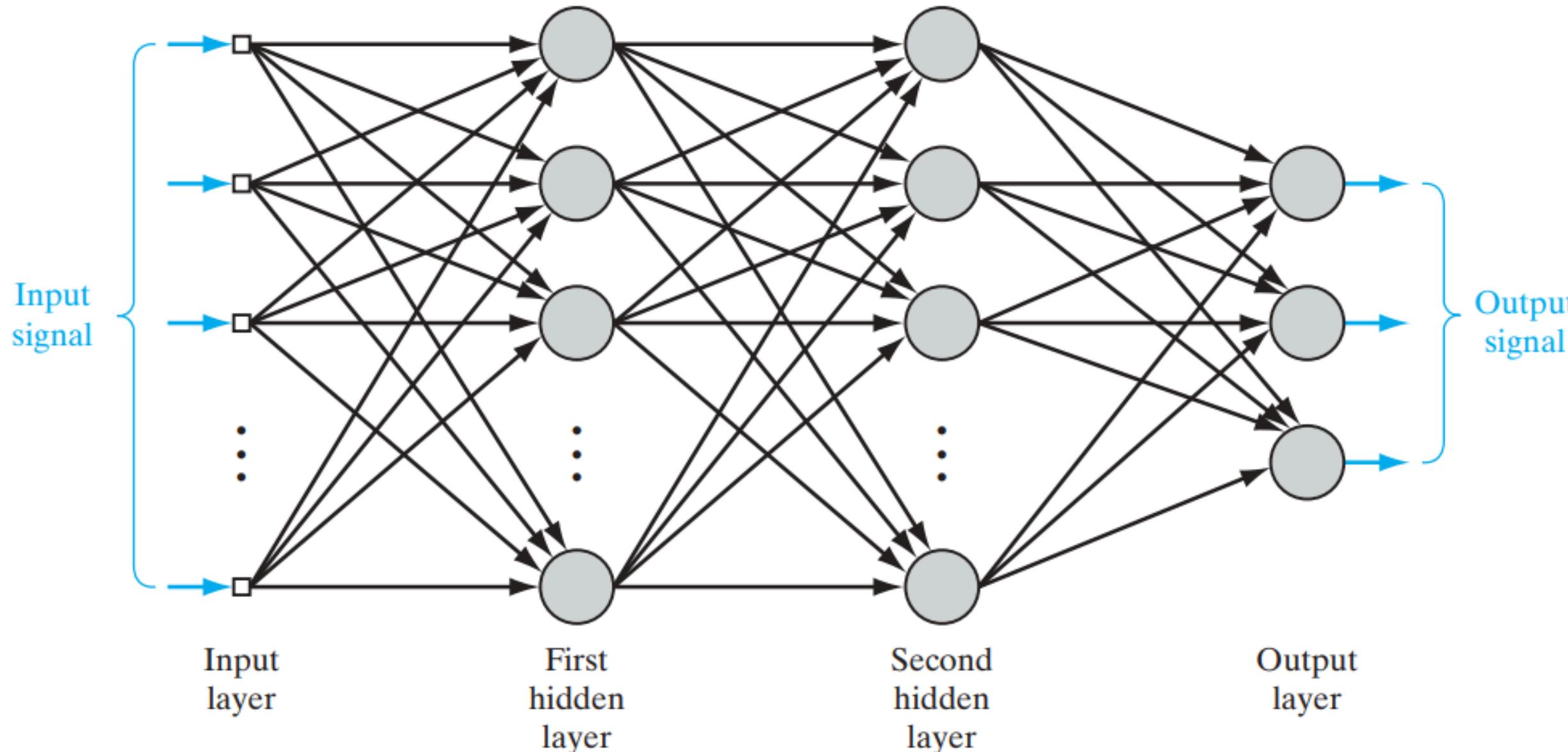
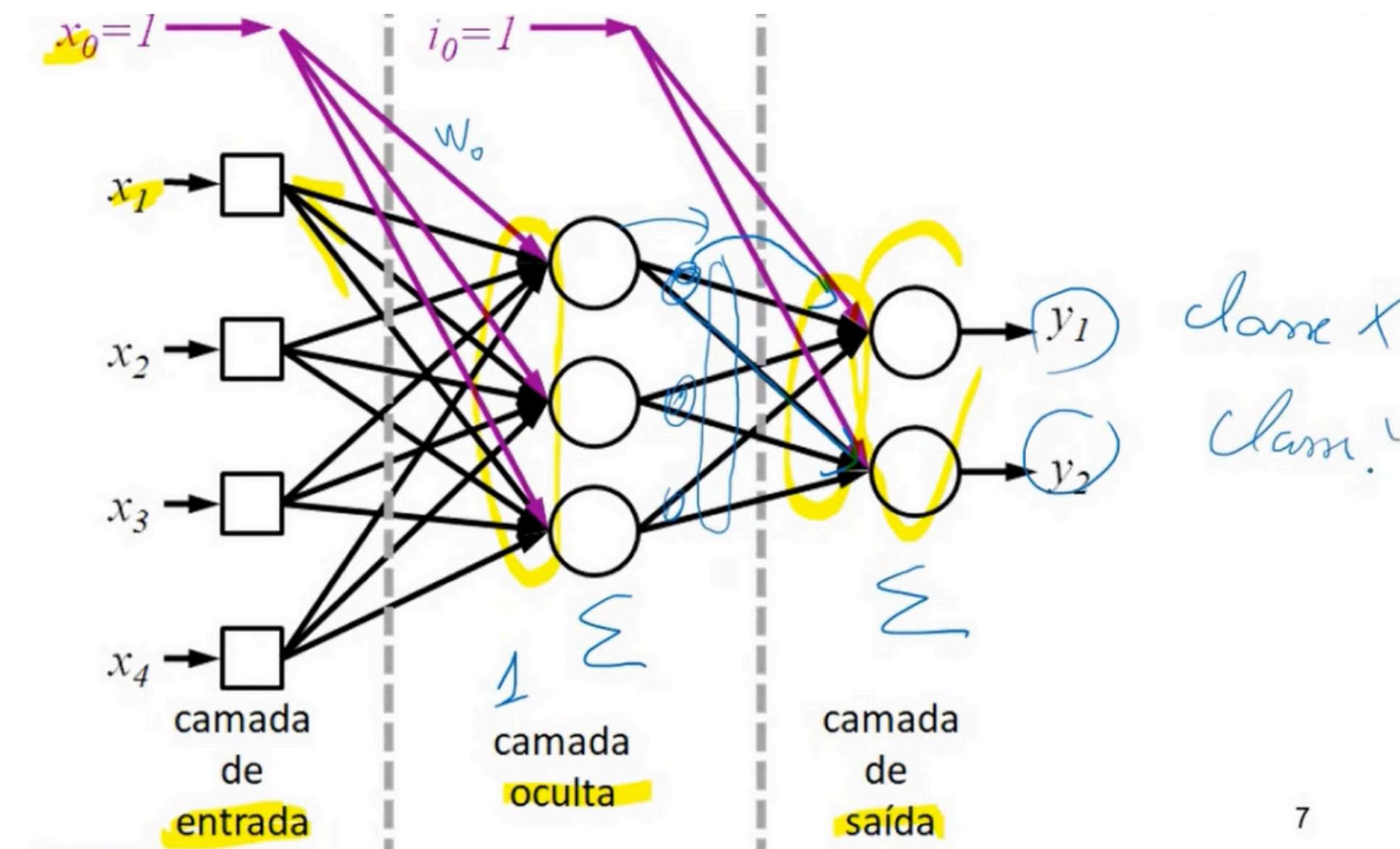


FIGURE 4.1 Architectural graph of a multilayer perceptron with two hidden layers.

**Características fundamentais:**

- Neurônios com função de ativação não linear e diferenciável.
- Camadas ocultas que não estão diretamente ligadas nem à entrada nem à saída.
- Alta conectividade, determinada pelos pesos sinápticos.

# Exemplo de MLP



7

# Conceitos arquiteturais sobre Redes Multi-Layer Perceptron (MLP)

Cada neurônio oculto ou de saída de um perceptron multicamadas é projetado para realizar dois cálculos:

1. O cálculo do sinal de função: que aparece na saída de cada neurônio, que é expresso como uma função não linear contínua do sinal de entrada e dos pesos sinápticos associados a esse neurônio;
2. o cálculo de uma estimativa do vetor gradiente (ou seja, os gradientes da superfície de erro em relação aos pesos conectados às entradas de um neurônio).

## 1. Function of the Hidden Neurons

Atuam como detectores de características, transformando não linearmente os dados de entrada em um espaço onde as classes se tornam mais separáveis.

## 2. Credit-Assessment Problem

O problema de credit-assessment exerce responsabilidade pelos resultados da rede aos neurônios ocultos, e é resolvido pelo algoritmo de backpropagation no treinamento do perceptron multicamadas.

# Conceitos arquiteturais sobre Redes Multi-Layer Perceptron (MLP)

## Sinais de Função:

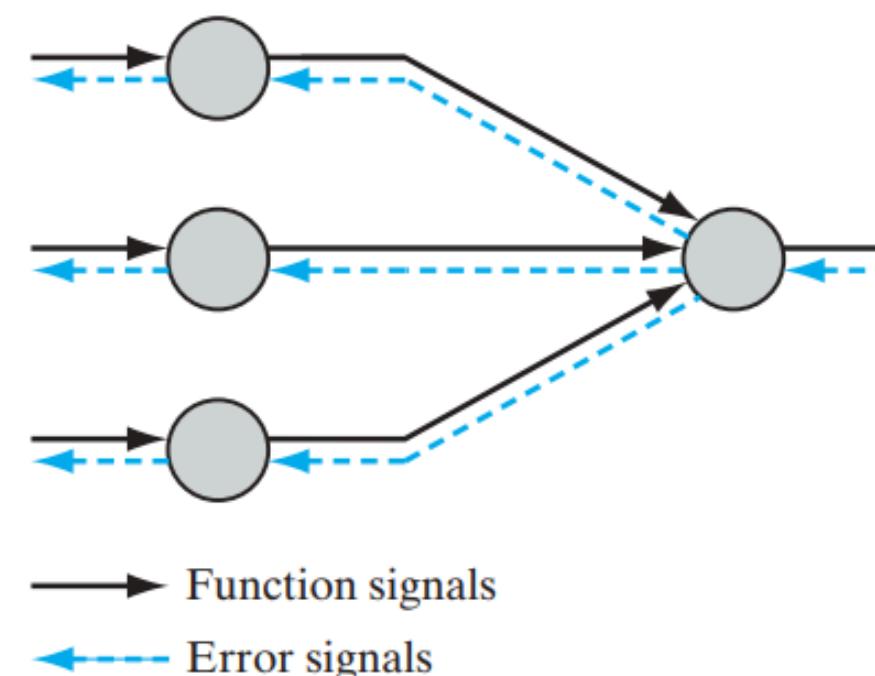
São os sinais de entrada que percorrem a rede no sentido direto (**forward**), passando de neurônio em neurônio até gerar a saída. Eles são chamados assim porque:

- Contribuem para o resultado final da rede (**função útil**).
- Em cada neurônio, são processados como uma função das entradas e dos pesos.

## Sinais de Erro:

São os sinais que se originam na saída da rede e se propagam para trás (**backward**), camada por camada.

- São usados para ajustar os pesos com base no erro da saída.
- Cada neurônio calcula esse sinal com base em uma função relacionada ao erro.



**FIGURE 4.2** Illustration of the directions of two basic signal flows in a multilayer perceptron: forward propagation of function signals and back propagation of error signals.

# Batch Learning (Aprendizado em lote):

## Conceito Geral

- No aprendizado em lote, os pesos da rede são atualizados após a apresentação de todos os exemplos do conjunto de treinamento.
- Cada ciclo completo sobre os dados é chamado de época.
- A função de custo usada é a energia média de erro:

$$\mathcal{E}_{\text{eq}}(N) = \frac{1}{N} \sum_{n=1}^N \mathcal{E}(n) = \frac{1}{2N} \sum_{n=1}^N \sum_{k \in \mathcal{C}} e_k^2(n)$$

Definição dos termos:

- $N$ : número total de exemplos no conjunto de treinamento.
- $\mathcal{E}(n)$ : erro no exemplo  $n$ .
- $\mathcal{C}$ : conjunto de neurônios de saída.
- $e_k(n)$ : erro do neurônio de saída  $k$  no exemplo  $n$ .
- $\mathcal{E}_{\text{eq}}(N)$ : energia média do erro para todos os exemplos.

# Batch Learning (Aprendizado em lote):

## Processamento por Épocas

- Os exemplos são **embaralhados aleatoriamente** a cada época.
- A **curva de aprendizado** é baseada em **várias execuções** com diferentes condições iniciais.
- A **curva final** é obtida pela **média das curvas** (ensemble averaging).

## Vantagens do Aprendizado em Lote

- Estimativa precisa do vetor gradiente:

$$\nabla \mathcal{E}_{\text{eq}}$$

- Garante **convergência** (sob condições simples) para um **mínimo local**.
- Permite **paralelização** do processo de aprendizado.

## Desvantagens e Observações

- **Alta demanda de armazenamento**, pois todos os exemplos devem estar disponíveis ao mesmo tempo.
- Pode ser interpretado como uma forma de **inferência estatística**.

# Batch Learning (Aprendizado em lote):

```
==== TREINAMENTO BATCH LEARNING ====
Configuração: LR=0.1, Epochs=50, Batch Size=Todos (455 amostras)
-----
Época 1/50 | Perda: 0.6938 | Acurácia: 0.3978
Época 6/50 | Perda: 0.6091 | Acurácia: 0.8725
Época 11/50 | Perda: 0.5221 | Acurácia: 0.9099
Época 16/50 | Perda: 0.4412 | Acurácia: 0.9231
Época 21/50 | Perda: 0.3754 | Acurácia: 0.9341
Época 26/50 | Perda: 0.3249 | Acurácia: 0.9407
Época 31/50 | Perda: 0.2868 | Acurácia: 0.9407
Época 36/50 | Perda: 0.2577 | Acurácia: 0.9451
Época 41/50 | Perda: 0.2353 | Acurácia: 0.9451
Época 46/50 | Perda: 0.2176 | Acurácia: 0.9473
Época 50/50 | Perda: 0.2061 | Acurácia: 0.9451
-----
Treinamento Batch Learning concluído!

==== Batch Learning (Teste) ====
[[38  4]
 [ 2 70]]
Legenda: [ [Verdadeiro Negativo, Falso Positivo], [Falso Negativo, Verdadeiro Positivo] ]
```

# Aprendizado On-line (On-line Learning)

## Conceito Geral

- No aprendizado on-line, os pesos da rede são ajustados exemplo por exemplo.
- A função de custo minimizada é a energia de erro instantânea total:

$$\mathcal{E}_i(n) = \frac{1}{2} e_i^2(n)$$

- $\mathcal{E}_i(n)$ :

É a energia de erro instantânea no instante  $n$  para o exemplo  $i$ .

Essa é a função de custo que mede o quanto errado está o modelo naquele instante.

- $\frac{1}{2}$ :

Um fator de conveniência matemática.

Ele facilita a derivada da função de custo (especialmente no cálculo do gradiente), pois a derivada do quadrado ( $e_i^2$ ) traz um fator 2, que é cancelado com esse  $\frac{1}{2}$ .

- $e_i(n)$ :

É o erro instantâneo cometido pela rede no instante  $n$  para o exemplo  $i$ .

Calculado por:

$$e_i(n) = d_i(n) - y_i(n)$$

Onde:

- $d_i(n)$ : saída desejada (ou esperada) para o exemplo  $i$ .
- $y_i(n)$ : saída produzida pela rede para o mesmo exemplo.
- $e_i^2(n)$ :

É o erro ao quadrado, que penaliza erros grandes mais fortemente e garante que o custo seja sempre positivo (ou zero).

# Aprendizado On-line (On-line Learning)

## Processo por Época de N Exemplos

- Cada par  $[x(i), d(i)]$  é processado individualmente com atualização imediata dos pesos:  
Em vez de esperar todos os exemplos para fazer uma única atualização (como no batch), o aprendizado on-line ajusta os pesos da rede a cada novo exemplo apresentado, imediatamente após o processamento.
- O ciclo continua até que todos os N exemplos tenham sido processados:  
Isso define uma "época": é quando todos os dados disponíveis (N exemplos) foram vistos uma vez, um por um, com atualização após cada um.
- Paralelização difícil, mas resposta é quase imediata:  
Como os exemplos são processados sequencialmente com atualizações contínuas, é mais difícil usar múltiplos processadores. Por outro lado, o sistema responde e aprende rapidamente, ideal para aplicações em tempo real.

# Aprendizado On-line (On-line Learning)

```
== TREINAMENTO ONLINE LEARNING ==
Configuração: LR=0.01, Epochs=50, Batch Size=1 (amostra por amostra)
-----
Época 1/50 | Perda: 0.3873 | Acurácia: 0.9451
Época 2/50 | Perda: 0.1794 | Acurácia: 0.9516
Época 3/50 | Perda: 0.1474 | Acurácia: 0.9560
Época 4/50 | Perda: 0.1366 | Acurácia: 0.9582
Época 5/50 | Perda: 0.1319 | Acurácia: 0.9582
Época 6/50 | Perda: 0.1291 | Acurácia: 0.9560
Época 7/50 | Perda: 0.1273 | Acurácia: 0.9560
Época 8/50 | Perda: 0.1263 | Acurácia: 0.9538
Época 9/50 | Perda: 0.1256 | Acurácia: 0.9538
Época 10/50 | Perda: 0.1252 | Acurácia: 0.9516
Época 11/50 | Perda: 0.1249 | Acurácia: 0.9516
Época 12/50 | Perda: 0.1246 | Acurácia: 0.9516
Época 13/50 | Perda: 0.1244 | Acurácia: 0.9516
Época 14/50 | Perda: 0.1242 | Acurácia: 0.9516
Época 15/50 | Perda: 0.1239 | Acurácia: 0.9538
Época 16/50 | Perda: 0.1235 | Acurácia: 0.9560
Época 17/50 | Perda: 0.1231 | Acurácia: 0.9560
Época 18/50 | Perda: 0.1227 | Acurácia: 0.9560
Época 19/50 | Perda: 0.1224 | Acurácia: 0.9582
Época 20/50 | Perda: 0.1222 | Acurácia: 0.9582
Época 21/50 | Perda: 0.1220 | Acurácia: 0.9582
...
== Online Learning (Teste) ==
[[40 2]
 [ 4 68]]
Legenda: [ [Verdadeiro Negativo, Falso Positivo], [Falso Negativo, Verdadeiro Positivo] ]
```

# Aprendizado On-line (On-line Learning)

## Características

- **Estocástico: busca em espaço de pesos aleatório:**

Como a ordem dos exemplos afeta as atualizações, o processo é naturalmente estocástico, o que pode ajudar a explorar melhor o espaço de soluções.

- **Evita mínimos locais com mais eficiência:**

A aleatoriedade introduzida pelos exemplos pode ajudar o modelo a escapar de mínimos locais da função de custo — um problema comum em redes neurais.

- **Requer menos memória (sem armazenar todo o batch):**

O modelo só precisa de um exemplo por vez, o que o torna eficiente em ambientes com poucos recursos de memória.

- **Útil quando há poucos dados disponíveis de forma sequencial:**

Ideal para cenários em que os dados chegam em tempo real ou em streaming, como sensores ou rede social.

- **Detecta mudanças sutis em dados não estacionários:**

Por estar sempre se atualizando, o modelo é mais sensível e adaptável a **mudanças no padrão dos dados ao longo do tempo**, diferente do batch learning que pode ficar “preso” a padrões passados.

# Generalization em redes MLP

Conceito de Generalização

Definição (Haykin, Seção 4.11):

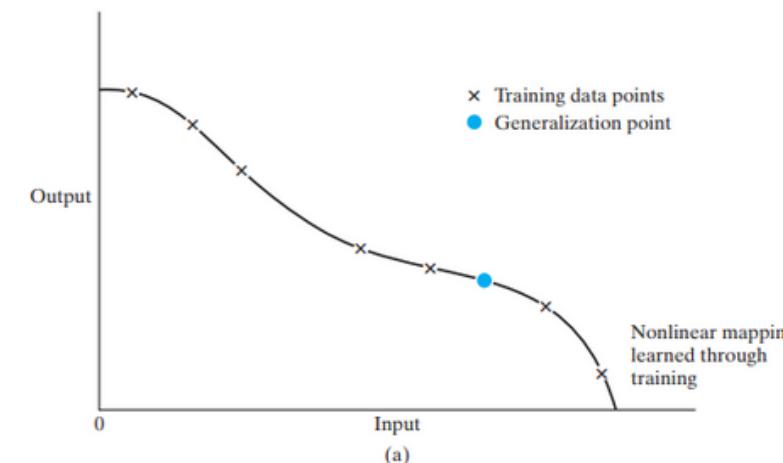
- "Capacidade da rede de produzir saídas corretas para dados não vistos durante o treinamento."

Processo de Generalização

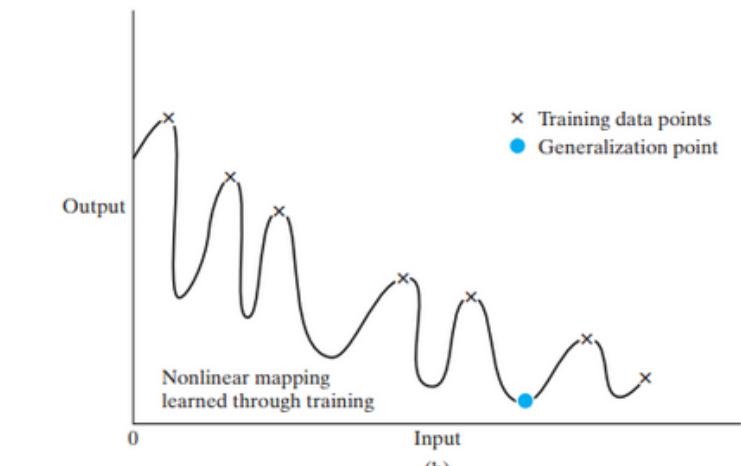
- Treinamento: A rede ajusta seus pesos (via backpropagation) para "encaixar" os dados de treino.
- Interpolação: Redes MLP com funções de ativação contínuas geram mapeamentos contínuos, permitindo generalização (Fig. 4.16a no Haykin).

Riscos: Overfitting (Sobreajuste)

- Causa: Memorização de ruídos ou padrões irrelevantes nos dados de treino.
- Sintoma: Baixo erro no treino, mas alto erro no teste.



(a) Mapeamento não linear adequadamente ajustado com boa generalização.



(b) Mapeamento não linear superajustado com generalização ruim.

Presença de ruídos nos dados de entrada

# Cross-Validation em redes MLP

## Conceito de Validação Cruzada

- Objetivo: Avaliar a capacidade de generalização do modelo e evitar overfitting.
- Definição: Técnica que divide o dataset em múltiplos subconjuntos para treino e validação iterativamente.

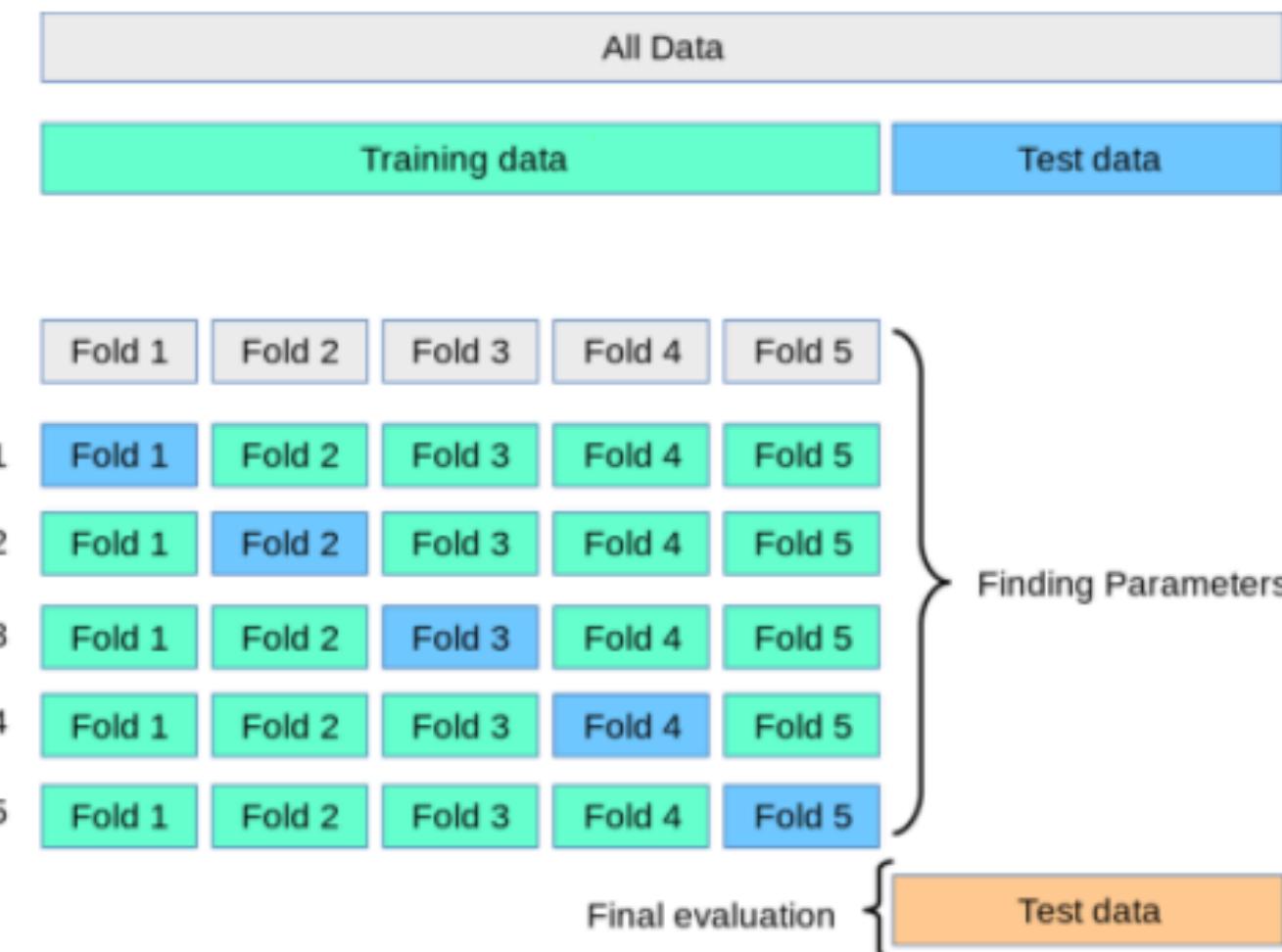
## Tipos de Validação Cruzada:

### A. k-Fold Cross-Validation (Mais comum em MLP)

Processo:

1. Divide os dados em k partes iguais.
2. Treina o modelo k vezes, usando cada parte como validação uma vez.

Vantagem: Utiliza todos os dados para treino e validação.

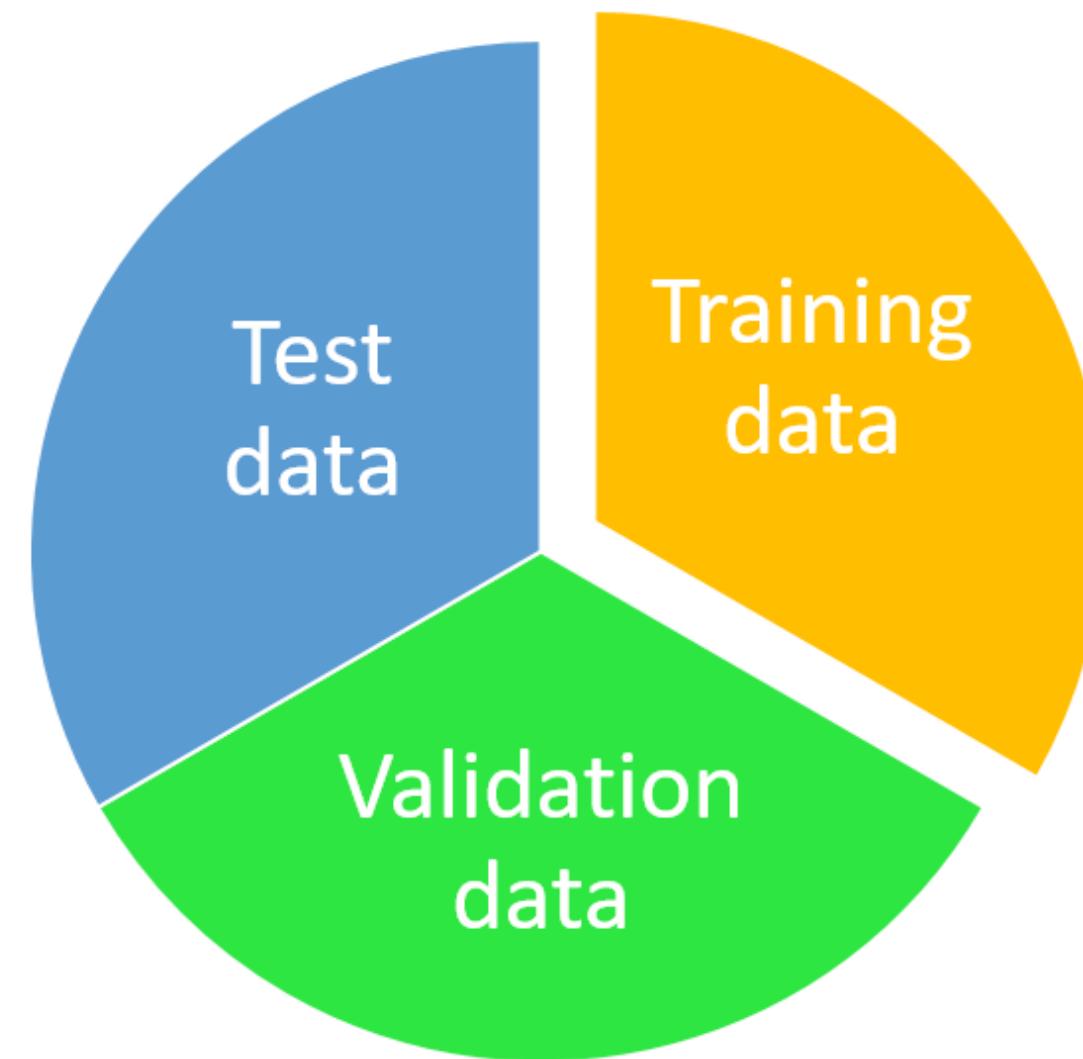


# Cross-Validation em redes MLP

## Tipos de Validação Cruzada:

### B. Holdout Method (Validação Simples)

- Divide em: 70% treino, 30% teste (ou 80/20).
- Uso: Rápido, mas menos robusto que k-Fold.

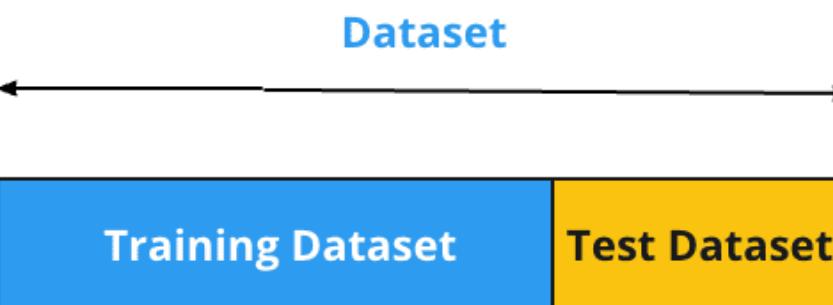


# Cross-Validation em redes MLP

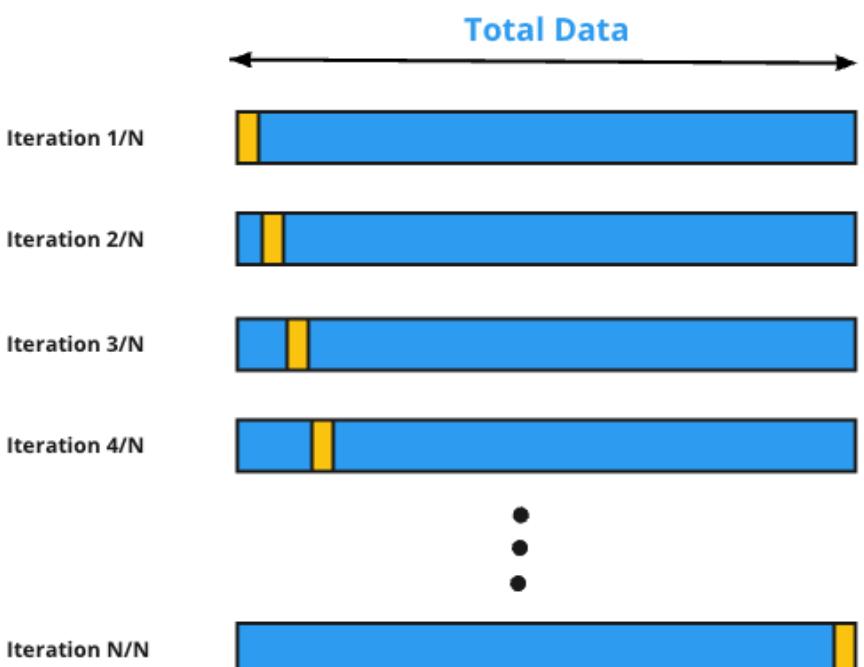
## Tipos de Validação Cruzada:

### C. Leave-One-Out (LOO)

- Extremo de k-Fold:  $k=N$  ( $N = \text{nº de amostras}$ ).
- Custo computacional alto, porém preciso.

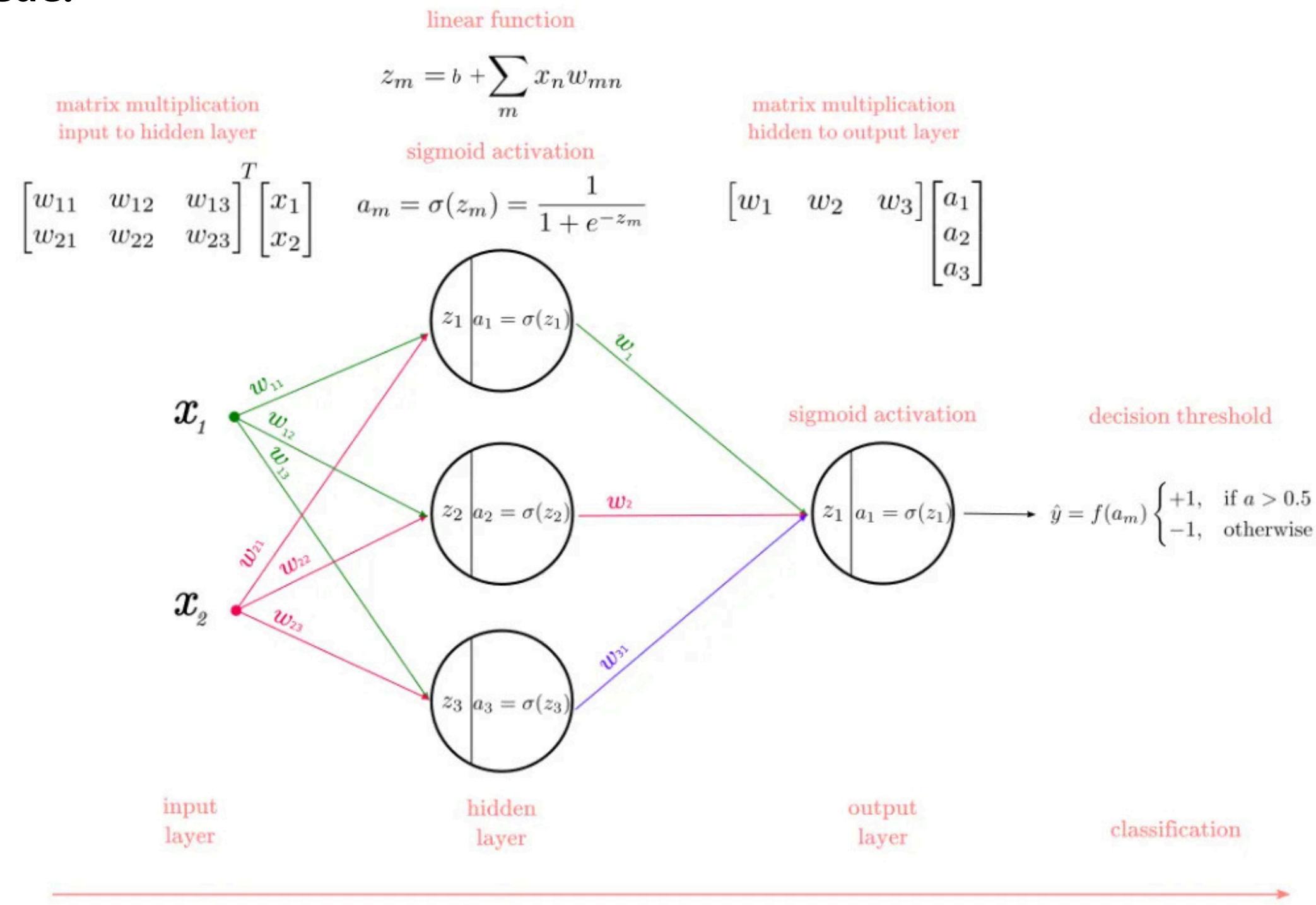


### LOOCV: Leave One Out Cross Validation

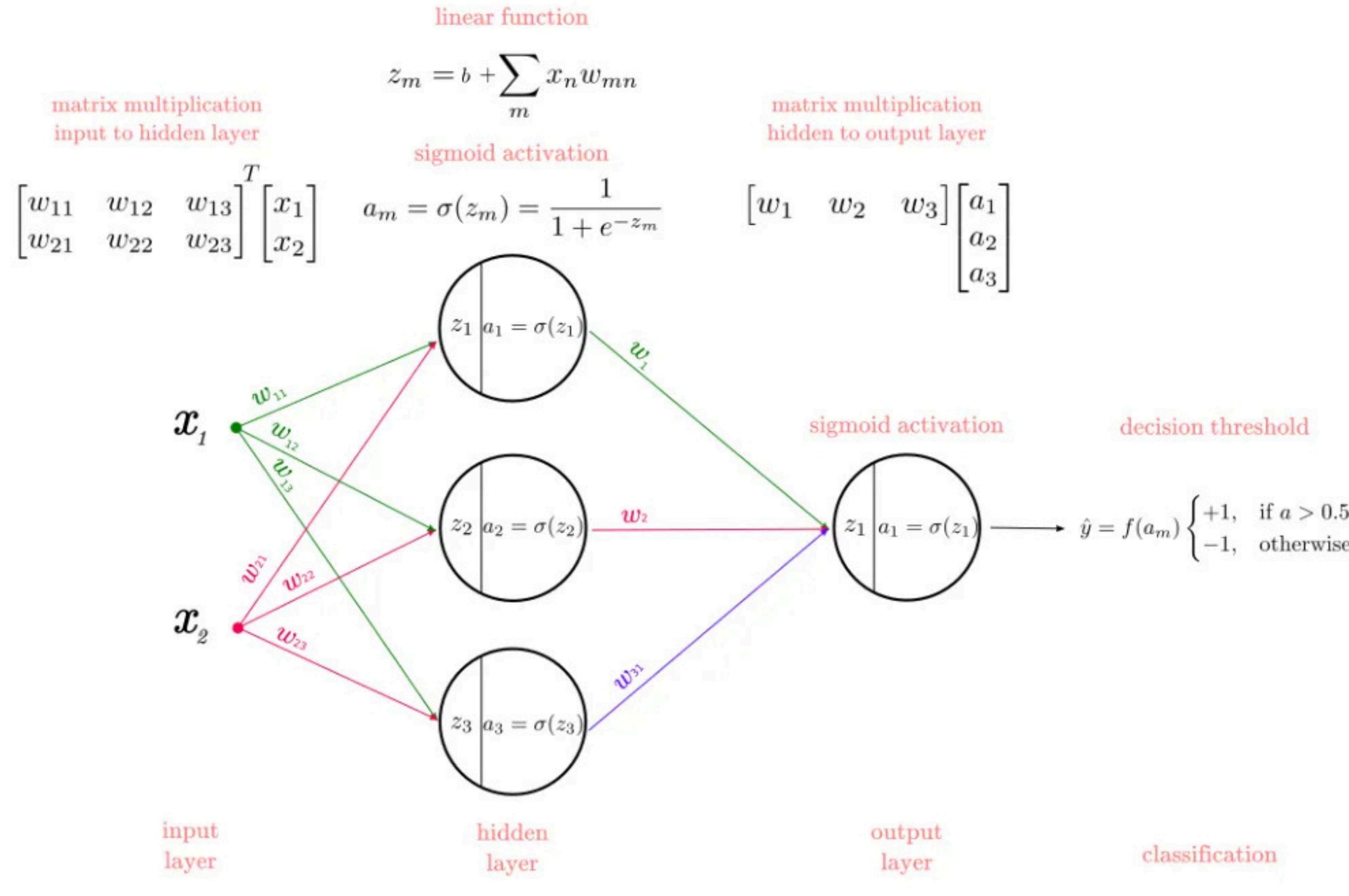


# Forward Propagation

**Forward Propagation (Propagação direta)** é o processo de calcular as saídas do MLP para uma determinada entrada. Envolve o cálculo da soma ponderada das entradas e a aplicação de uma função de ativação em cada camada da rede.



# Forward Propagation



Information flow forward pass neural network with one hidden layer

# Funções de Ativação

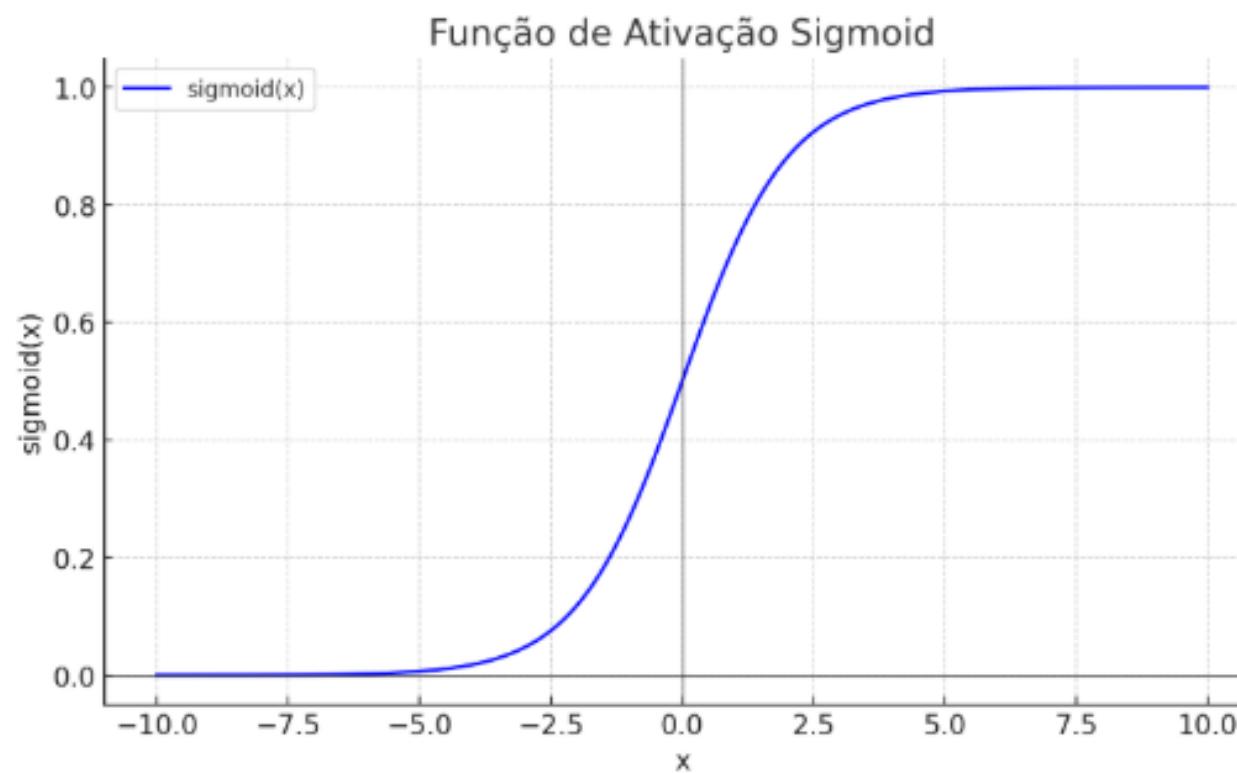
## Função de Ativação Sigmoid

Fórmula matemática:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Características:

- Intervalo de saída:  $(0, 1)$
- Forma da curva: em “S” suave (sigmoidal)
- Valor central:  $\sigma(0) = 0.5$
- Derivada:  $\sigma'(x) = \sigma(x)(1 - \sigma(x))$



# Funções de Ativação

## Função de Ativação Sigmoid

### Vantagens:

- Comporta-se como um "gatilho suave", ativando o neurônio de forma gradual à medida que a entrada aumenta.
- É útil para problemas de classificação binária, pois pode ser interpretada como uma probabilidade.

### Desvantagens:

- **Vanishing gradient:** Derivadas muito pequenas para entradas muito grandes ou muito negativas, dificultando o treinamento (os gradientes “somem”).
- Saída não centrada em zero: Isso pode causar atualizações desbalanceadas nos pesos durante o treinamento.

# Funções de Ativação

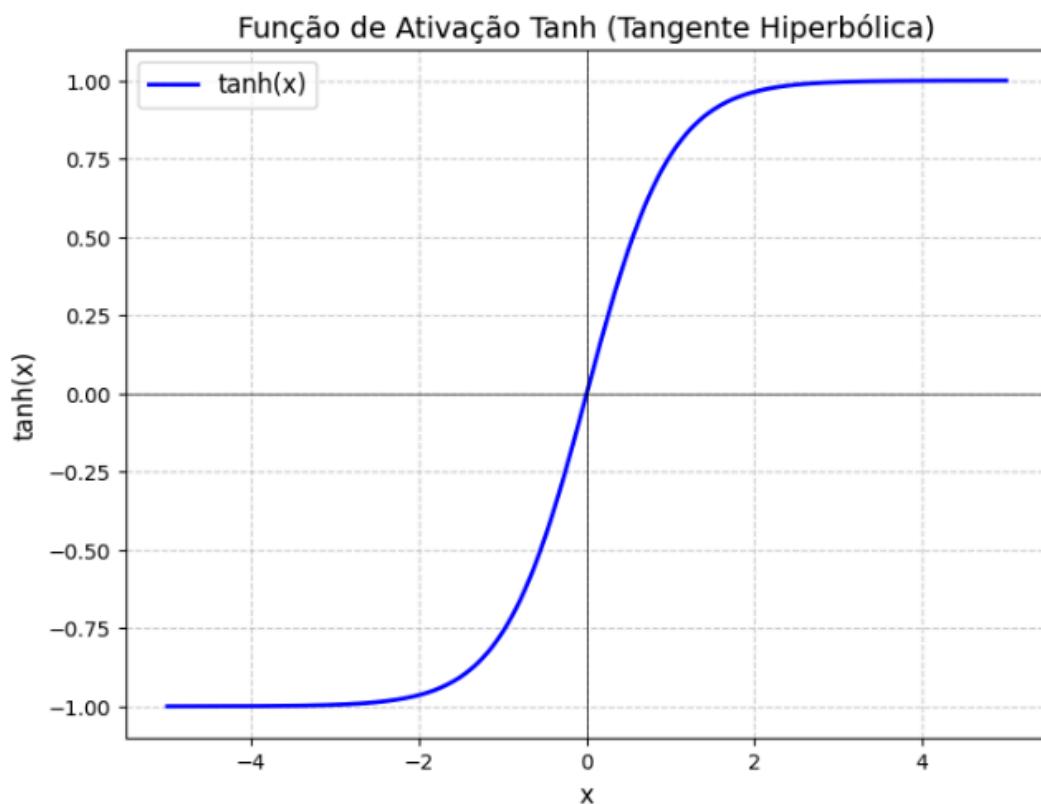
## Função de Ativação tanh

Fórmula matemática:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$$

Características:

- Intervalo de saída:  $(-1, 1)$
- Forma da curva: semelhante à Sigmoid, mas centrada em zero
- Valor central:  $\tanh(0) = 0$
- Derivada:  $\sigma'(x) = \sigma(x)(1 - \sigma(x))$



# Funções de Ativação

## Função de Ativação Tanh

### Vantagens:

- Intervalo de saída:  $[-1, +1]$ , o que ajuda a normalizar os dados e melhora o treinamento (evita viés positivo, comum na sigmoide).
- É uma função ímpar ( $\tanh(-x) = -\tanh(x)$ ), o que ajuda na convergência em certos modelos.

### Desvantagens:

- Assim como a sigmoide, satura nas extremidades (gradientes próximos de zero para  $|x|$  grande), o que pode atrasar ou parar o treinamento em redes profundas.
- Exige operações exponenciais (embora menos crítico com hardware moderno).
- Se uma rede saturar frequentemente em  $-1$  ou  $+1$ , os neurônios podem parar de aprender (problema similar ao da sigmoide).

# Funções de Ativação

## ReLU (Rectified Linear Unit)

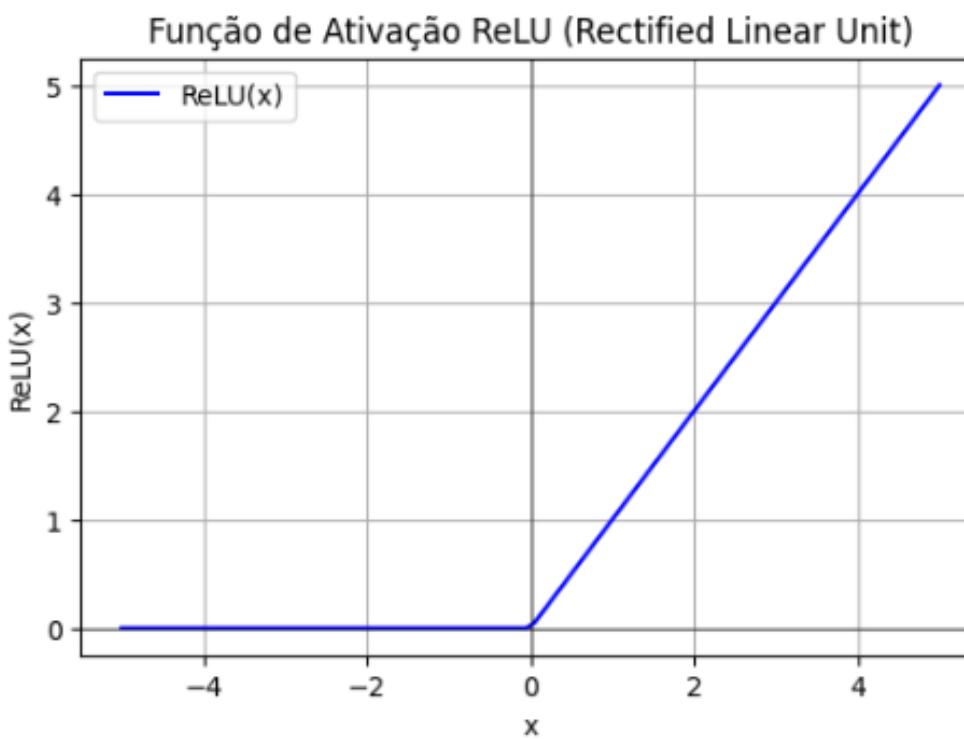
**Definição matemática:**

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{se } x > 0, \\ 0 & \text{se } x \leq 0. \end{cases}$$

**Características:**

- **Intervalo de saída:  $[0, +\infty)$**
- **Valor central: linear para valores positivos, constante zero para valores negativos**
- **Derivada:**

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{se } x > 0, \\ 0 & \text{se } x \leq 0. \end{cases}$$



# Funções de Ativação

## ReLU (Rectified Linear Unit)

### Vantagens:

- Simples e eficiente: Cálculo extremamente rápido, pois envolve apenas uma comparação.
- Não satura para valores positivos: Evita o problema do gradiente desaparecer em grande parte dos casos.
- Acelera a convergência: Em muitos modelos, redes com ReLU convergem mais rapidamente do que com funções sigmoid ou tanh.

### Desvantagens:

- Morte de neurônios: Se muitos neurônios recebem valores negativos, eles podem "morrer" (sempre produzir saída 0), impedindo o aprendizado.
- Não é centrada em zero: Pode introduzir viés nas ativações, dificultando a convergência em alguns casos.

# Funções de Ativação

## ELU (Exponential Linear Unit)

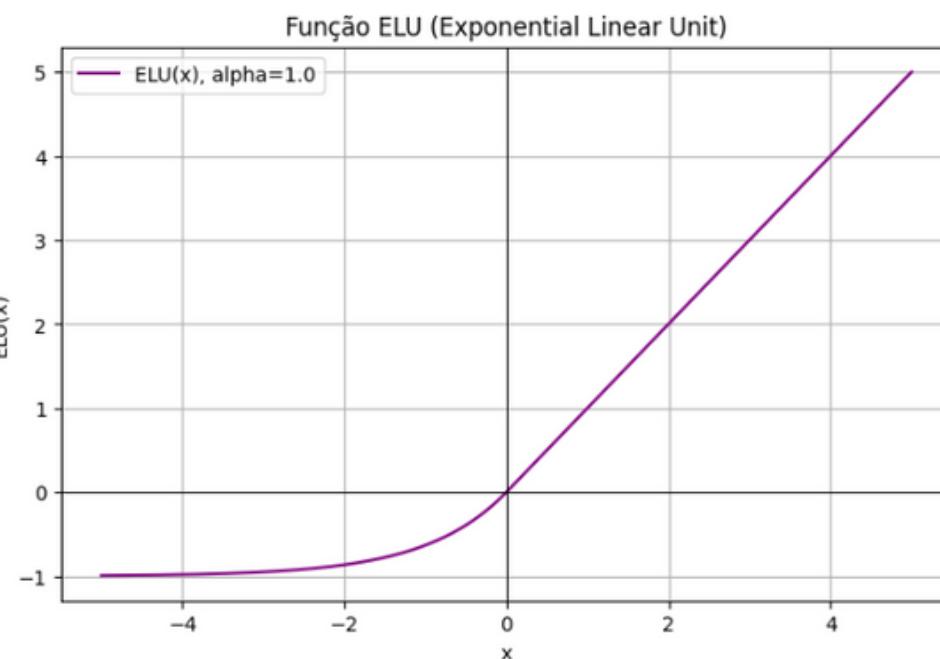
**Definição matemática:**

$$\text{ELU}(x) = \begin{cases} x & \text{se } x > 0 \\ \alpha(e^x - 1) & \text{se } x \leq 0 \end{cases}$$

**Características:**

- **Intervalo de saída:**  $(-\alpha, +\infty)$
- **Valor central:** Em  $x = 0$ ,  $\text{ELU}(0) = 0$
- **Derivada:**

$$\frac{d}{dx} \text{ELU}(x) = \begin{cases} 1 & \text{se } x > 0 \\ \text{ELU}(x) + \alpha & \text{se } x \leq 0 \end{cases}$$



# Funções de Ativação

## ELU (Exponential Linear Unit)

### Vantagens:

- **Evita o problema do “morte de neurônios”:** Ao contrário da ReLU, ELU permite valores negativos (suavizados), reduzindo o número de neurônios que param de aprender. Não satura para valores positivos: Evita o problema do gradiente desaparecer em grande parte dos casos.
- **Saída centrada em zero:** Isso pode acelerar o treinamento por reduzir o viés durante o aprendizado.

### Desvantagens:

- **Mais custosa computacionalmente:** Envolve o cálculo de exponenciais, o que é mais pesado que a ReLU.
- **Escolha do parâmetro  $\alpha$ :** Requer ajuste de um parâmetro hiperparamétrico que controla a curva para valores negativos.
- **Possível saturação:** Para entradas negativas muito pequenas, a função pode saturar (derivadas próximas de zero), o que pode prejudicar o aprendizado.

# Funções de Ativação

## Softmax

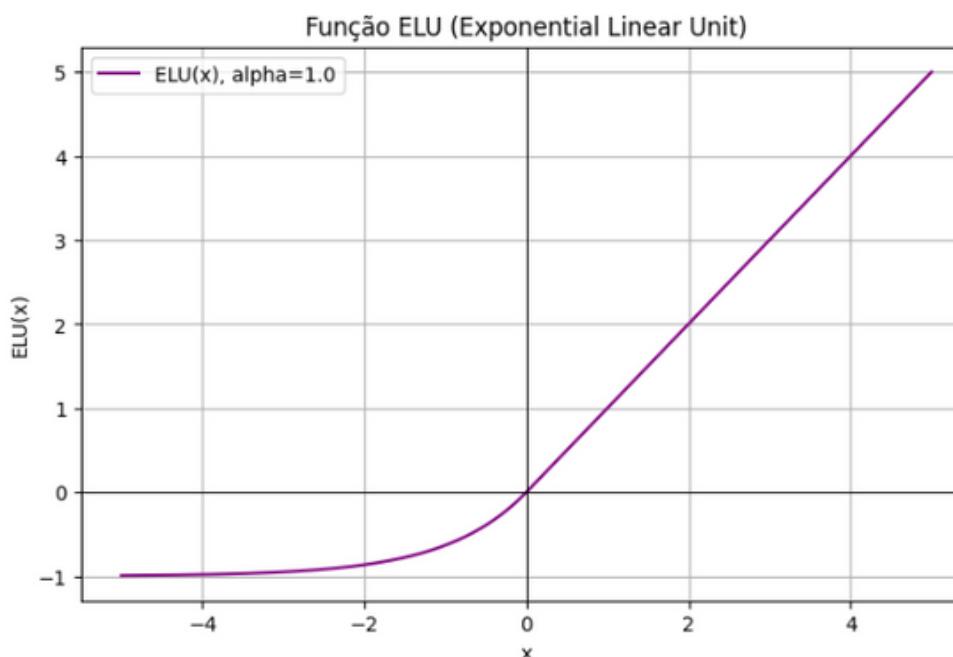
**Definição matemática:**

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

**Características:**

- **Intervalo de saída:** vetor  $\mathbf{y} \in [0, 1]^K$ , com  $\sum_{i=1}^K y_i = 1$ .
- **Valor central:** não se aplica diretamente, pois depende do vetor de entrada completo.
- **Derivada:**

$$\frac{\partial y_i}{\partial z_j} = y_i(\delta_{ij} - y_j)$$



# Funções de Ativação

## Softmax

### Vantagens:

- **Probabilística:** Transforma as saídas em uma distribuição de probabilidade, útil para tarefas de classificação multi-classe.
- **Interpretação clara:** Cada saída pode ser interpretada como a “confiança” do modelo naquela classe.
- **Diferenciável:** Funciona bem com backpropagation e permite otimizar funções de perda como entropia cruzada.

### Desvantagens:

- **Não usada em camadas ocultas:** É geralmente utilizada apenas na última camada de redes neurais classificadoras.
- **Sensível a outliers:** Um valor de entrada muito maior que os outros pode dominar a distribuição (efeito exponencial).
- **Pode causar saturação:** Se as diferenças entre os valores forem grandes, os gradientes podem se tornar muito pequenos, dificultando o treinamento (problema de vanishing gradient).

# Forward Propagation

1. Entrada:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.1 \end{bmatrix}$$

2. Pesos e bias da camada de entrada para a oculta:

$$W_{\text{input-hidden}} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} 0.5 & -0.2 & 0.1 \\ 0.3 & 0.8 & -0.5 \end{bmatrix} \quad \mathbf{b}_{\text{hidden}} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

3. Cálculo da ativação da camada oculta:

$$z = W^T \cdot x + b \Rightarrow z = \begin{bmatrix} 0.5 & 0.3 \\ -0.2 & 0.8 \\ 0.1 & -0.5 \end{bmatrix} \cdot \begin{bmatrix} 0.6 \\ 0.1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

$$z_1 = 0.5 * 0.6 + 0.3 * 0.1 + 0.1 = 0.3 + 0.03 + 0.1 = 0.43$$

$$z_2 = -0.2 * 0.6 + 0.8 * 0.1 + 0.1 = -0.12 + 0.08 + 0.1 = 0.06$$

$$z_3 = 0.1 * 0.6 - 0.5 * 0.1 + 0.1 = 0.06 - 0.05 + 0.1 = 0.11$$

Aplicamos a função sigmoide:

$$a_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}$$

$$a_1 = \sigma(0.43) \approx 0.605$$

$$a_2 = \sigma(0.06) \approx 0.515$$

$$a_3 = \sigma(0.11) \approx 0.527$$

# Forward Propagation

4. Pesos e bias da camada oculta para a saída:

$$W_{\text{hidden-output}} = [0.7 \quad -0.6 \quad 0.2] \quad b = 0.05$$

5. Cálculo da ativação da camada oculta:

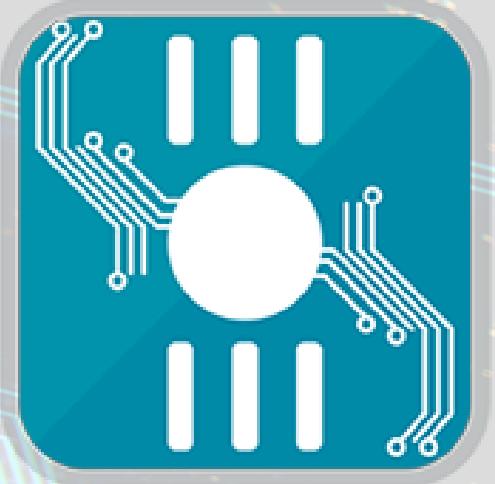
$$z = W \cdot a + b = 0.7 * 0.605 + (-0.6) * 0.515 + 0.2 * 0.527 + 0.05$$

$$z = 0.4235 - 0.309 + 0.1054 + 0.05 = 0.2699$$

Aplicamos a **sigmoide**:

$$a = \sigma(0.2699) \approx 0.567 \Rightarrow y = \begin{cases} 1 & \text{se } a \geq 0.5 \\ 0 & \text{caso contrário} \end{cases} \Rightarrow y = 1$$

# IMPLEMENTAÇÃO



# ENGENHARIA DA COMPUTAÇÃO

---

UNIVERSIDADE FEDERAL DO MARANHÃO

# Obrigado!