

# Redes Neurais Artificiais

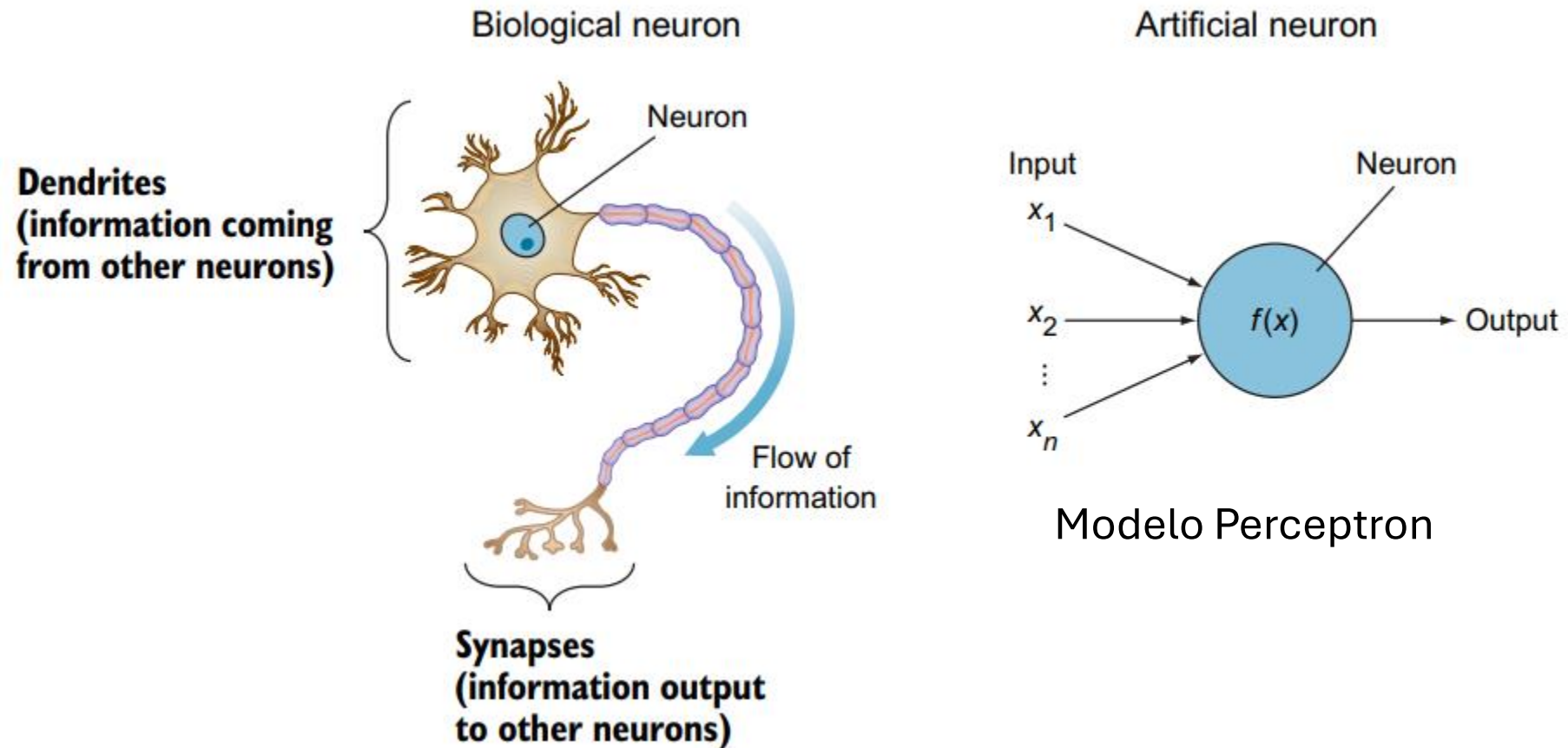
ESPECIALIZAÇÃO EM ANÁLISE E CIÊNCIA DE DADOS



Vinícius Rodrigues Oviedo

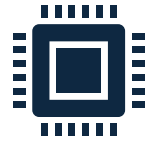
Santa Maria, 2024

# O que é rede neural



ELGENDY, Mohamed. **Deep Learning for Vision Systems**. Shelter Island: Manning Publications, 2020.

# PLAYGROUND DO TENSORFLOW



<https://playground.tensorflow.org/>

# Tipos de redes neurais

## **Perceptron:**

- O tipo mais simples de rede neural (1 única camada/nó/neurônio) ;
- Utilizado principalmente para problemas de classificação linear.

## **Redes Neurais Feedforward (FNN):**

- Redes onde os dados fluem em uma única direção (entrada → saída) sem ciclos ou *loops*;
- Incluem Perceptrons Multicamadas (MLPs).

**Redes Neurais Convolucionais (CNN):** processamento de imagens e reconhecimento de padrões.

**Redes Neurais Recorrentes (RNN):** projetadas para processar sequências de dados, como séries temporais ou texto.

**Long Short-Term Memory (LSTM):** variação da RNN que resolve problemas de dependência de longo prazo.

**Redes Neurais de Memória a Curto Prazo (GRU):** outra variação de RNN com uma arquitetura mais simples e menos parâmetros.

# Tipos de redes neurais

## **Redes Neurais de Base Radial (RBF):**

- Utilizam funções de base radial como funções de ativação;
- Eficazes em problemas de classificação e regressão onde os dados têm um padrão radial.

## **Redes Neurais Generativas Adversárias (GAN):**

- constituídas por duas redes (o gerador e o discriminador) que competem entre si.
- Utilizadas para gerar dados sintéticos, como imagens e textos realistas.

## **Autoencoders:**

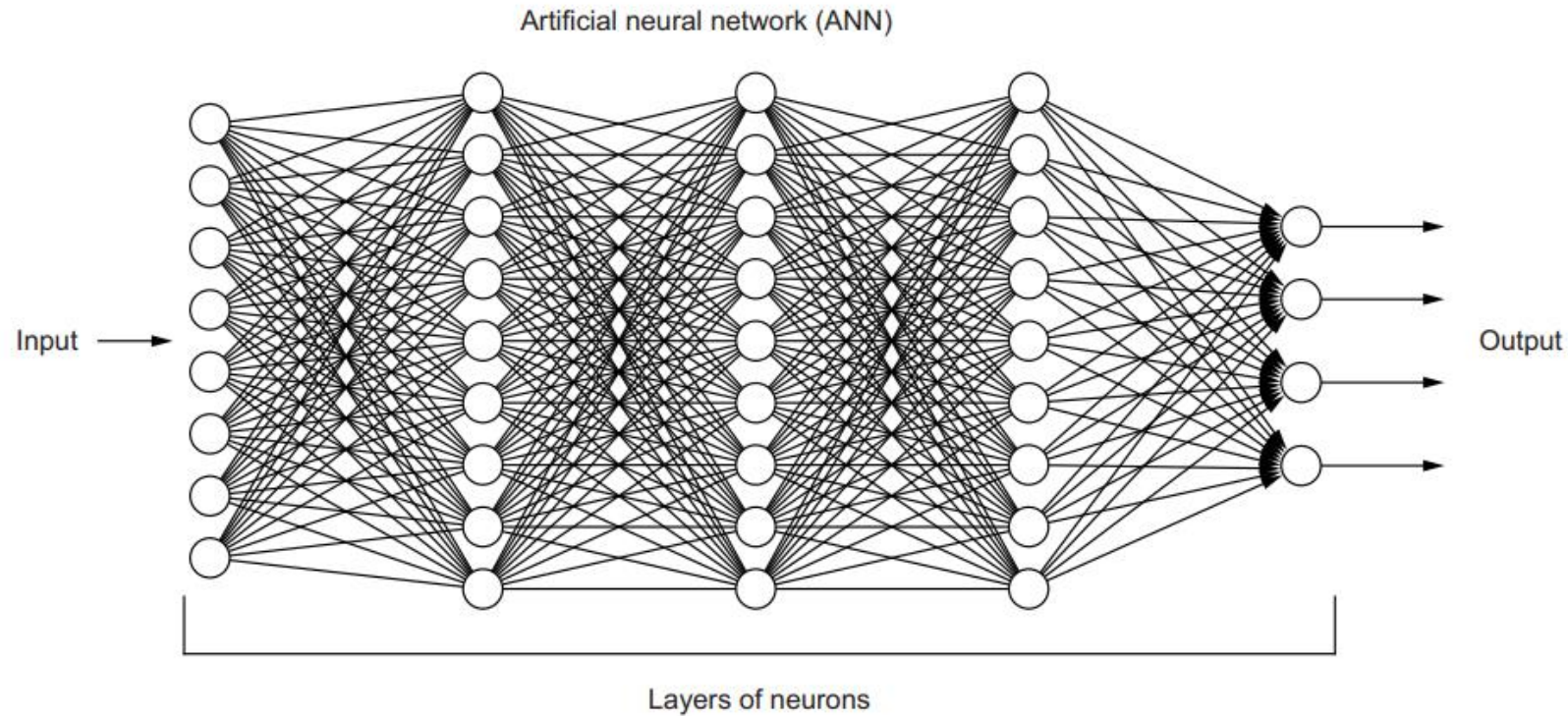
- Projetadas para aprender representações eficientes dos dados;
- Frequentemente usadas para redução de dimensionalidade ou detecção de anomalias.

## **Transformers:**

Recentemente populares em processamento de linguagem natural (NLP);

Utilizam mecanismos de atenção para capturar relações entre diferentes partes de uma sequência de dados.

# Perceptron multicamada

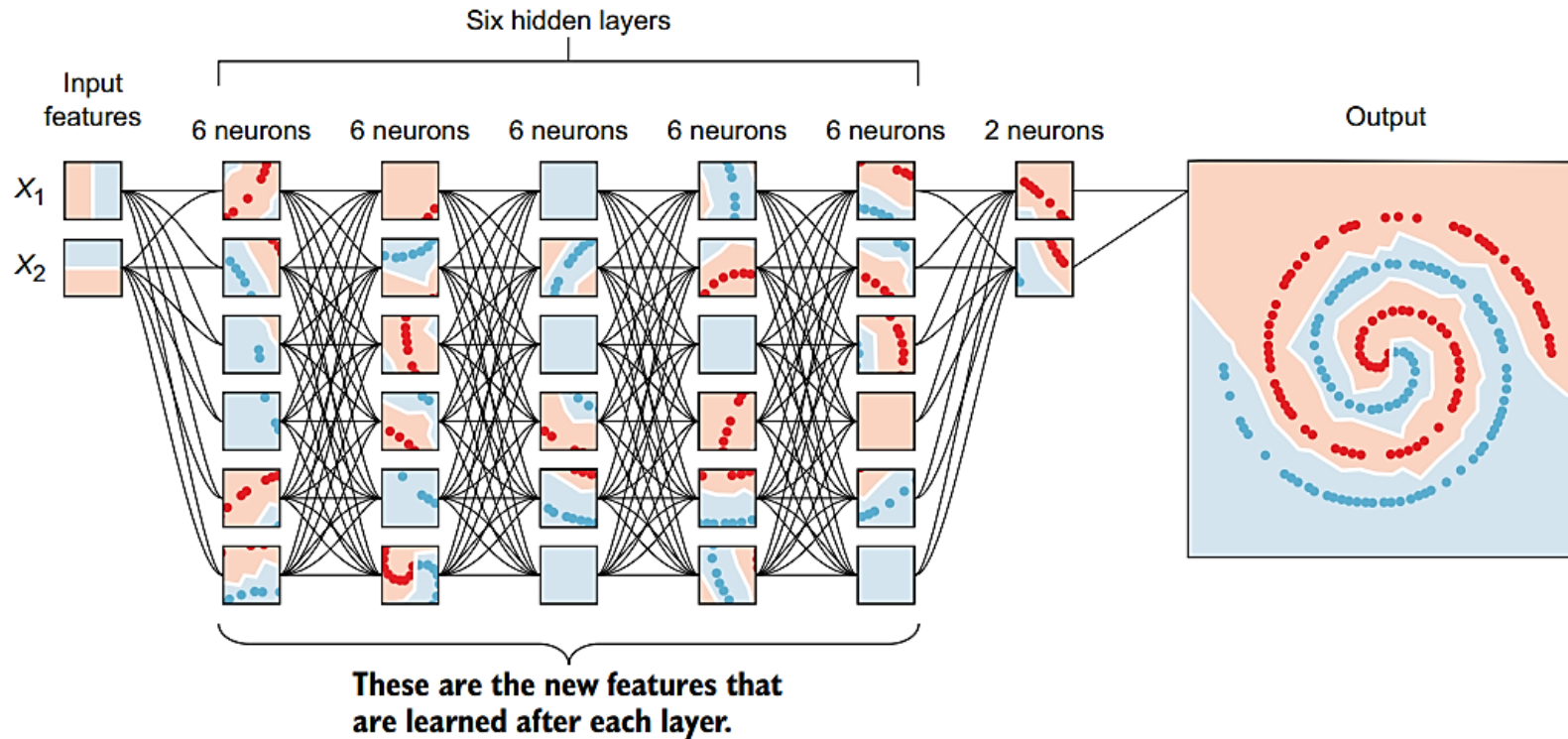


**Figure 2.2** An artificial neural network consists of layers of nodes, or neurons connected with edges.

ELGENDY, Mohamed. **Deep Learning for Vision Systems**. Shelter Island: Manning Publications, 2020.

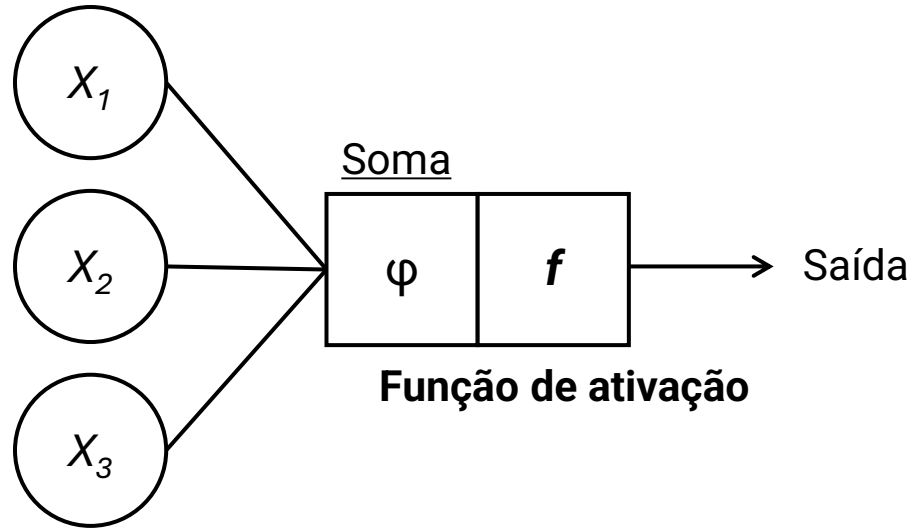
# Perceptron multicamada

**Reconhecimento de padrões:** padrões mais simples nas primeiras camadas e mais complexos nas finais



ELGENDY, Mohamed. **Deep Learning for Vision Systems**. Shelter Island: Manning Publications, 2020.

# Funcionamento da rede



Onde,

$\varphi$  = função de transferência/soma;

$f$  = função de ativação

$x_n$  = enésimo nó da rede;

$w_i$  = peso associado ao nó  $x_n$ ;

$b_k$  = bias (parâmetro de incerteza/erro associado)

Função de soma:

$$\varphi = x_1 \cdot w_2 + x_2 \cdot w_1 + x_3 \cdot w_3$$

**Função identidade:**

$$f(\varphi) = \varphi$$

**Função ReLU:**

$$f(\varphi) = (0, x_{máx})$$

**Função sigmoide:**

$$f(\varphi) = \frac{1}{1 + e^{-\varphi}}$$

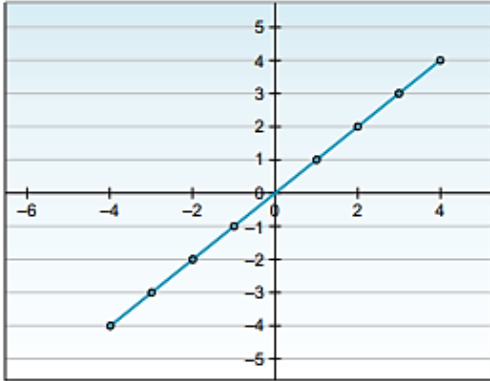
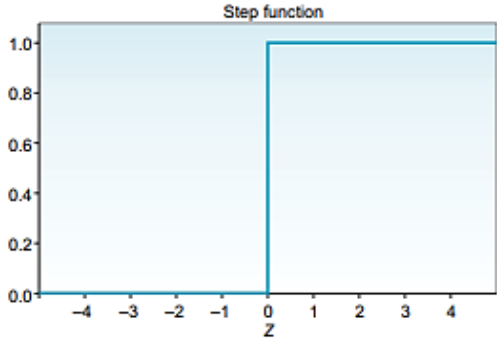
**Função tangente hiperbólica:**

$$f(\varphi) = \frac{2}{1 + e^{-2\varphi}} - 1$$



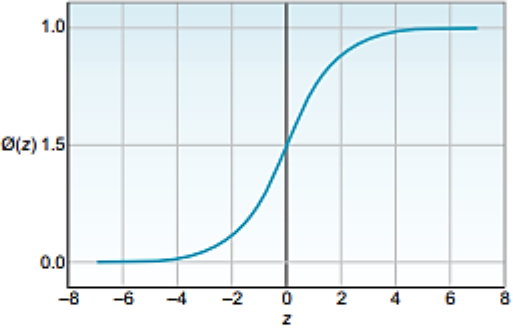
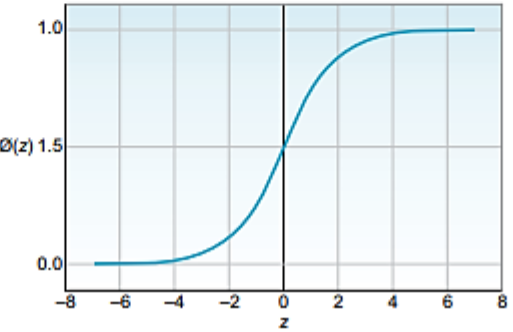
# Funções de ativação

Table 2.1 A cheat sheet of the most common activation functions

Activation function	Description	Plot	Equation
Linear transfer function (identity function)	The signal passes through it unchanged. It remains a linear function. Almost never used.		$f(x) = x$
Heaviside step function (binary classifier)	Produces a binary output of 0 or 1. Mainly used in binary classification to give a discrete value.		$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$

# Funções de ativação

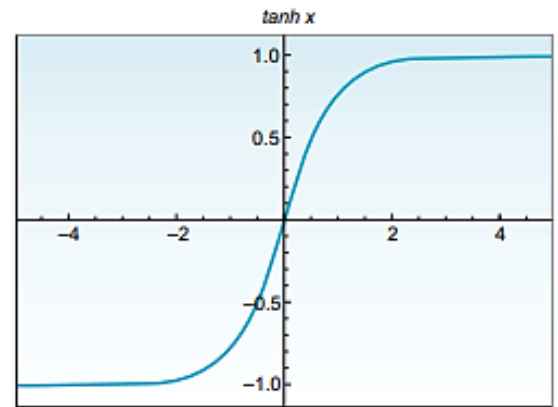
Table 2.1 A cheat sheet of the most common activation functions

Activation function	Description	Plot	Equation
Sigmoid/ logistic function	Squishes all the values to a probability between 0 and 1, which reduces extreme values or outliers in the data. Usually used to classify two classes.		$\sigma(z) = \frac{1}{1 + e^{-z}}$
Softmax function	A generalization of the sigmoid function. Used to obtain classification probabilities when we have more than two classes.		$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$

# Funções de ativação

Hyperbolic  
tangent func-  
tion (tanh)

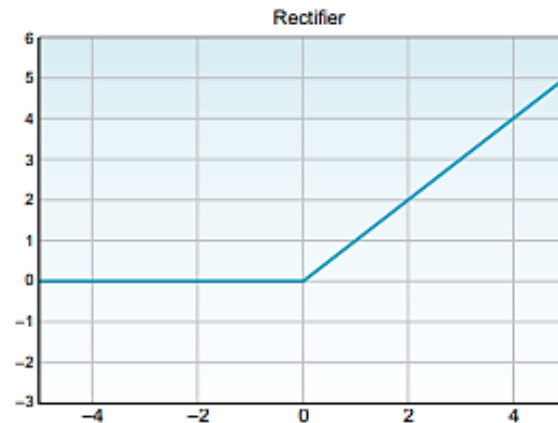
Squishes all values to the range of -1 to 1. Tanh almost always works better than the sigmoid function in hidden layers.



$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$
$$= \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Rectified  
linear unit  
(ReLU)

Activates a node only if the input is above zero. Always recommended for hidden layers. Better than tanh.

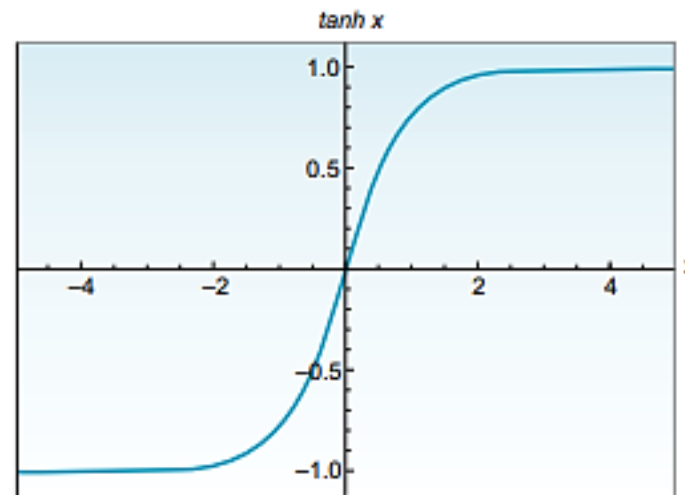


$$f(x) = \max(0, x)$$

# Funções de ativação

Hyperbolic  
tangent func-  
tion (tanh)

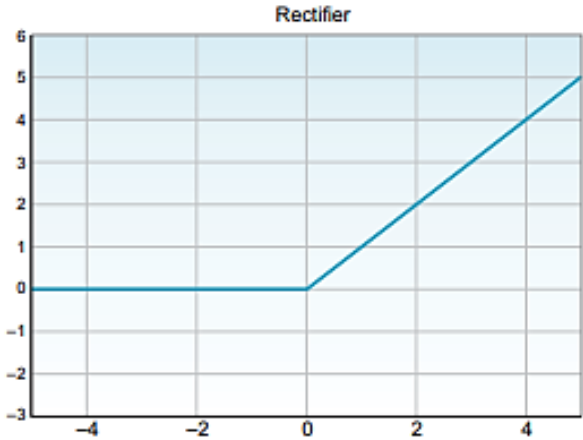
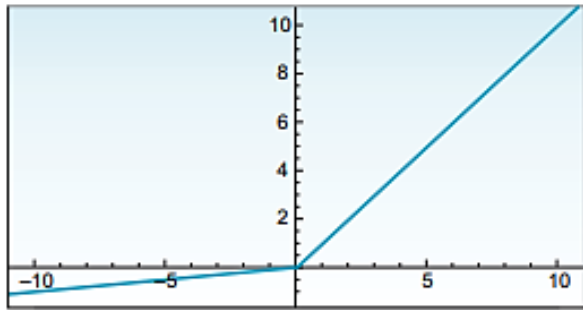
Squishes all values  
to the range of -1  
to 1. Tanh almost  
always works better  
than the sigmoid  
function in hidden  
layers.



$$\begin{aligned}\tanh(x) &= \frac{\sinh(x)}{\cosh(x)} \\ &= \frac{e^x - e^{-x}}{e^x + e^{-x}}\end{aligned}$$

ELGENDY, Mohamed. **Deep Learning for Vision Systems**. Shelter Island: Manning Publications, 2020.

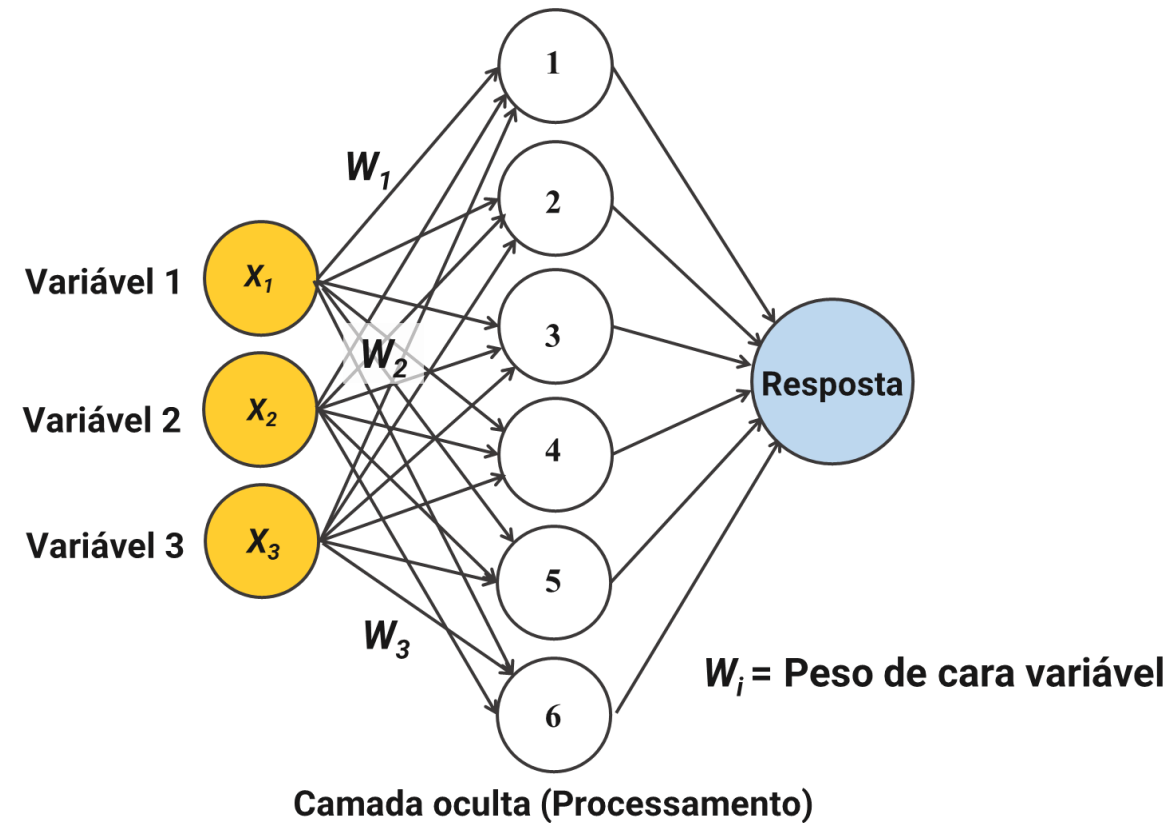
# Funções de ativação

Rectified linear unit (ReLU)	Activates a node only if the input is above zero. Always recommended for hidden layers. Better than tanh.		$f(x) = \max(0, x)$
Leaky ReLU	Instead of having the function be zero when $x < 0$ , leaky ReLU introduces a small negative slope (around 0.01) when $(x)$ is negative.		$f(x) = \max(0.01x, x)$

# Arquitetura da Rede Neural Artificial (RNA)

## Estrutura da rede

- ❑ Dados de entrada : Camada oculta : Resposta
- ❑ Pode ter 1 ou mais camadas ocultas
- ❑ Ajuste dos parâmetros do algoritmo:
  - Taxa de aprendizagem (%) (**learning rate**)
  - Função de ativação: indica se o neurônio é ativado ou não
  - Função de otimização dos pesos (solver)  
Verifica a contribuição de cada variável na predição  
Adam, SGD, RMSProp, Adamax, Adamgrad...



# HIPER-PARÂMETROS

- Função de ativação\*

ReLu, identity, tanh (regressão)

Sigmoid, softmax (classificação)

- Função de otimização dos pesos

- Número de camadas ocultas

- Número de neurônios na(s) camada(s) oculta(s)

- Batch size: geralmente múltiplos de 2

(2, 4, 6, 8, 16, 32, 64, 128)

- Regularização (L1 e L2)

- Dropout

- Early stopping (paciência para tolerar erros)

ReLu, sigmoid e tanh funcionam melhor para camadas ocultas

\* Tanh funciona um pouco melhor que sigmoid para camadas ocultas

**Métricas para regressão:**  $R^2$ , MSE, RMSE, MAE, MAPE...

**Métricas para classificação:** acurácia, precisão, Gini score, cross-entropy...

**Função de perda:** Val\_loss e loss podem utilizar a mesma função objetiva (métrica).

**Métrica mais comum:**  $R^2$  | RMSE

# Performance do algoritmo e função de perda

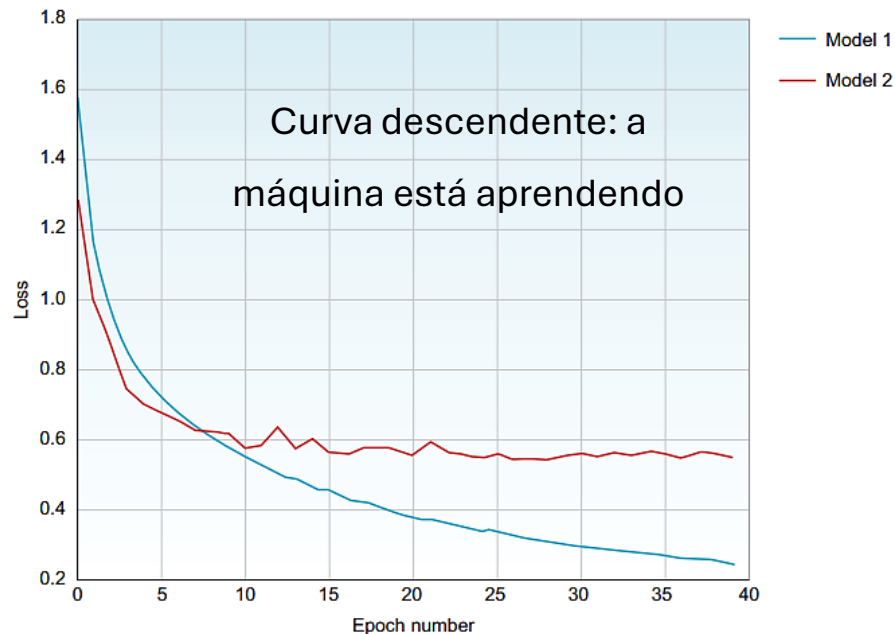


Figure 2.24 A visualization of the loss functions of two separate models plotted over time

O **modelo #1** está fazendo um trabalho melhor em minimizar o erro, enquanto o **modelo #2** começa melhor até 6 *epochs* e depois estabiliza

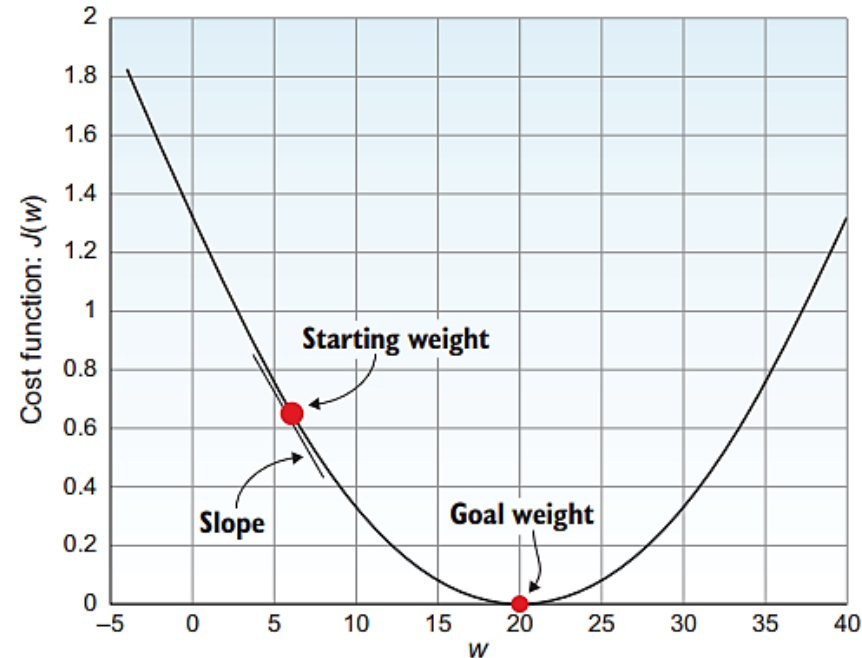


Figure 2.26 The error function with respect to its weight for a single perceptron is a 2D curve.

## Seleção de função de perda (*loss function*):

- MSE para regressão
- Cross-entropy para classificação



# Bibliotecas

**Tensorflow** → biblioteca de código aberto desenvolvida pelo Google para computação numérica e aprendizado de máquina.

Amplamente utilizada para construir, treinar e implantar modelos de aprendizado profundo (deep learning).

**Keras** → uma API de alto nível para construção e treinamento de modelos de aprendizado profundo. Inicialmente independente, agora é integrada ao TensorFlow como seu *frontend* padrão.



# Por que importar TensorFlow antes do Keras?

**Integração:** Keras é integrada como parte do TensorFlow desde a versão 2.0, sendo acessada através do módulo `tensorflow.keras`. Isso garante uma compatibilidade total entre as funcionalidades de ambas as bibliotecas.

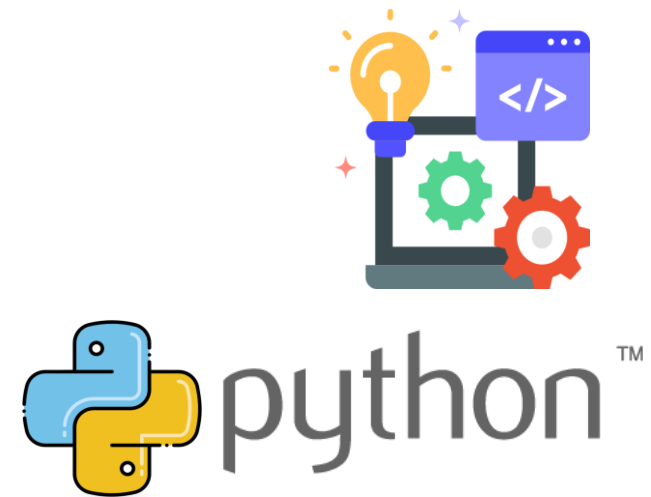
**Configuração:** Importar TensorFlow primeiro assegura que todas as configurações e inicializações necessárias sejam feitas antes de Keras, prevenindo potenciais conflitos ou erros.

**Desempenho:**

- O TensorFlow configura o *backend* computacional que Keras utiliza.
- Importá-lo primeiro garante que a alocação de recursos e otimizações de desempenho sejam feitas corretamente.

# Aplicações

- Problemas de regressão
- Problemas de classificação
- *Clustering*
- Processamento de linguagem natural (NLP)
- Processamento de imagem
- Reconhecimento de padrões
- Geração de Dados (Redes Generativas)
- Redução de Dimensionalidade
- Controle e robótica
- Outros



# Exemplo prático 1

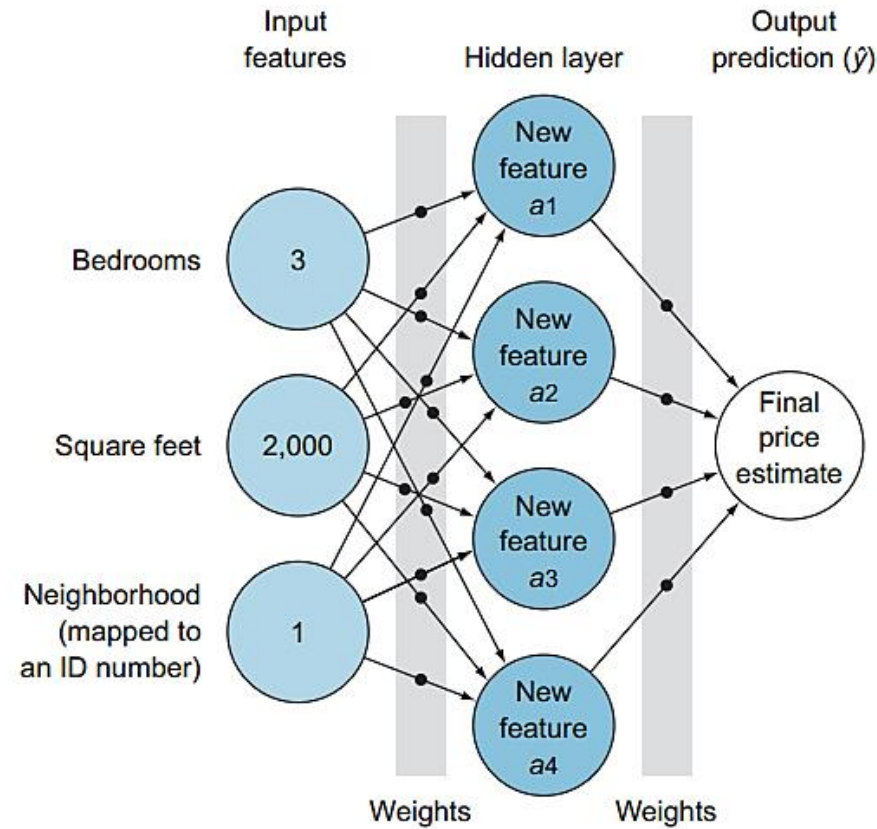


Figure 2.22 A small neural network to estimate the price of a house based on three features: how many bedrooms it has, how big it is, and which neighborhood it is in

# Exemplo prático 2

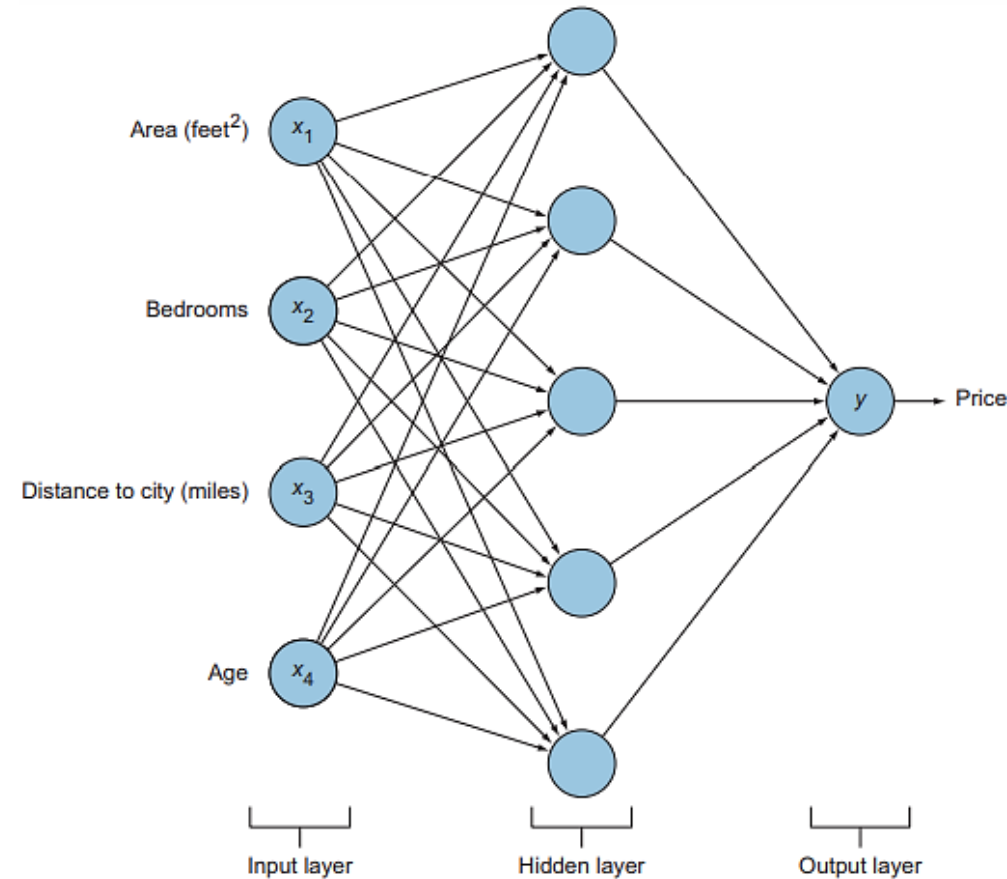
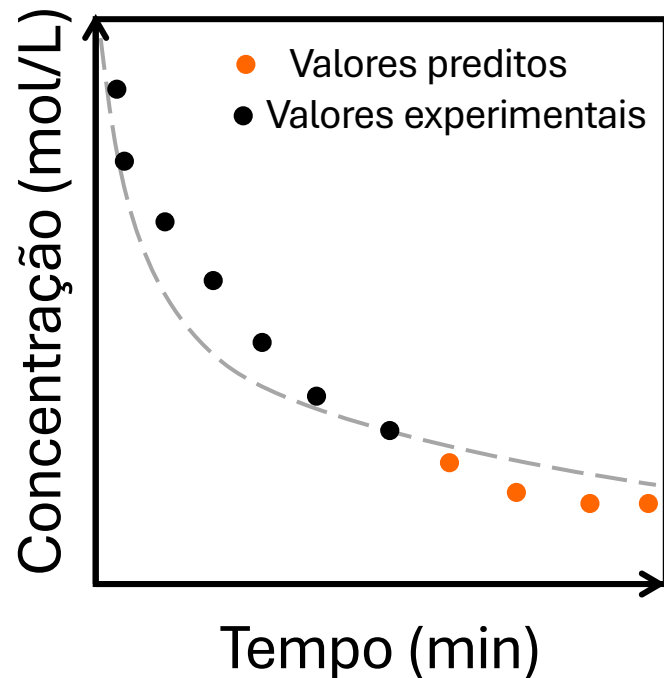


Figure 2.28 If we want to predict house prices based on only four features (inputs) and one hidden layer of five neurons, we'll have 20 edges (weights) from the input to the hidden layer, plus 5 weights from the hidden layer to the output prediction.

# Exemplo prático 3



**Temática:** Um tratamento avançado de água foi realizado em escala de bancada (laboratório) para o tratamento de uma água contaminada por corante cancerígeno.

O ensaio durou 180 min (3 h), atingindo **50% de remoção** do corante da água utilizando um catalisador alternativo.

Deseja-se saber se **após o tempo do ensaio ( $t > 3$  h)**, o catalisador **removerá mais** do corante da água ou se **permanecerá em 50%** de remoção.

# Implementação (Google colab)

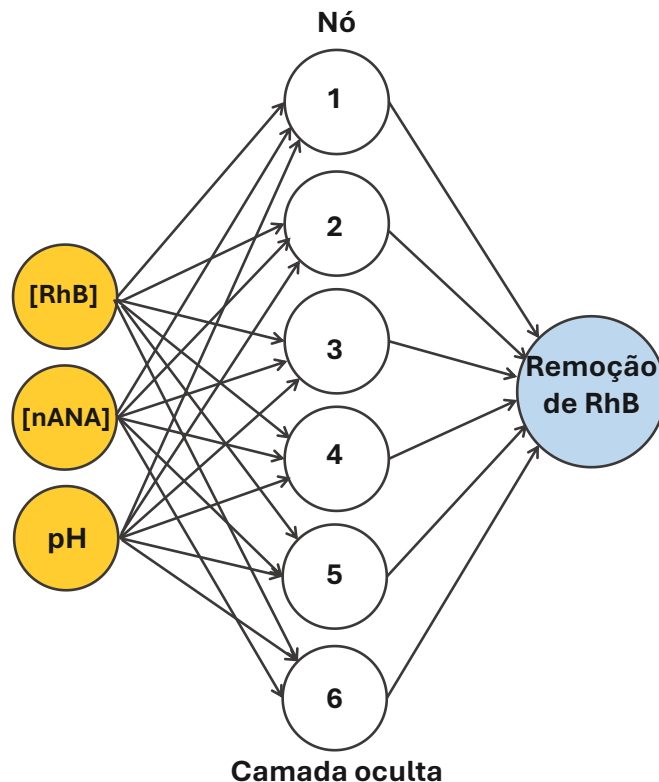
```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense

# X = seleção das variáveis de entrada
x = data.drop(['Abs'], axis=1)  #(Retira colunas não utilizadas no modelo)
y = data['Abs'] # Variável resposta

xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=42)
xtrain.shape,xtest.shape,ytrain.shape,ytest.shape
```

# Implementação (Google colab)

```
# Criação do modelo
model = Sequential()
model.add(Dense(6, input_dim=3, activation='relu', kernel_initializer='uniform'))
model.add(Dense(1))
```



Arquitetura da rede neural criada → 3:6:1

**3 neurônios** de entrada

**1** camada oculta com **6 neurônios**

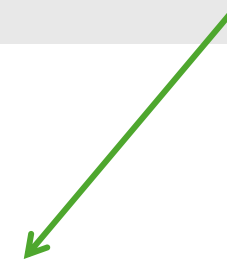
**1 neurônio** de saída



# Implementação (Google colab)

```
# Compilação do modelo
model.compile(optimizer='SGD', loss='mse', metrics=['mse'])

# Definição do EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=4, verbose=0,
restore_best_weights=True)
```



**Verbose = 0** (Modo silencioso. Nenhuma saída será exibida durante o treinamento)

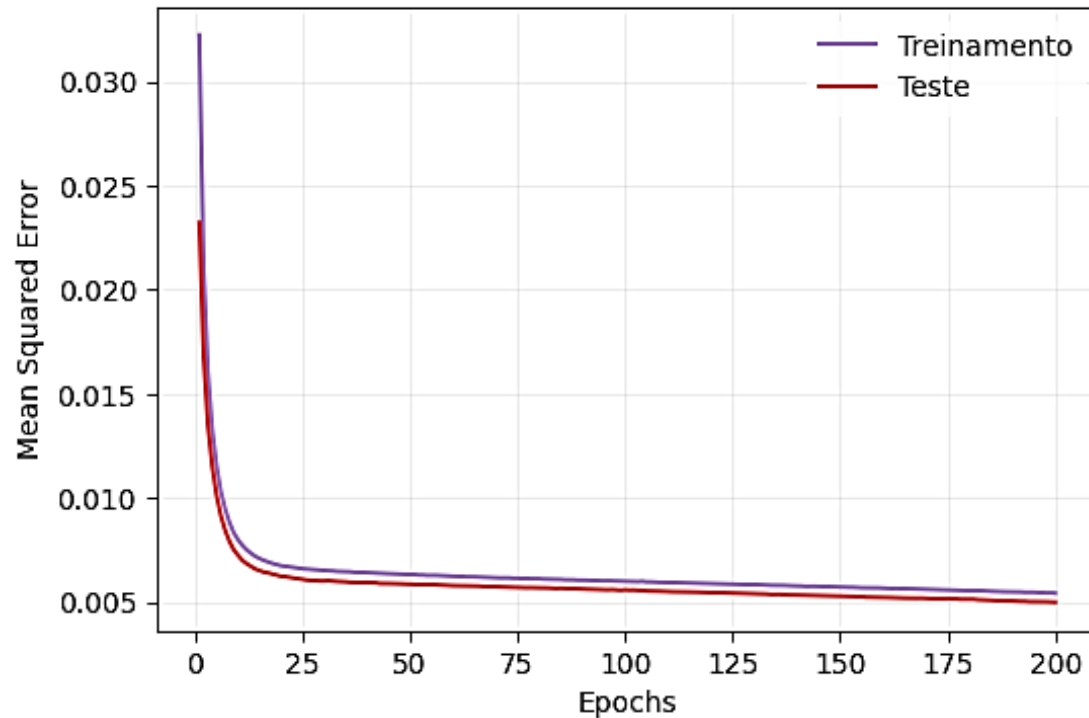
**Verbose = 1** (Barra de progresso. Uma barra de progresso será exibida durante o treinamento)

**Verbose = 2** (Uma linha por época. Para cada época, uma única linha de saída será exibida.)

# Output

Curvas muito próximas: quanto mais próximas (e decrescente), melhor!

→ Bom aprendizagem e boa generalização



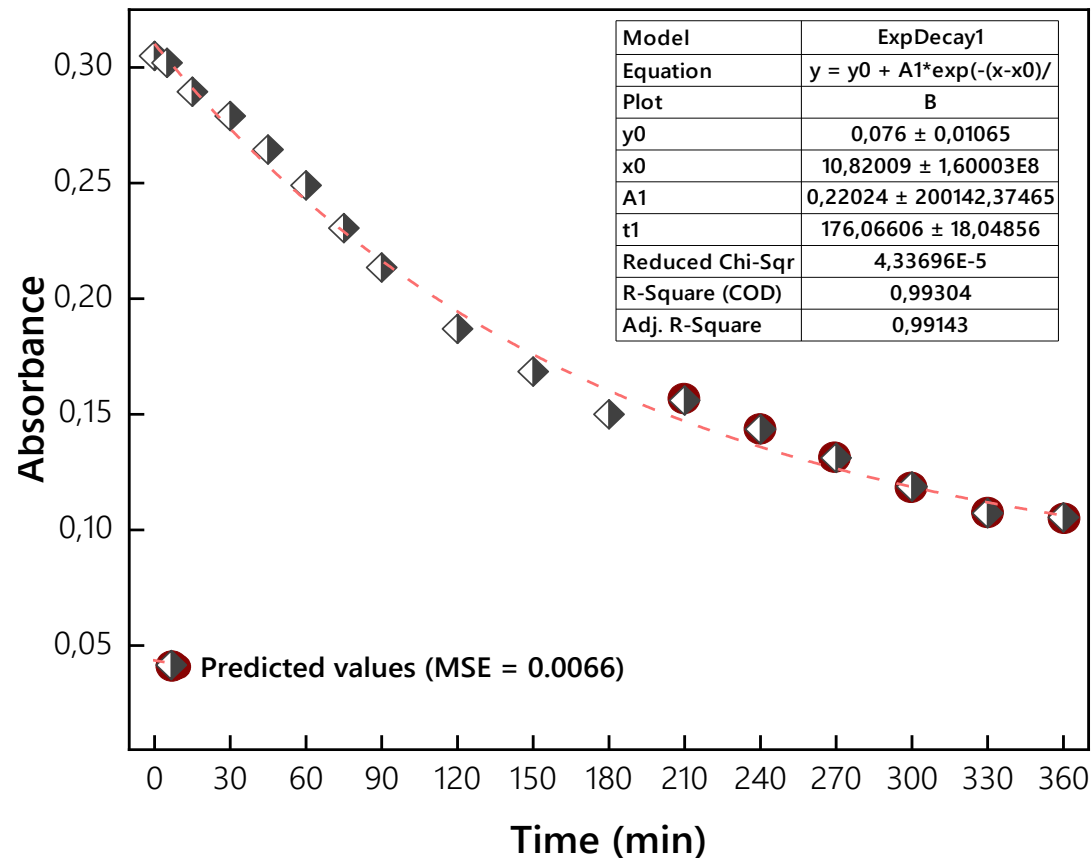
14/14 [=====] - 0s 1ms/step

4/4 [=====] - 0s 3ms/step

Batch Size: 32, Train  $R^2$ : 0.6920367333755781, Test  $R^2$ : 0.6905011661650048, Last MSE: 0.005432787351310253, Last Val MSE: 0.004999096971005201

- A **curva de decaimento** indica que a máquina está aprendendo com os dados com minimização do erro.
- O erro estabiliza em aproximadamente **27 epochs**.
- $R^2$  e MSE muito próximos para treino e teste.
- $RMSE_{\text{treino}} = 0,073$  |  $RMSE_{\text{teste}} = 0,070$
- Verificar possibilidade de haver *overffiting*.

# Resultado da predição e interpretação física



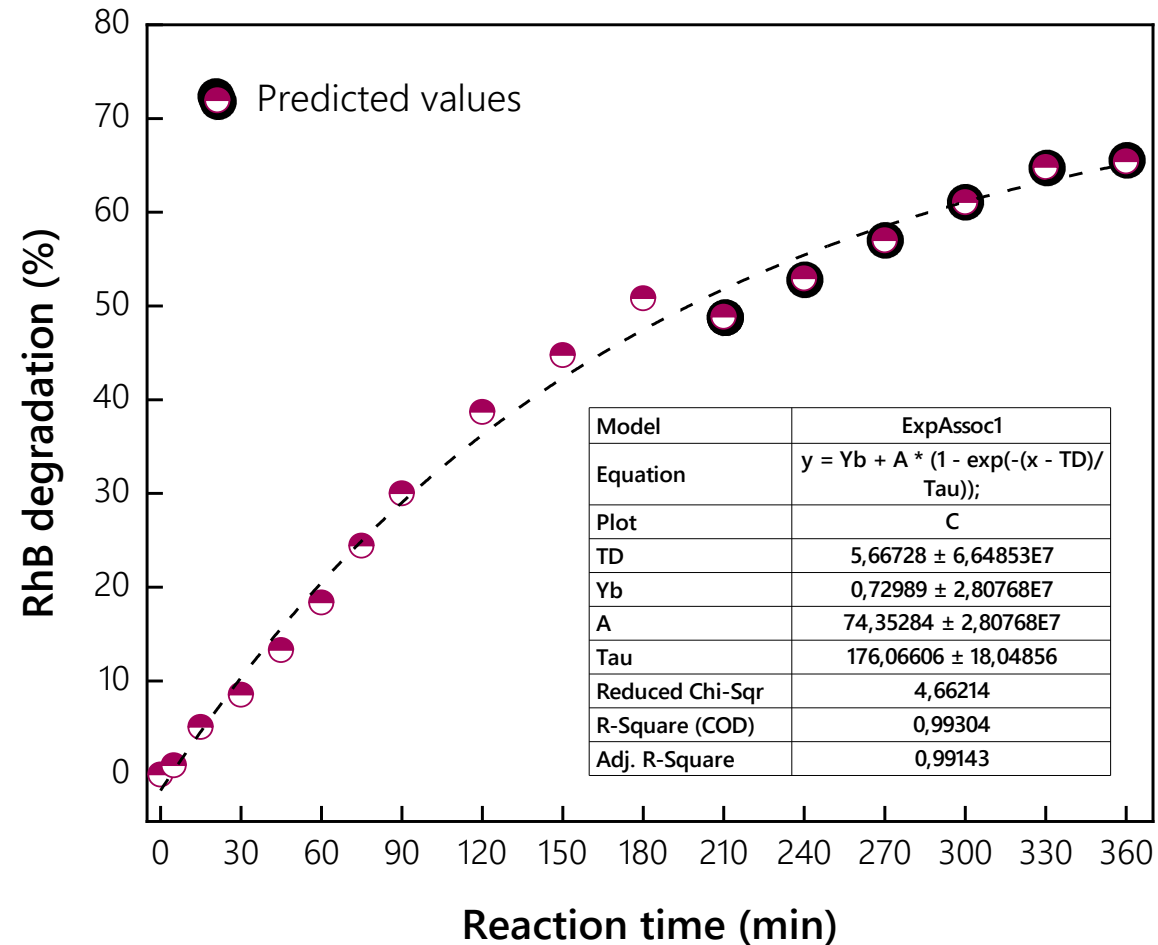
Após 180 min (**3 h**) → A remoção do corante continua

Em 360 min (**6 h**) → A reação química ainda está ocorrendo

*Tempo de operação mínimo: **6 horas**.*

- E após 8 horas, que resultado teríamos?
- Paramos o processo em 6 h ou damos continuidade?
- Qual o tempo mínimo para parar o processo e fazer manutenção do reator?

# Resultado da predição e interpretação física



Cerca de **70% de remoção** do corante rodamina B é atingido em 6 h