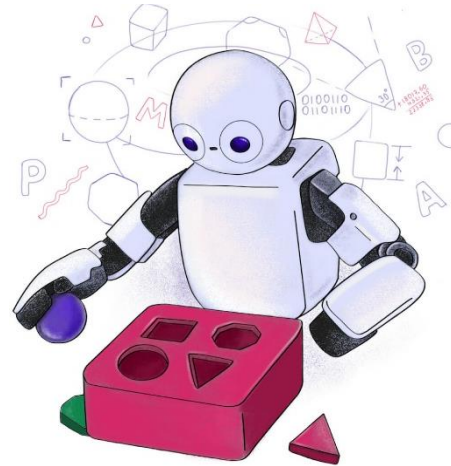


# TP557 - Tópicos avançados em IoT e Machine Learning: *Regressão com DNNs (Parte II)*



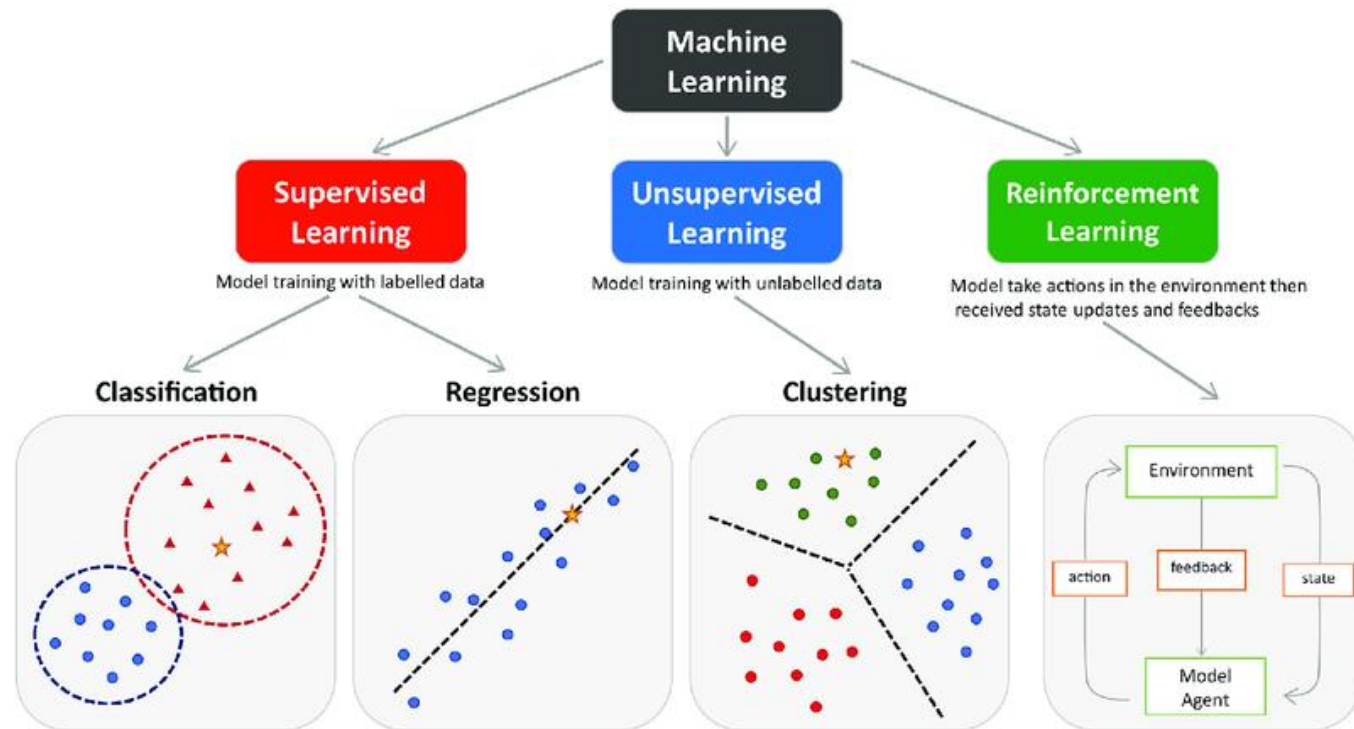
***Inatel***

Felipe Augusto Pereira de Figueiredo  
felipe.figueiredo@inatel.br

# O que vamos ver?

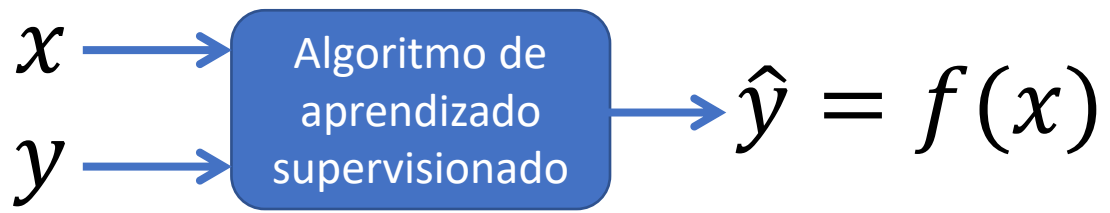
- Anteriormente, vimos através de um exemplo simples como usar a biblioteca TensorFlow para criar uma rede neural e resolver um problema de regressão, ou seja, ajuste de curva.
- O objetivo era ter um contato inicial com a biblioteca e seus princípios básicos de funcionamento.
- Nesse tópico, vamos estender o que vimos anteriormente para um problema mais prático de regressão usando uma base de dados do mundo real.

# Tipos de aprendizado



- Antes de entrarmos na regressão propriamente dita, vamos entender um pouco sobre as formas como os algoritmos de ML aprendem e as arquiteturas que podemos encontrar.
- Podemos agrupar os algoritmos de ML através da forma como eles aprendem:
  - Aprendizado supervisionado
  - Aprendizado não-supervisionado
  - Aprendizado por reforço
- Existem outros, mas esses são os três mais conhecidos.

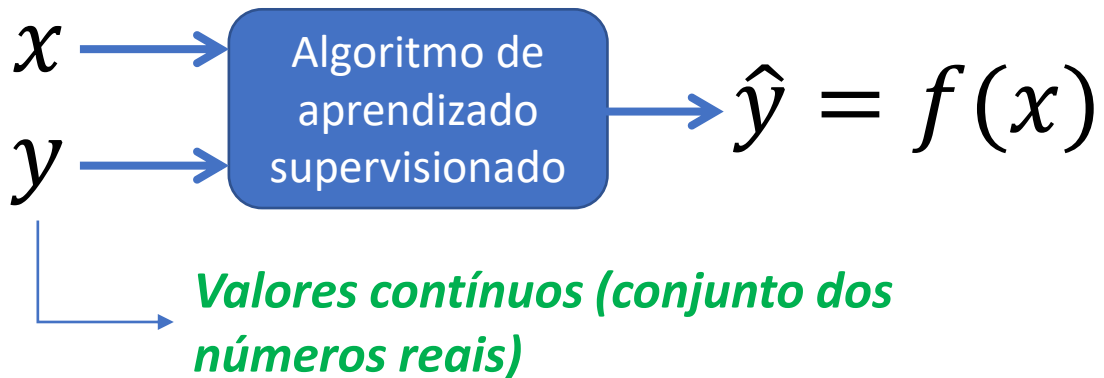
# Aprendizado supervisionado



- O modelo é treinado usando um conjunto de dados que contém exemplos de entrada (também conhecidos como ***atributos***) junto com as saídas correspondentes (***rótulos*** ou respostas corretas).
- O objetivo é que o modelo aprenda a ***mapear as entradas para as saídas corretas***, de modo que ele possa fazer previsões precisas em novos dados para os quais as saídas corretas não são conhecidas (i.e., generalização).

# Aprendizado supervisionado

## Regressão

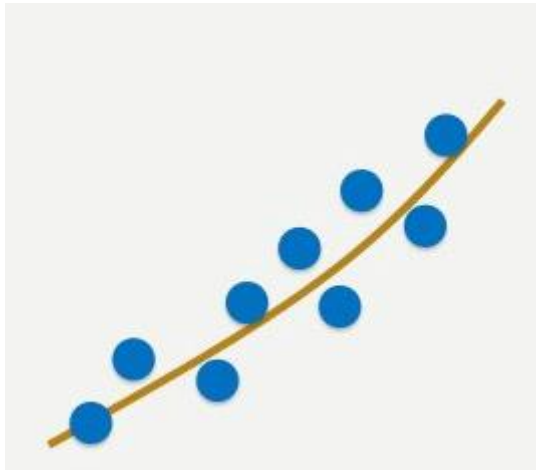


## Classificação

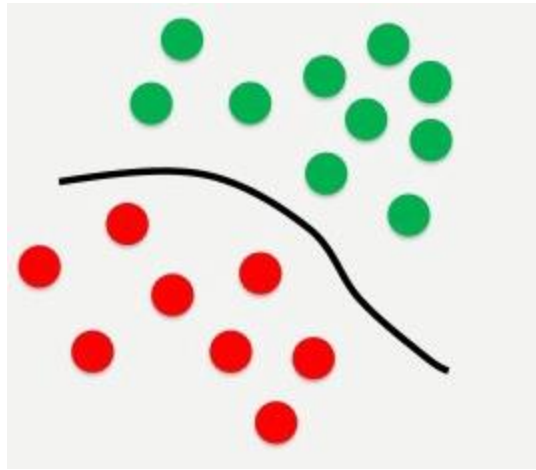


- O termo ***supervisionado*** reflete o fato de que durante o treinamento, o modelo é ***guiado*** ou ***supervisionado*** pelos ***rótulos*** para ***aprender a mapear as entradas para as saídas desejadas***.
- Algoritmos de aprendizado supervisionado são frequentemente usados em tarefas como ***classificação*** (atribuir uma categoria ou classe a uma entrada) e ***regressão*** (predizer um valor numérico).
- São algoritmos usados em predição de preços de imóveis, detecção de objetos, etc.

# Classificação e regressão



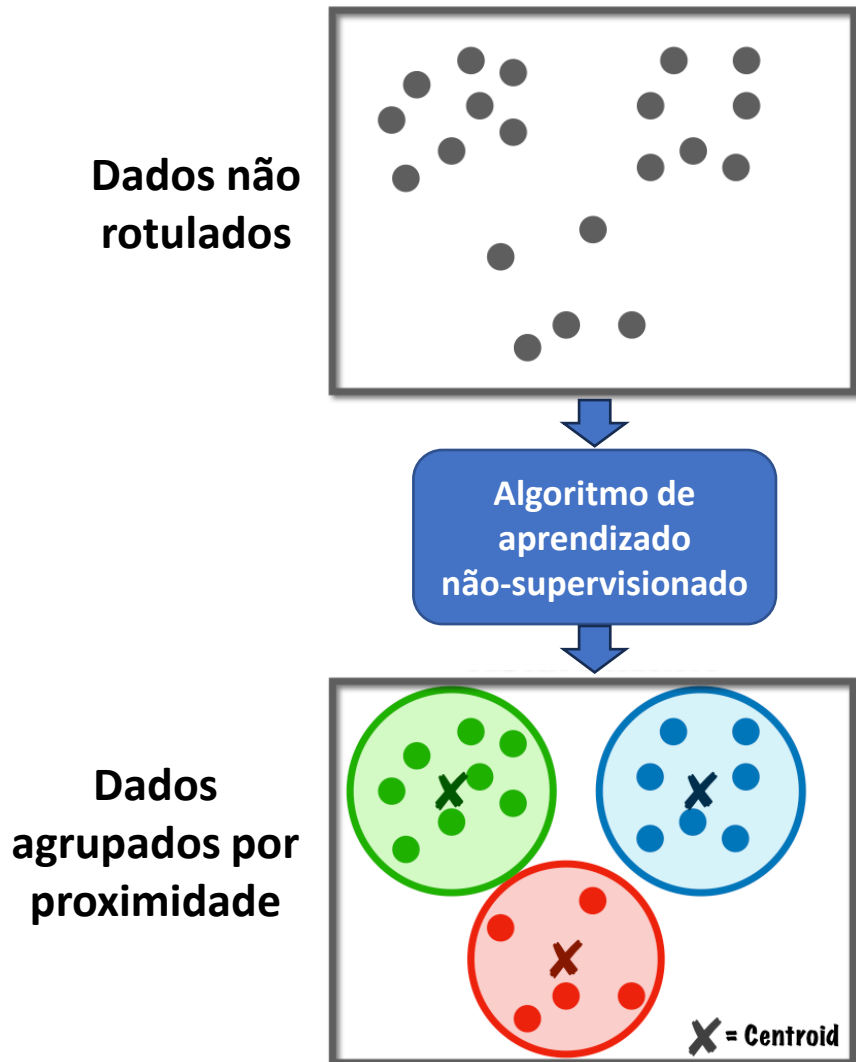
$f(x)$  **aproxima** o comportamento dos dados.



$f(x)$  **separa** os dados em classes.

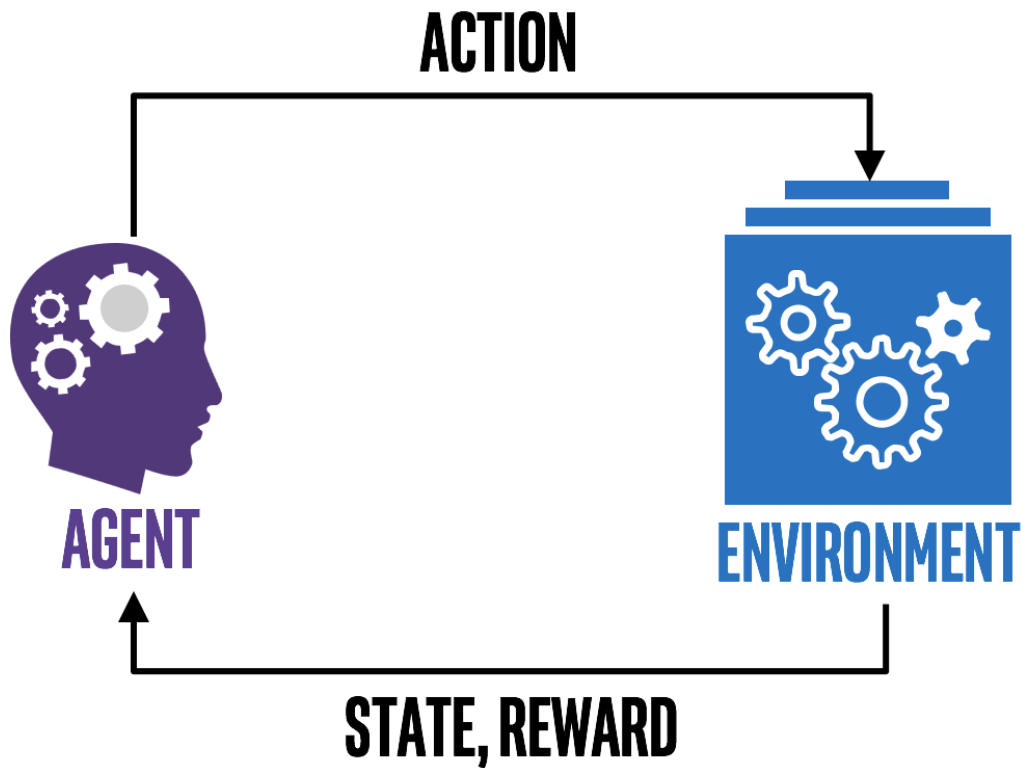
- O objetivo da **regressão** é encontrar uma função que tenha o menor erro possível em relação a todos os pontos do conjunto de dados.
- O objetivo da **classificação** é encontrar uma função que separe, i.e., classifique, os dados com o menor erro possível.
- Nosso curso focará mais nesses dois problemas de aprendizado supervisionado.

# Aprendizado não-supervisionado



- Como já dá para supor, o modelo é treinado sem rótulos.
- Ou seja, não se sabe quais são as respostas corretas para as entradas.
- No aprendizado não-supervisionado, o modelo busca descobrir padrões, estruturas ou relações intrínsecas nos dados sem o auxílio de orientação explícita, se baseando apenas, por exemplo, na similaridade entre as entradas (i.e., os atributos).
- Algoritmos usados em detecção de anomalias, redução de dimensionalidade, compressão, etc.

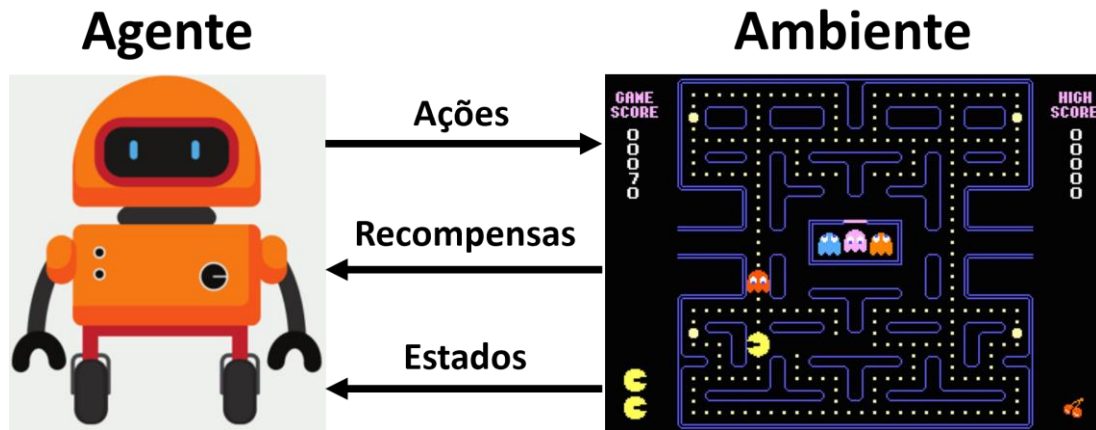
# Aprendizado por reforço



- Aprendizado totalmente diferente dos anteriores, pois *não existem exemplos de treinamento*, sejam eles rotulados ou não.
- Frequentemente usado em situações em que não é possível ter um conjunto de treinamento, ou em ambientes onde as respostas não são conhecidas com antecedência.
- Abordagem de aprendizado em que um agente, i.e., o algoritmo de ML, aprende a tomar decisões interagindo com um ambiente.



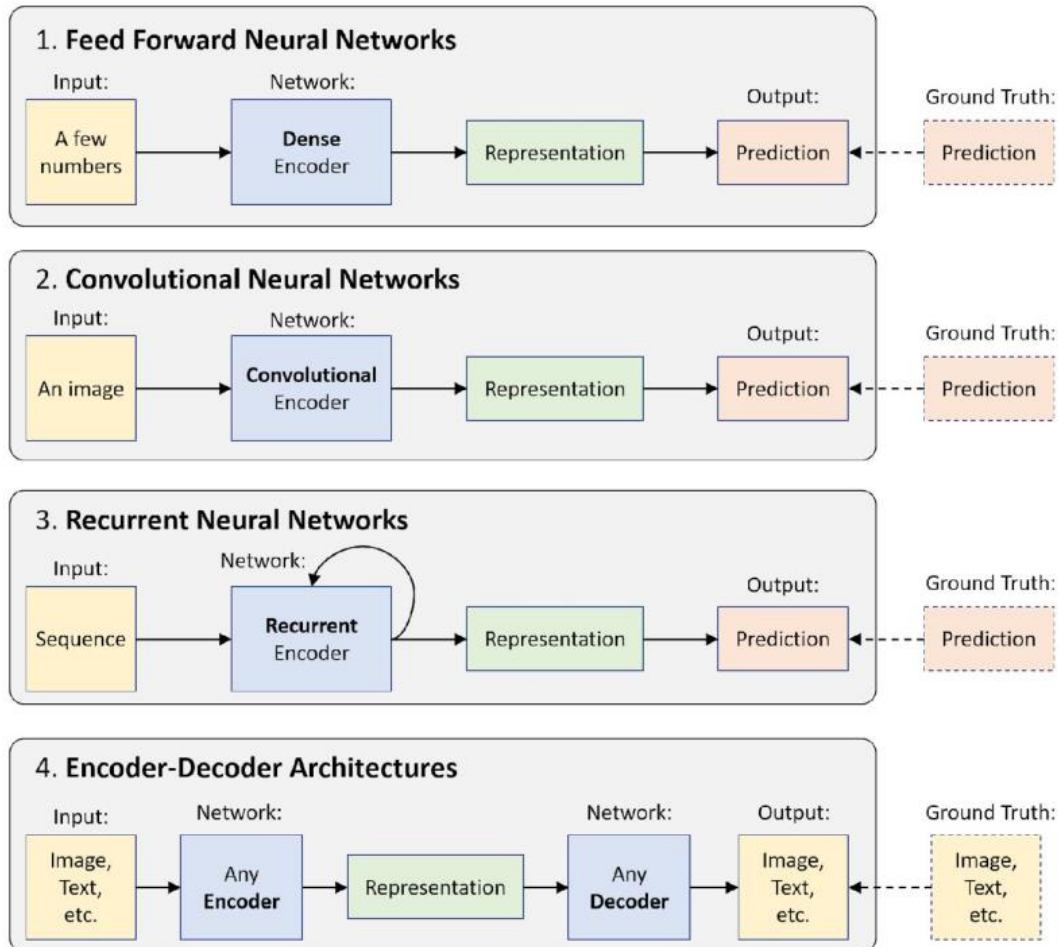
# Aprendizado por reforço



- O agente toma ações em um ambiente para maximizar a recompensa acumulada ao longo do tempo.
- O objetivo é encontrar uma função, chamada de política, que mapeie os estados na sequência de ações que maximize a recompensa.
- A principal característica do aprendizado por reforço é que o agente aprende através de tentativa e erro, ajustando suas ações com base nas recompensas que recebe do ambiente.
- Algoritmos usados em jogos, robótica, carros autônomos, etc.

# Modelos de aprendizado supervisionado

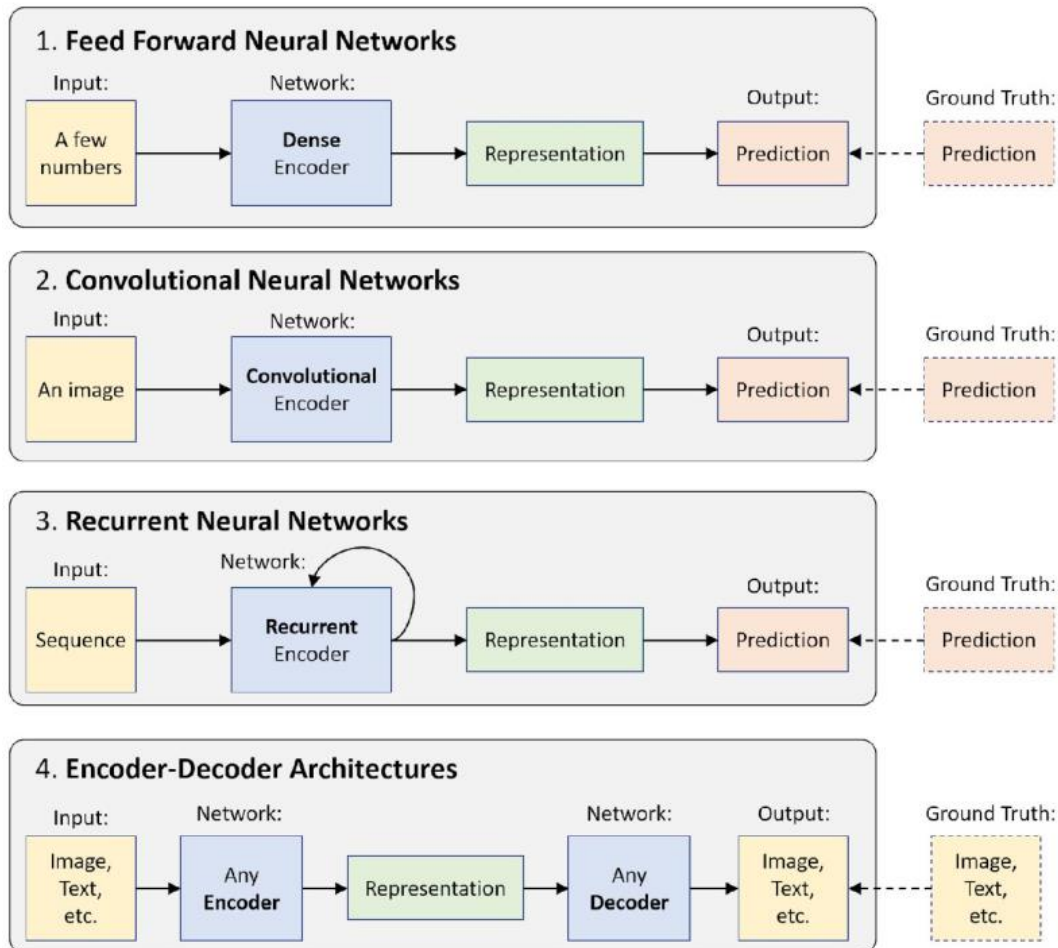
## Supervised Learning



- Dos três paradigmas de aprendizado, neste curso, vamos focar no supervisionado.
- Dentro do aprendizado supervisionado, temos alguns modelos (ou arquiteturas) de redes neurais que são bastante usados.
- **OBS.:** Além das redes neurais, existem outros modelos que seguem esse paradigma de aprendizado: regressão linear/logística, árvores de decisão, k-vizinhos mais próximos, etc.

# Rede neural de alimentação direta (DNN)

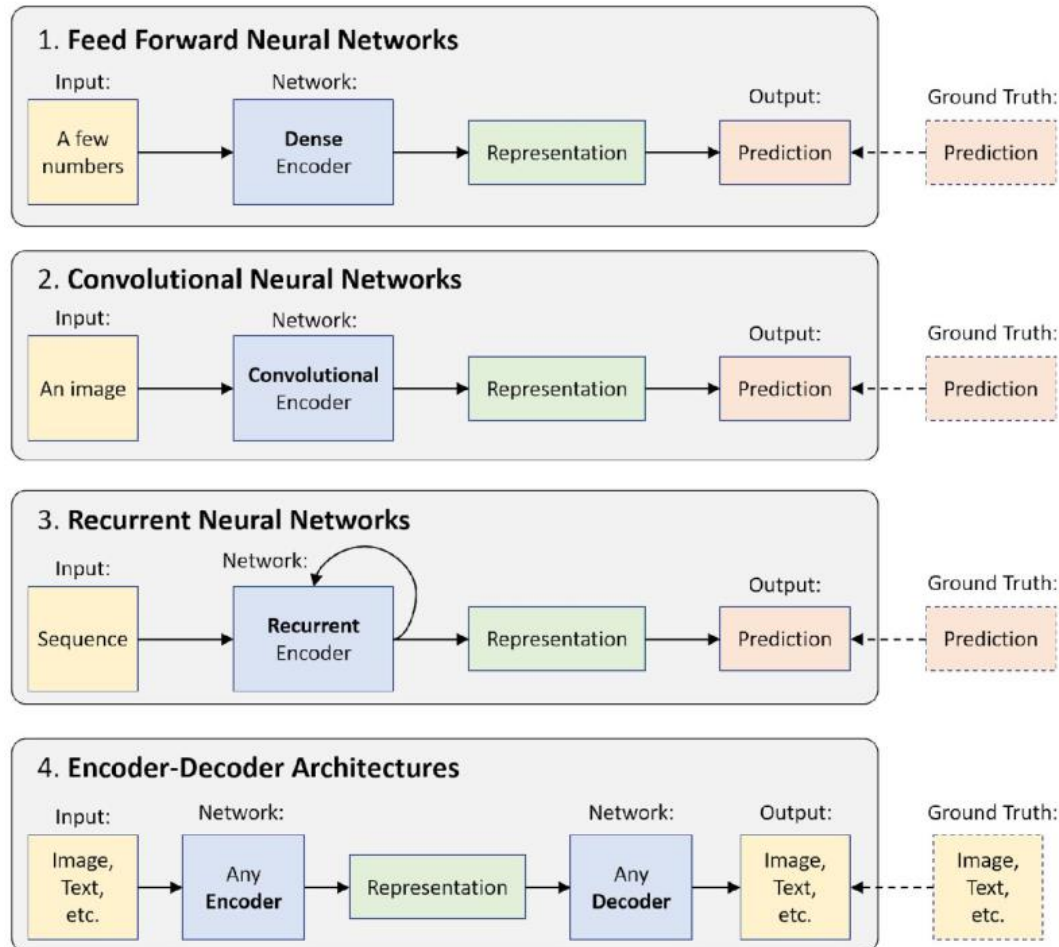
## Supervised Learning



- Modelo que vimos anteriormente.
- Também chamado de rede neural densa, devido aos neurônios estarem densamente conectados.
- Ainda outro nome que este modelo recebe é *Multilayer Perceptron* (MLP).
- É chamada de “*feed forward*” porque a informação flui através da rede da entrada para saída.

# Rede neural convolucional (CNN)

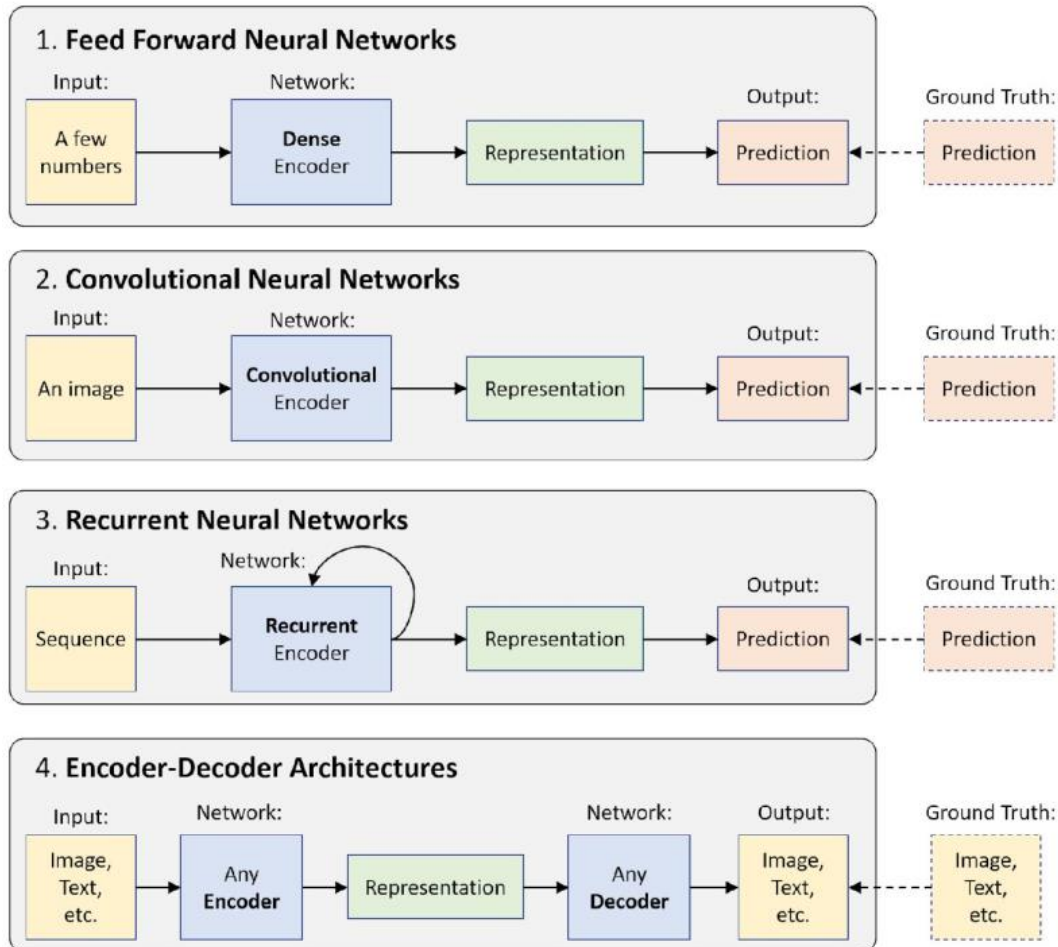
## Supervised Learning



- Arquitetura projetada para lidar com dados matriciais, como imagens e vídeos.
- O coração das CNNs são as camadas convolucionais.
  - Elas aplicam operações de convolução para extrair e identificar padrões espaciais em imagens ou vídeos.
  - Isso permite que a rede aprenda automaticamente a detectar características visuais como bordas, texturas e outros padrões visuais.
- São altamente eficazes em problemas de processamento de imagem, incluindo classificação, detecção e segmentação.

# Rede neural recorrente (RNN)

## Supervised Learning

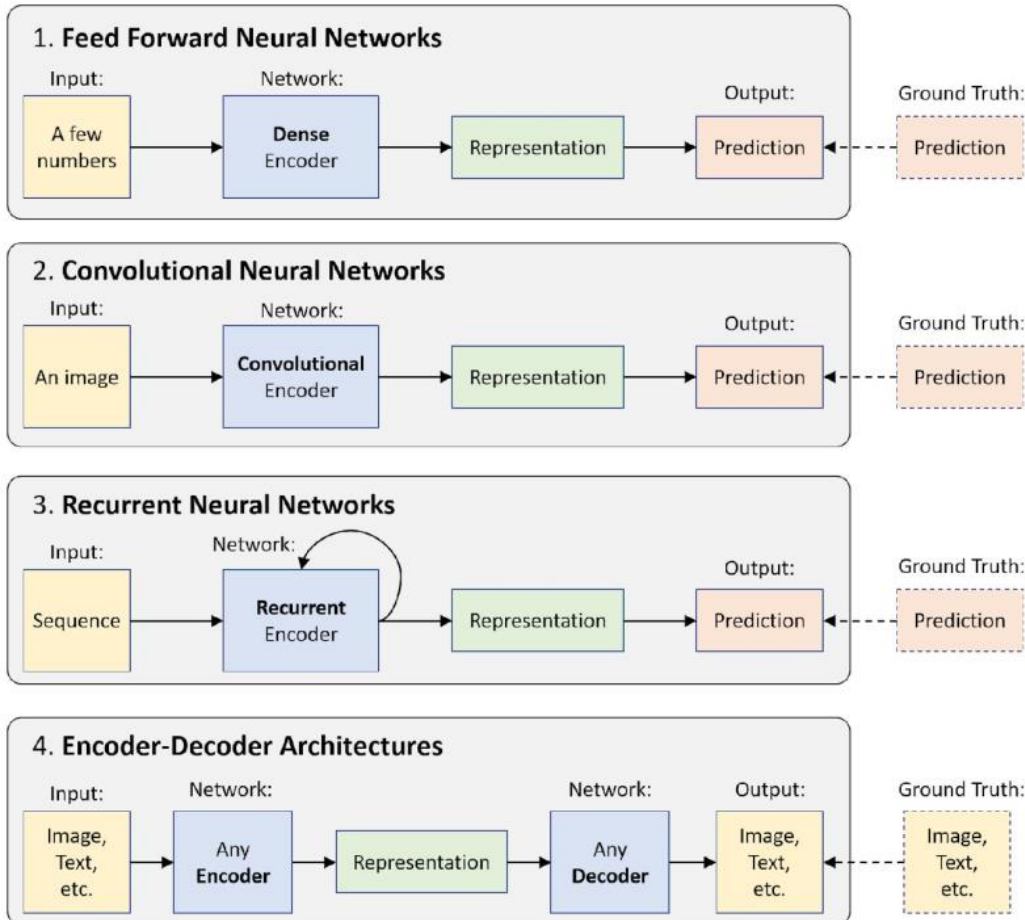


- Modelo projetado para lidar com dados sequenciais, onde a ordem e a dependência temporal dos elementos são importantes.
- As RNNs têm uma memória de estado interna que permite que elas mantenham informações sobre as entradas anteriores ao longo do tempo.
- São adequadas para tarefas que envolvam sequências, como análise de texto, reconhecimento de fala, previsão de séries temporais e tradução automática.



# Autoencoders

## Supervised Learning

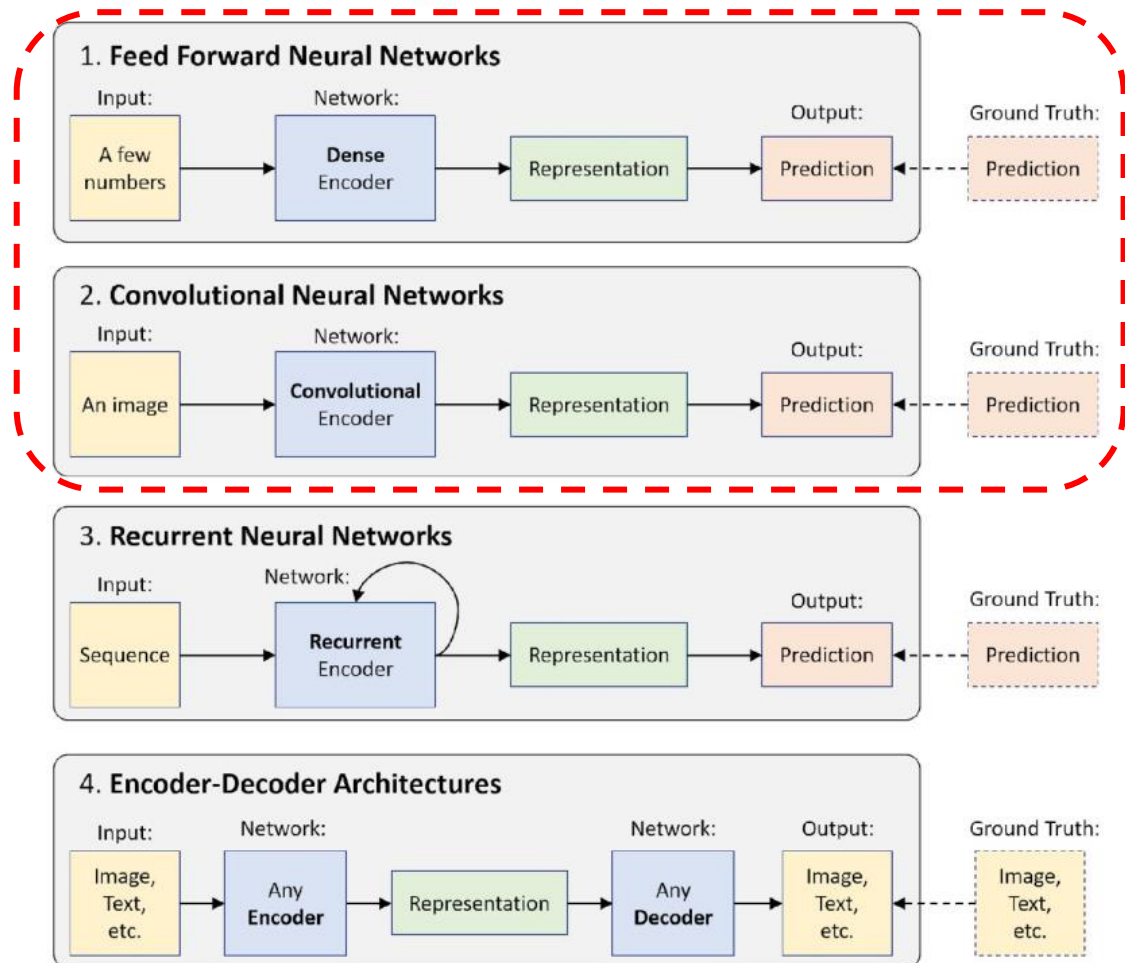


**OBS.:** Alguns autores não o consideram supervisionado.

- Modelo de rede neural composto por duas partes principais: o *encoder* e o *decoder*.
- Objetivo é aprender uma representação latente que capture as principais características e padrões dos dados de entrada.
- O *encoder* transforma a entrada em uma representação latente (de menor ou maior dimensionalidade) e o *decoder* tenta reconstruir a entrada original a partir dela.
- São usados em compressão de dados, remoção de ruído, geração de dados sintéticos, etc.

# Modelos de aprendizado supervisionado

## Supervised Learning

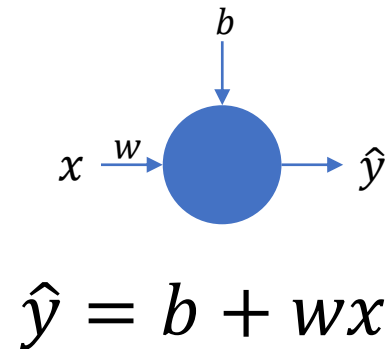
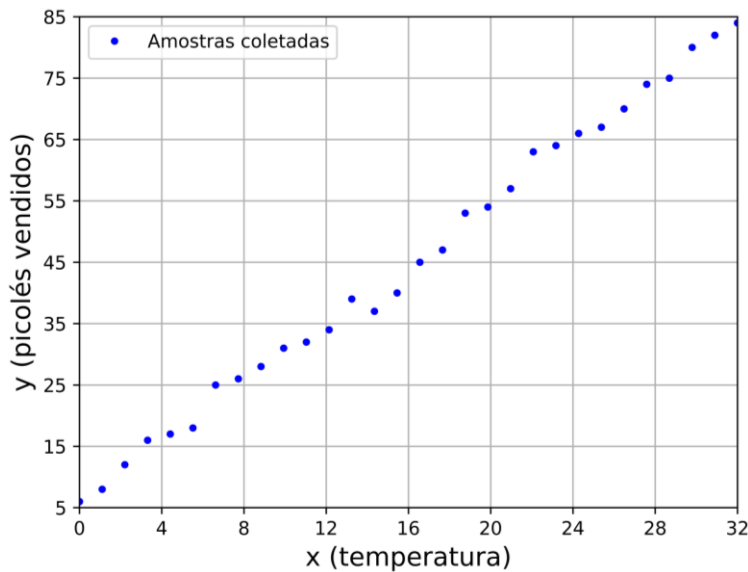


- No nosso curso, iremos focar mais nas DNNs e CNNs.

# Regressão

$$\mathbf{x} = \{-1, 0, 1, 2, 3, 4\}$$

$$\mathbf{y} = \{-3, -1, 1, 3, 5, 7\}$$



- Anteriormente, nós resolvemos um problema de regressão bem simples, onde queríamos mapear um único valor de  $x$  em um valor de saída,  $y$ .
- Fizemos o mapeamento usando uma reta.
- Podemos fazer uma analogia com o problema de prever o número de picolés que serão vendidos em um dia,  $y$ , dado a temperatura média daquele dia,  $x$ .
- Esse problema só tem um **atributo** de entrada, a temperatura,  $x$ .

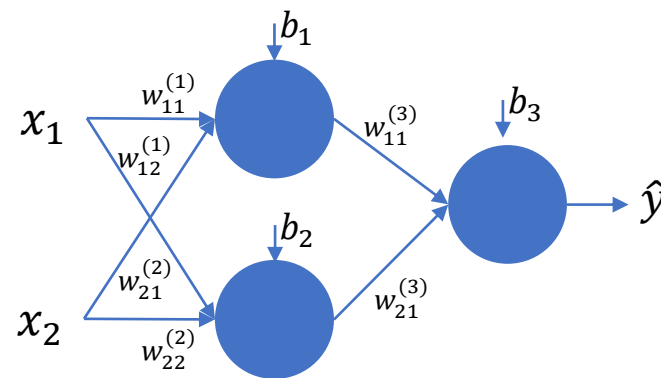
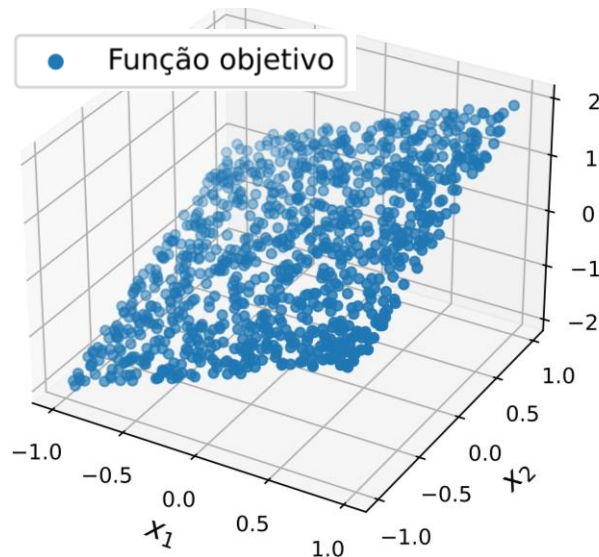


# Regressão

$$\mathbf{x}_1 = \{-1, 0, 1, 2, 3, 4\}$$

$$\mathbf{x}_2 = \{-8, 1, 3, 7, 0, 2\}$$

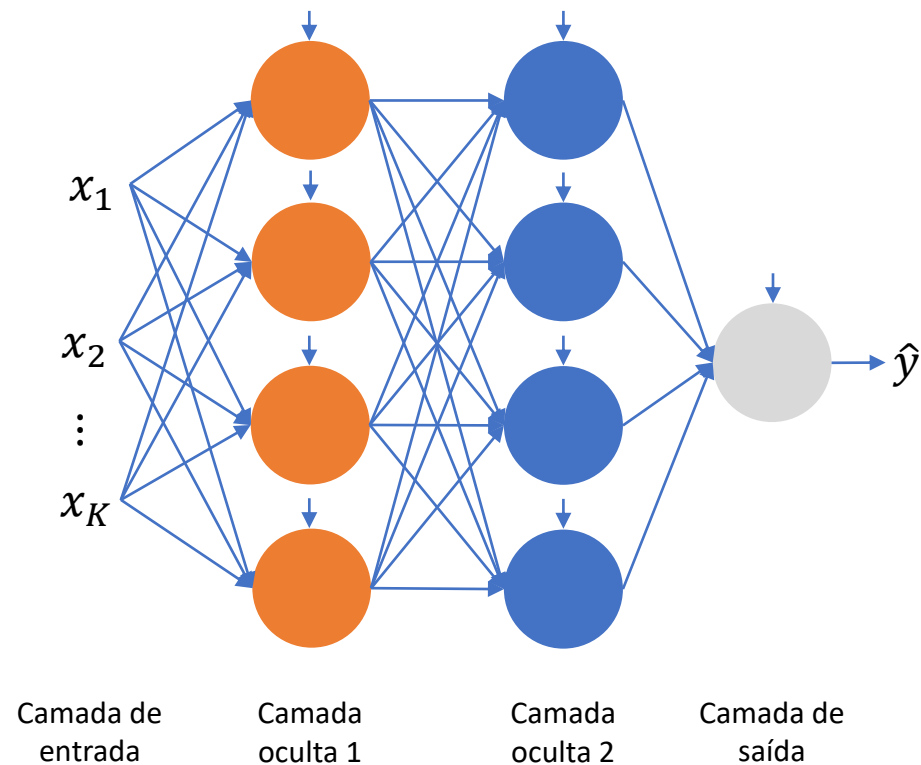
$$\mathbf{y} = \{-8, 0, 7, 1, 2, 3\}$$



$$\hat{y} = a_0 + a_1x_1 + a_2x_2$$

- Mas e se quisermos um modelo que leve em consideração não só a temperatura, mas o mês do ano também?
- O modelo agora terá 2 **atributos** (i.e., entradas).
- Com ativações lineares, a rede neural ao lado representa a função de um plano dado por  $\hat{y}$ .

# Regressão

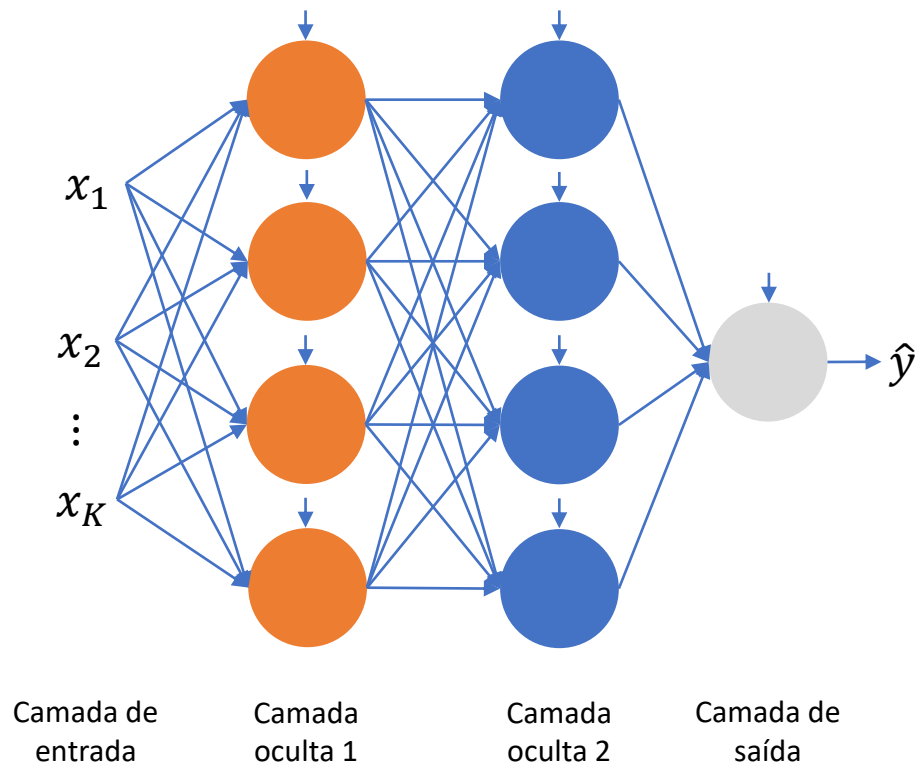


- Podemos extrapolar isso pra quantos ***atributos*** forem necessários.
- O modelo ao lado tem  $K$  ***atributos*** (i.e., entradas).
- Com ***ativações lineares***, a rede neural ao lado representa a função de um hiperplano.

$$\begin{aligned}\hat{y} &= a_0 + a_1x_1 + a_2x_2 + \dots + a_Kx_K \\ &= \sum_{i=0}^K a_ix_i,\end{aligned}$$

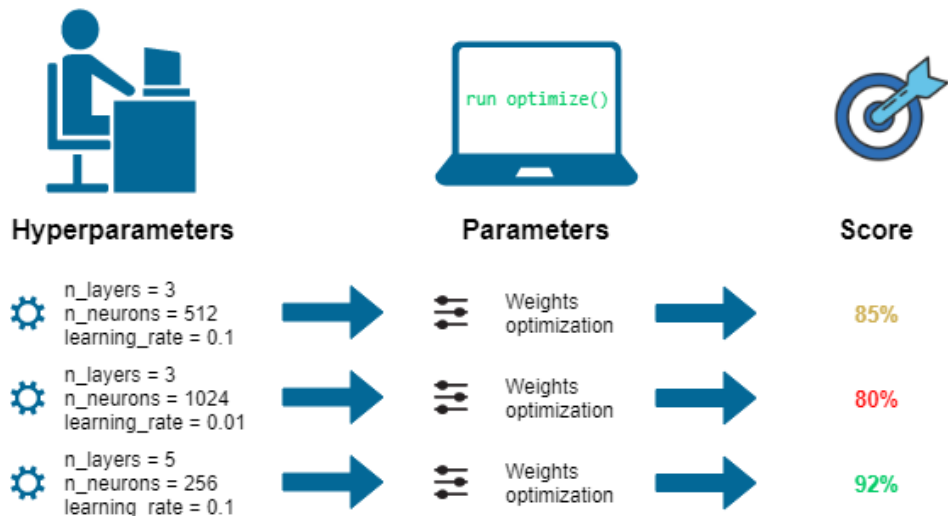
onde  $x_0 = 1$ .

# Aproximação universal de funções



- Com ativações não lineares (sigmóide, *relu*, etc.) e uma camada oculta, podemos aproximar qualquer tipo de função contínua, incluindo o hiperplano, bastando encontrar o número de neurônios necessários.
- Com duas camadas ocultas, podemos aproximar até funções com descontinuidades.
- Como encontramos o número ideal de camadas e neurônios?

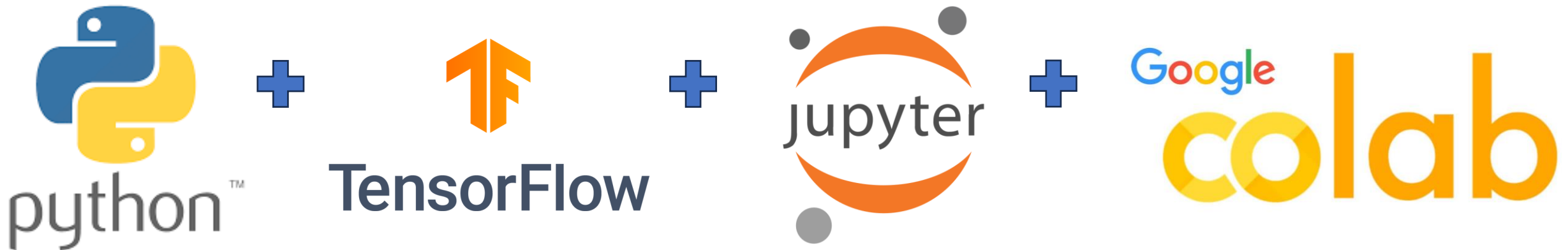
# Otimização hiperparamétrica



- É o processo de encontrar os melhores conjuntos de hiperparâmetros para um modelo de ML.
- Hiperparâmetros são parâmetros que não são aprendidos durante o treinamento do modelo, mas que afetam seu desempenho e comportamento.
- Exemplos de hiperparâmetros incluem taxa de aprendizagem, número de camadas e neurônios, tamanho do *mini-batch*, otimizador, e muitos outros.
- Algumas bibliotecas populares são: KerasTuner, Optuna, Scikit-learn, Hyperopt, etc.

# Exemplo

- [Regressão de preços de residências usando redes neurais densas \(DNNs\)](#)



# ML Workflow



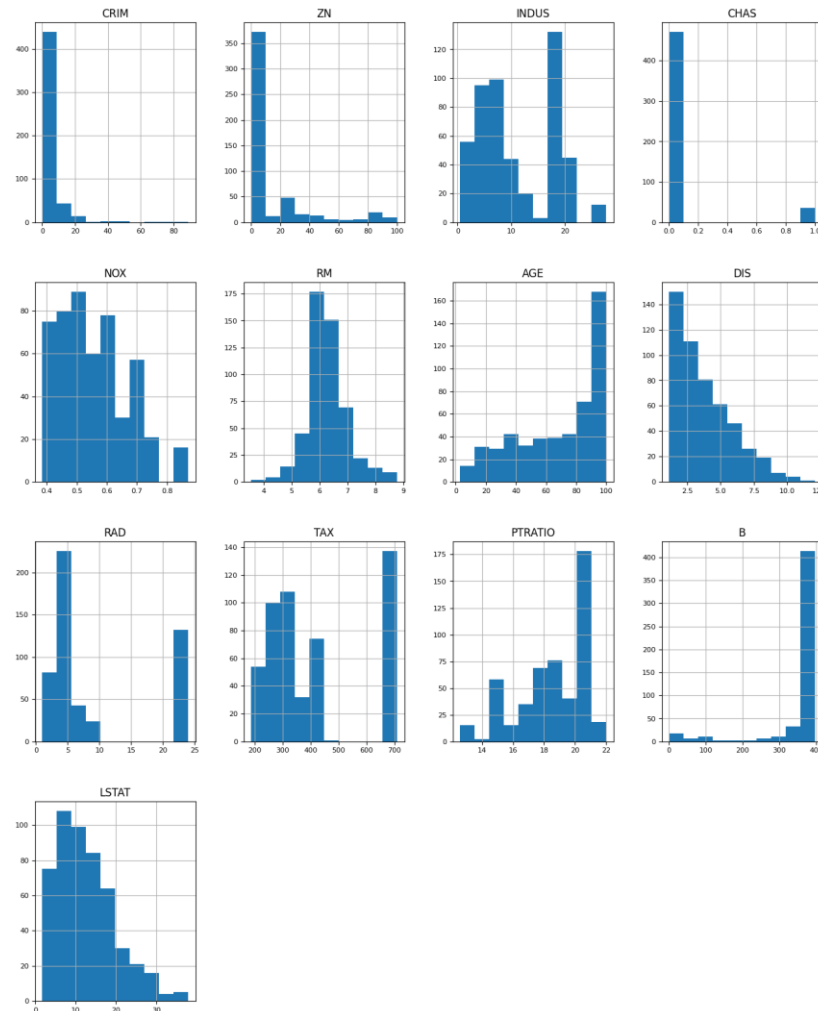
Coletar  
Dados

```
data = tf.keras.datasets.boston_housing
```

```
(x_train, y_train), (x_test, y_test) = data.load_data()
```

- O primeiro passo no fluxo de trabalho com modelos de ML envolve a coleta de dados.
- Posso coletar realmente, por exemplo, gravar sons ou vídeos, tirar fotos, etc. ou reusar um conjunto de dados existente.

# ML Workflow



- Na sequência, fazemos uma análise exploratória dos dados (*exploratory data analysis* – EDA), avaliando intervalos dos atributos e em busca de valores faltantes, discrepantes, etc.

# ML Workflow



```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
scaler.fit(x_train)
```

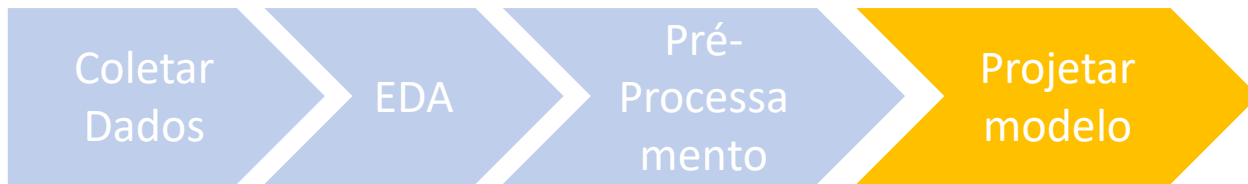
```
x_train_std = scaler.transform(x_train)
```

```
x_test_std = scaler.transform(x_test)
```

- Em seguida, realizamos o pré-processamento dos dados.
- Essa tarefa pode envolver a remoção de valores discrepantes, preenchimento ou remoção de exemplos com dados incompletos, escalonamento dos atributos, etc.



# ML Workflow



```
model = tf.keras.models.Sequential(  
    [  
        tf.keras.layers.Dense(20, input_shape=[13], activation='relu'),  
        tf.keras.layers.Dense(1)  
    ]  
)  
  
model.compile(  
    optimizer='adam',  
    loss='mse',  
    metrics=['mae']  
)
```

- Após, temos a fase de criação do modelo.
- Envolve a definição da arquitetura: quantidade de camadas, número de nós por camada, funções de ativação, otimizador, passo de aprendizagem, métricas, etc.

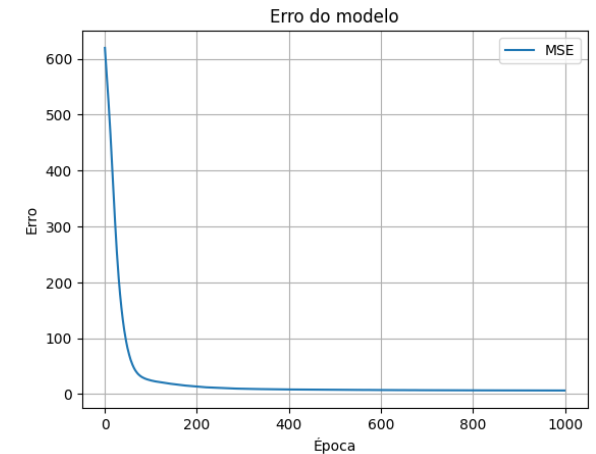
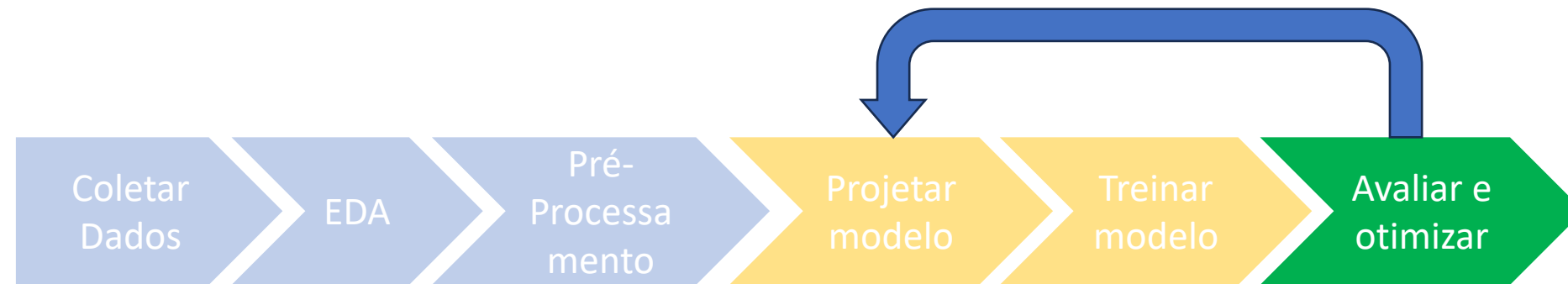
# ML Workflow



```
history = model.fit(  
    x_train_std,  
    y_train,  
    epochs=1000,  
)
```

- O treinamento do modelo vem na sequência.
- A entrada desta etapa são os dados já pré-processados.

# ML Workflow



```
test_eval = model.evaluate(x_test_std, y_test)
train_eval = model.evaluate(x_train_std, y_train)
```

```
tuner.search(
    x_train_std, y_train,
    epochs=500,
    validation_data=(x_test_std, y_test)
)
```

- Avaliar o modelo envolver analisar os resultados obtidos após o treinamento.
- Analisar indícios de que o modelo está aprendendo:
  - Curva de erro com caída rápida no início e redução ao longo do treinamento, se tornando praticamente constante (indicação de convergência).
  - Comparar os erros de treinamento e validação, os quais devem ser pequenos e próximos (caso contrário, indicação de sobreajuste).
- Se o modelo não estiver bom, devemos otimizá-lo, manualmente ou através de técnicas de otimização hiperparamétrica.

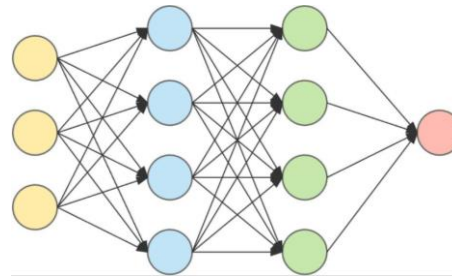
# ML Workflow



```
xt = np.array([1.1, 0., 9., 0., 0.6, 7., 92., 3.8, 4., 300., 21., 200, 19.5])
xt = np.reshape(xt, (1, 13))
xt_norm = scaler.transform(xt)
yt = model.predict(xt_norm)
```

- Após obtermos um bom modelo, o colocamos em “produção” para lidar com dados do mundo real (inéditos) e oferecer *insights* ou auxiliar em tomadas de decisão.

# ML Workflow



- Esse é o fluxo de trabalho que geralmente seguimos para trabalhar com modelos de aprendizado de máquina.
- O fluxo com o tinyML terá uma fase adicional intermediária entre avaliar/otimizar e a inferência, que será a etapa de conversão (i.e., compressão) do modelo para o executarmos em dispositivos embarcados.

# Atividades

- Quiz: “***TP557 – Regressão com DNNs (Parte II)***”.
- Exercício #1: [Regressão sem escalonamento](#)
- Exercício #2: [Otimização hiperparamétrica](#)

Perguntas?

Obrigado!



