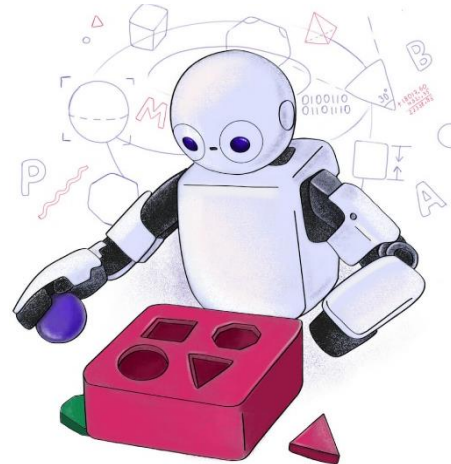


TP557 - Tópicos avançados em IoT e Machine Learning: *Regressão com DNNs (Parte II)*



Inatel

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

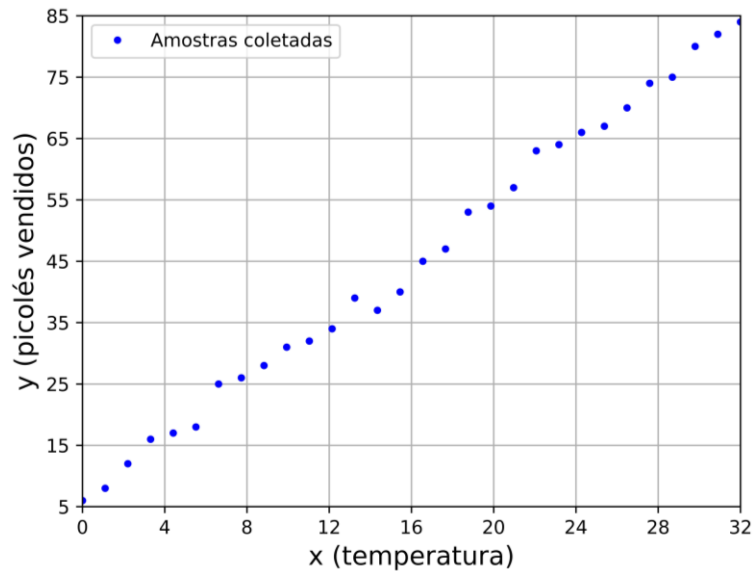
O que vamos ver?

- 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, um problema de 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*.

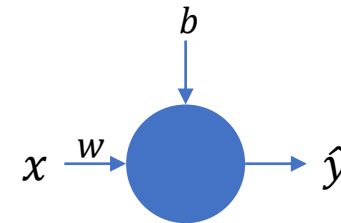
Regressão

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

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



- 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 como nossa função hipótese.

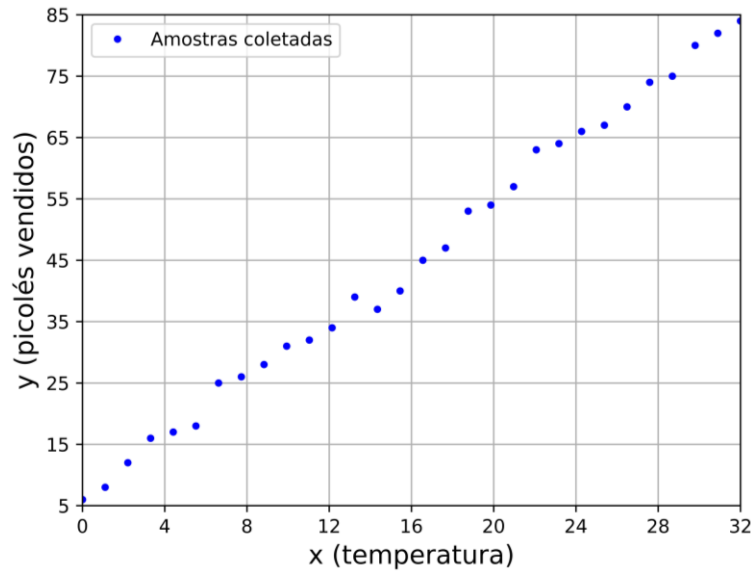


$$\hat{y} = b + wx$$

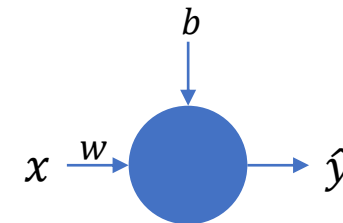
Regressão

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

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



- 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 .



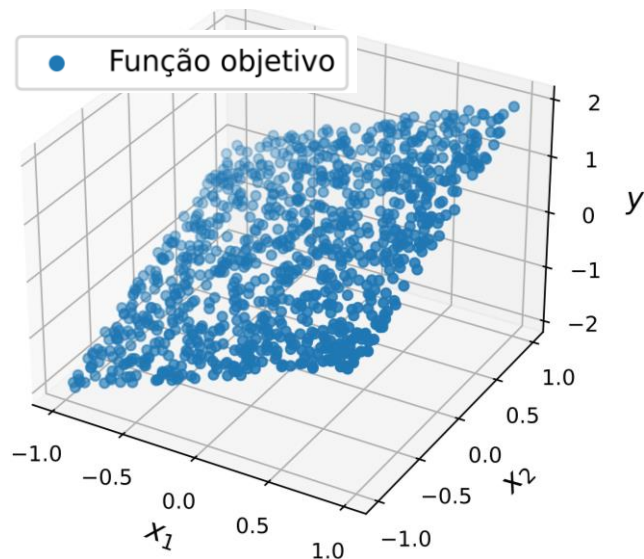
$$\hat{y} = b + wx$$

Regressão

$$x_1 = \{-1, 0, 1, 2, 3, 4\}$$

$$x_2 = \{-8, 1, 3, 7, 0, 2\}$$

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



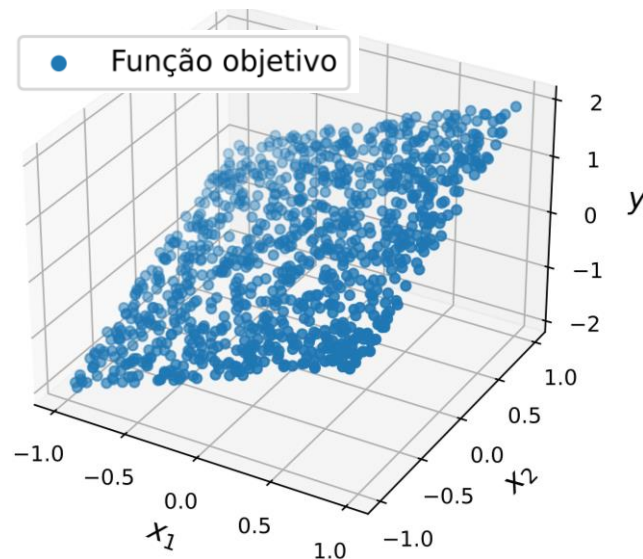
- Mas e se quisermos um modelo que *leve em consideração não só a temperatura*, mas o *mês do ano também*?
- O modelo agora terá 2 *atributos* (i.e., entradas).
- Por exemplo, a figura ao lado mostra a distribuição dos dados para um problema com dois atributos.
- Vejam que eles formam um plano.
- Portanto, existe uma relação linear entre os atributos e a saída.

Regressão

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

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

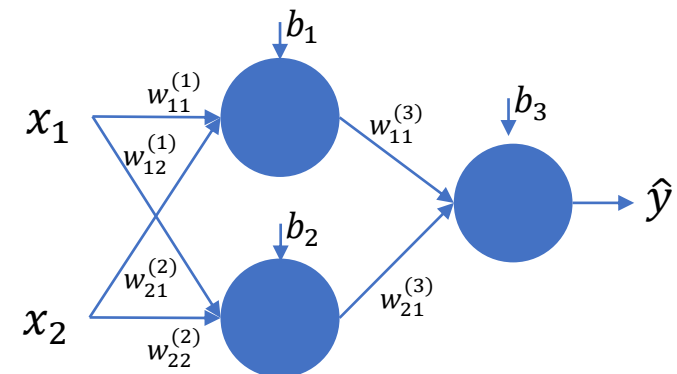
$$\mathbf{y} = \{-8, 0, 7, 1, 2, 3\}$$



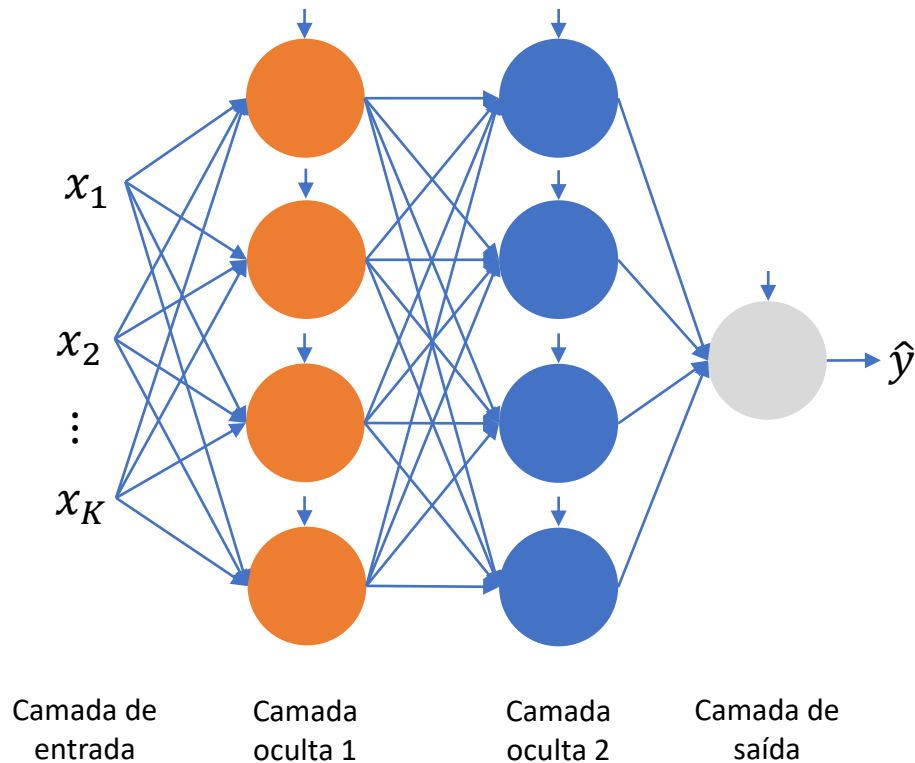
- Para aproximar esses dados, podemos usar a **equação de um plano** como nossa **função hipótese**:

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

- Usando **ativações lineares**, a rede neural abaixo representa a **função do plano**.



Regressão

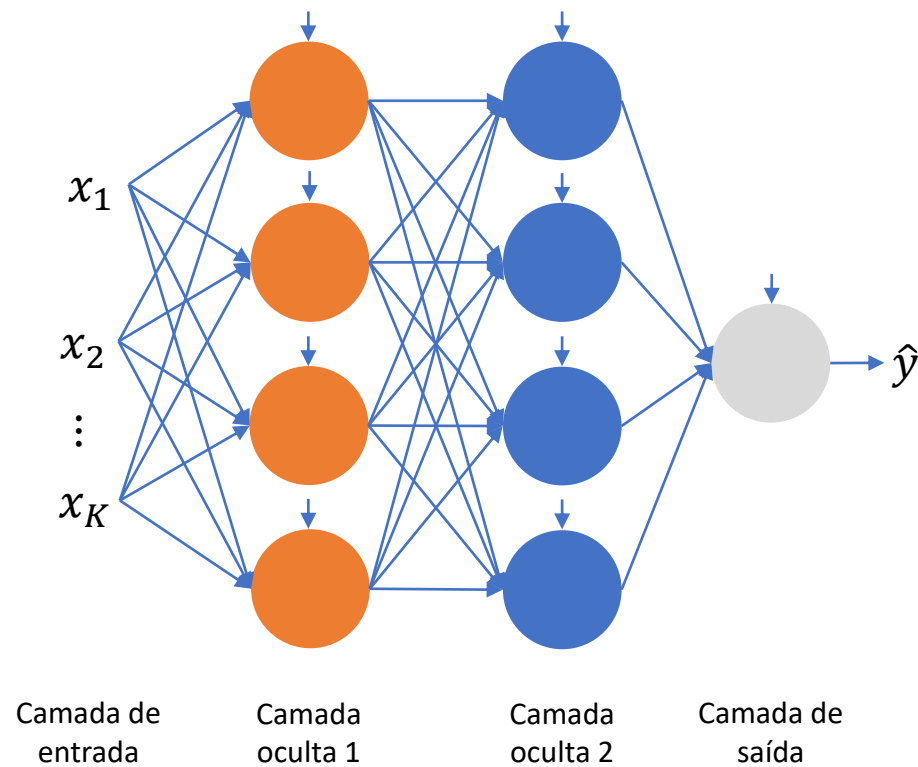


- Podemos extrapolar isso pra quantos ***atributos*** forem necessários.
- O modelo ao lado tem K ***atributos*** (i.e., entradas).
- Com ***ativações lineares***, a rede neural ao lado representa a ***função de um hiperplano***

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

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**.

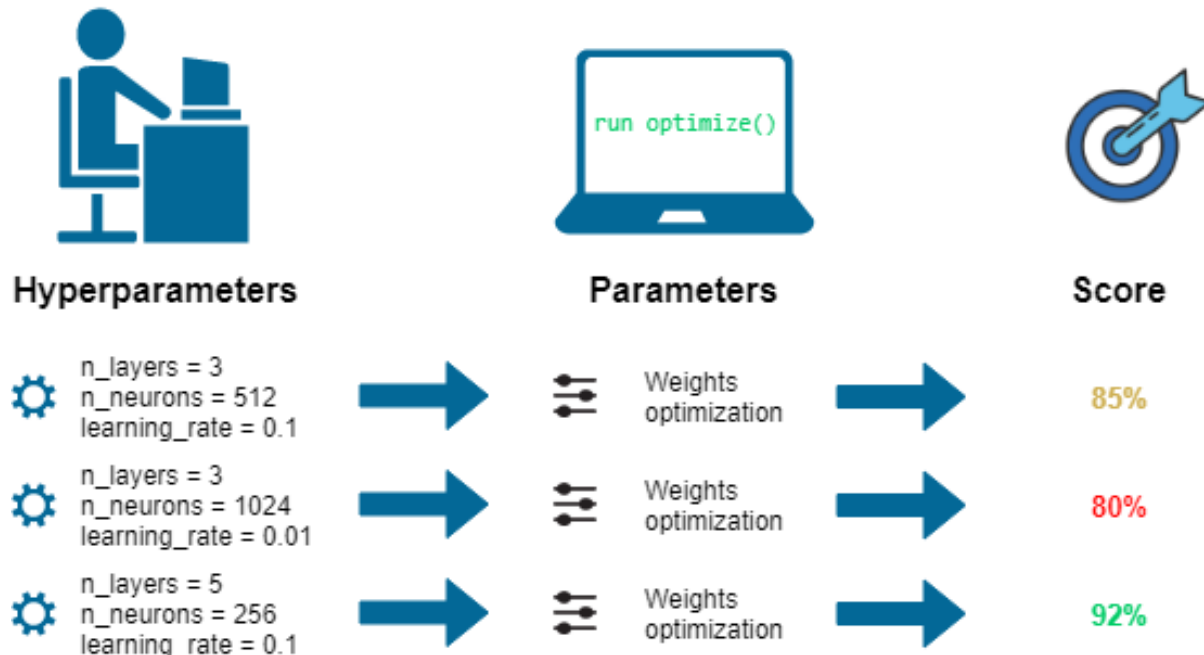
*Mas como encontramos o
número ideal de camadas e
neurônios?*

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*.

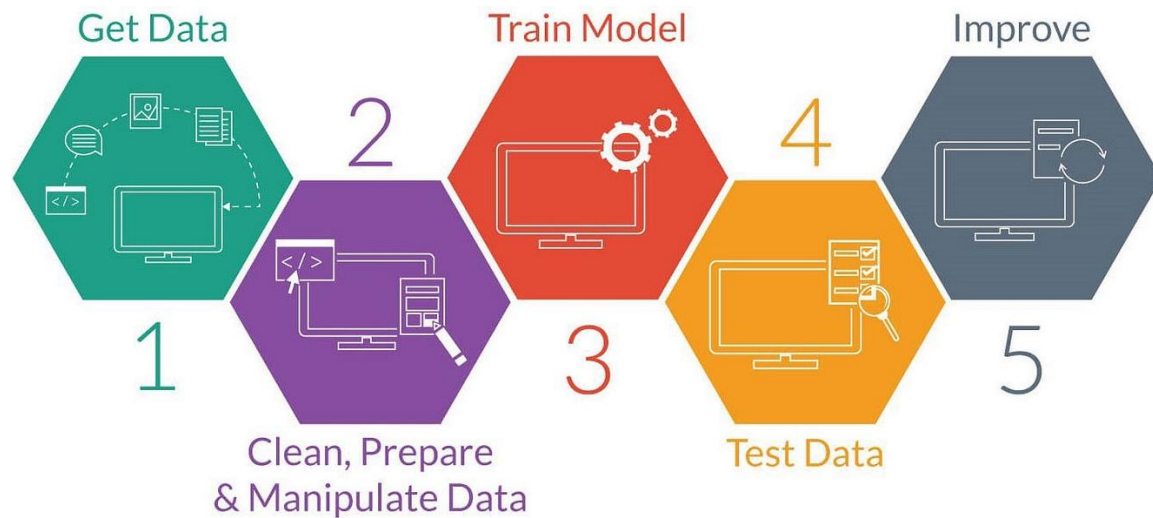


Otimização hiperparamétrica



- Exemplos de hiperparâmetros incluem a *taxa de aprendizagem*, *número de camadas* e *neurônios*, *tamanho do mini-batch*, *otimizador*, e muitos outros.
- Existem diversas bibliotecas que automatizam a busca pelos melhores hiperparâmetros.
- As mais populares são:
 - KerasTuner,
 - Optuna,
 - Scikit-learn,
 - Hyperopt, etc.

ML Workflow



- Antes de vermos um exemplo mais complexo sobre regressão com redes neurais densas, vamos falar sobre o fluxo de trabalho com modelos de ML.
- O fluxo de trabalho refere-se à *sequência de etapas que um projeto de aprendizado de máquina segue*, desde a coleta de dados até a implantação do modelo em produção.

ML Workflow



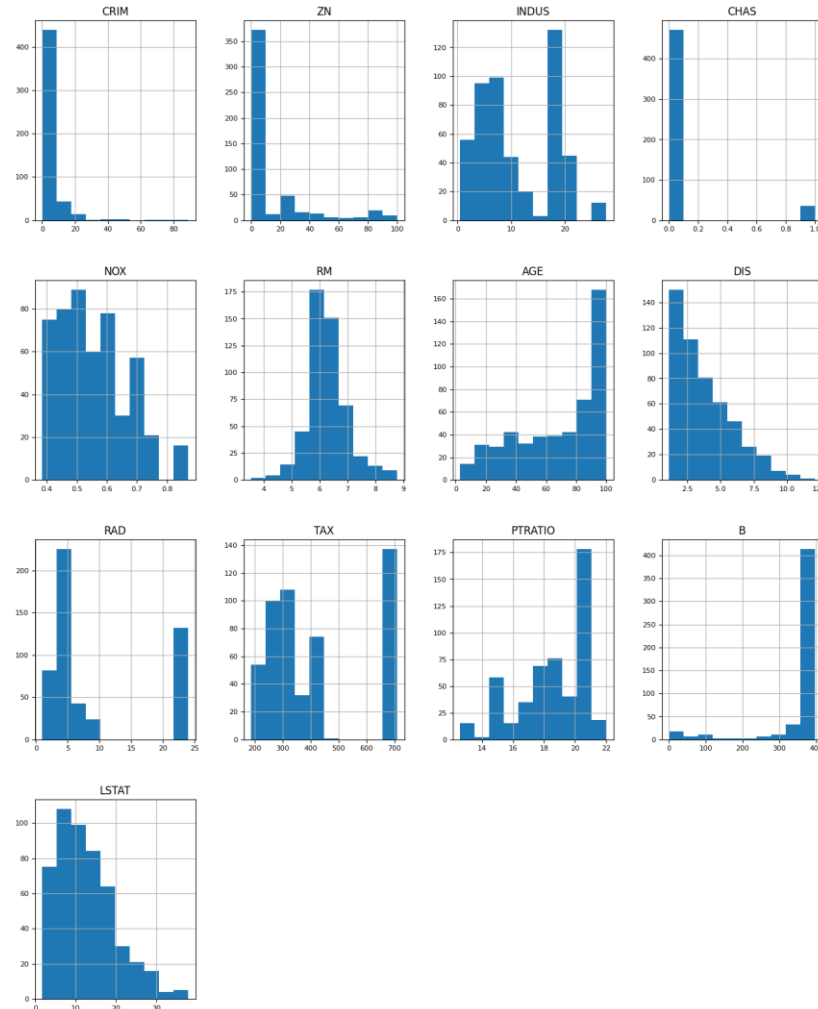
Coletar
Dados

```
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*.
- Podemos *coletar dados* 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 buscando 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*, *data augmentation*, 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.

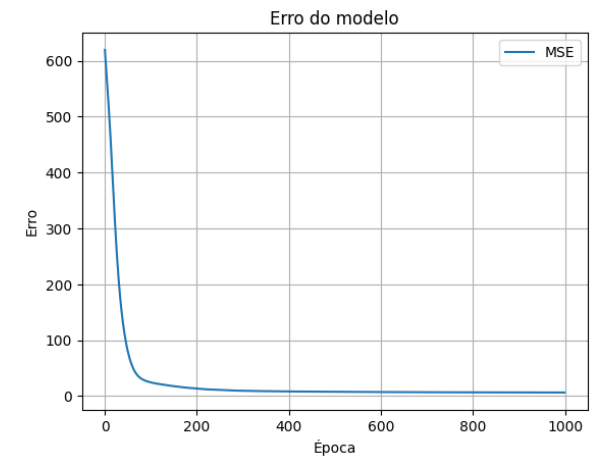
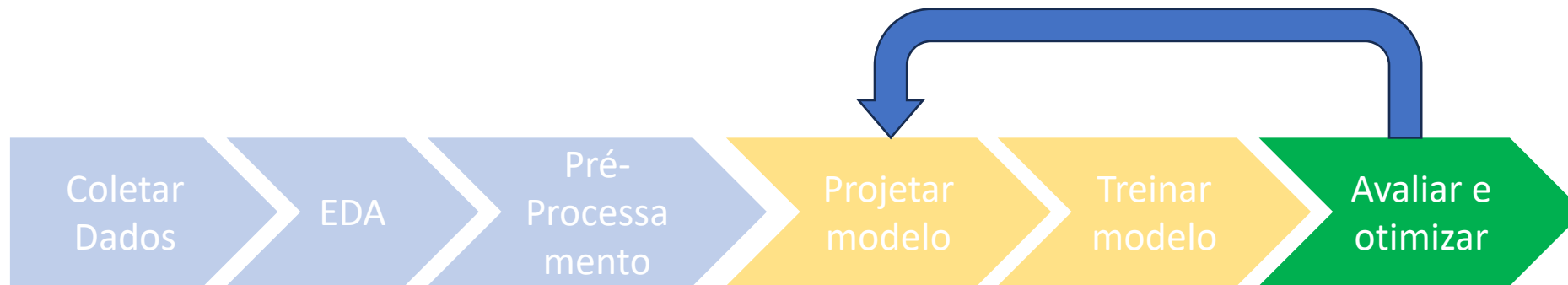
ML Workflow



```
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)
)
```

- **Análise** dos resultados obtidos após o treinamento.
- Analisar indícios de que o modelo está aprendendo:
 - Erros de treinamento e validação devem diminuir ao longo do treinamento até que se tornem praticamente constantes (indicação de convergência).
 - Ambos os erros devem ser pequenos e próximos.
- Se o modelo não estiver aprendendo, devemos **otimizá-lo manualmente ou através de técnicas de otimização hiperparamétrica**.

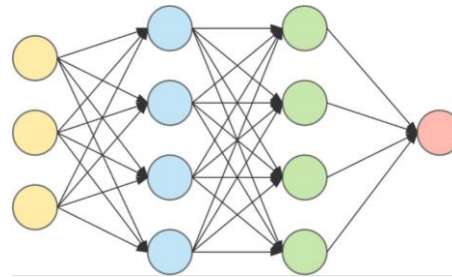
ML Workflow



```
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 (i.e., inéditos) e oferecer *insights* ou auxiliar em tomadas de decisão.

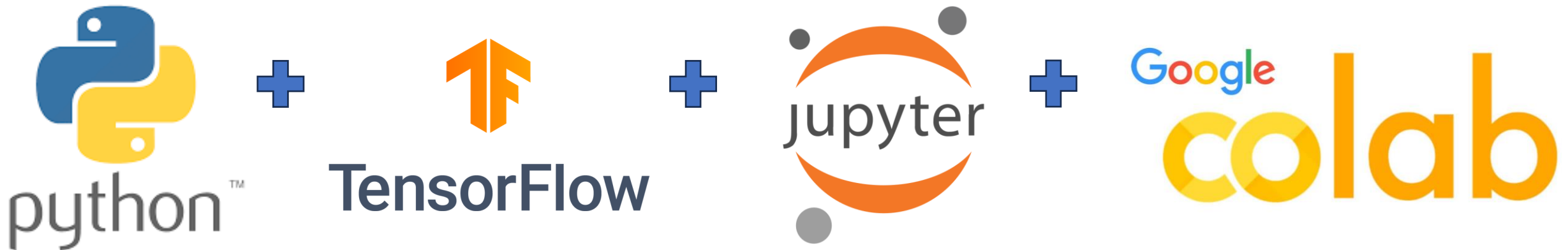
ML Workflow



- 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.

Exemplo

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



Atividades

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

Perguntas?

Obrigado!

