Lab 7 (a) - PCC177/BCC406

REDES NEURAIS E APRENDIZAGEM EM PROFUNDIDADE

Modelos Generativos

Prof. Eduardo e Prof. Pedro

Objetivos:

- Predição de série temporal com redes recorrentes (RNN)

Data da entrega : XX/YY

- Complete o código (marcado com ToDo) e quando requisitado, escreva textos diretamente nos notebooks. Onde tiver *None*, substitua pelo seu código.
- Execute todo notebook e salve tudo em um PDF **nomeado** como "NomeSobrenome-Lab6.pdf"
- Envie o PDF via google FORM

Este notebook é baseado em tensorflow e Keras.

Predição de preço de criptomoedas com redes recorrentes

Informação sobre o Bitcoin : https://www.kaggle.com/ibadia/bitcoin-101-bitcoins-and-detailed-insights

O valor de uma criptomoeda, assim como um ativo funanceiro do mercado de ações, pode ser configurado com uma série temporal. Aqui, consideraremos o valor ponderado do preço diário do Bitcoin para constuir nossa série. O objetivo deste estudo é predizer o próximo valor, baseado nos últimos valores da criptomoeda. Para tal, usaremos de redes recorrentes, pois as mesmas tem memória, o que é importante quando se trata de dados sequenciais.

Carregando os pacotes

```
# Importa as bibliotecas necessárias
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
import pandas as pd
from datetime import datetime
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
import plotly.offline as py
import plotly.graph_objs as go
import numpy as np
import seaborn as sns
py.init_notebook_mode(connected=True)
%matplotlib inline
```

Vamos usar o pacote ***quandl*** para baixar diretamente dados fornecidos por uma corretora de criptomoedas (Kraken).

```
!pip install quandl
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Collecting quandl
  Downloading Quandl-3.7.0-py2.py3-none-any.whl (26 kB)
Requirement already satisfied: pandas>=0.14 in /usr/local/lib/python3.7/dist-package
Collecting inflection>=0.3.1
  Downloading inflection-0.5.1-py2.py3-none-any.whl (9.5 kB)
Requirement already satisfied: numpy>=1.8 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: more-itertools in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from q
Requirement already satisfied: requests>=2.7.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
Installing collected packages: inflection, quandl
Successfully installed inflection-0.5.1 quandl-3.7.0
```

Carregando os dados

```
# baixa os dados da exchange Kraken, até o período atual.
import quandl
data = quandl.get('BCHARTS/KRAKENUSD', returns='pandas')
```

Entendendo os dados

```
#exibe as primeiras linahs
data.head()
```

Date	Open	High	Low	Close	Volume (BTC)	Volume (Currency)	Weighted Price
2014-01-07	874.67040	892.06753	810.00000	810.00000	15.622378	13151.472844	841.835522
2014-01-08	810.00000	899.84281	788.00000	824.98287	19.182756	16097.329584	839.156269
2014-01-09	825.56345	870.00000	807.42084	841.86934	8.158335	6784.249982	831.572913

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2713 entries, 2014-01-07 to 2021-06-20
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Open               2713 non-null   float64
 1   High               2713 non-null   float64
 2   Low                2713 non-null   float64
 3   Close              2713 non-null   float64
 4   Volume (BTC)       2713 non-null   float64
 5   Volume (Currency)  2713 non-null   float64
 6   Weighted Price     2713 non-null   float64
dtypes: float64(7)
memory usage: 169.6 KB
```

```
# verifica os últimos dados. Repare na data. Deve ter dados atuais (Jun / 2021).
data.tail()
```

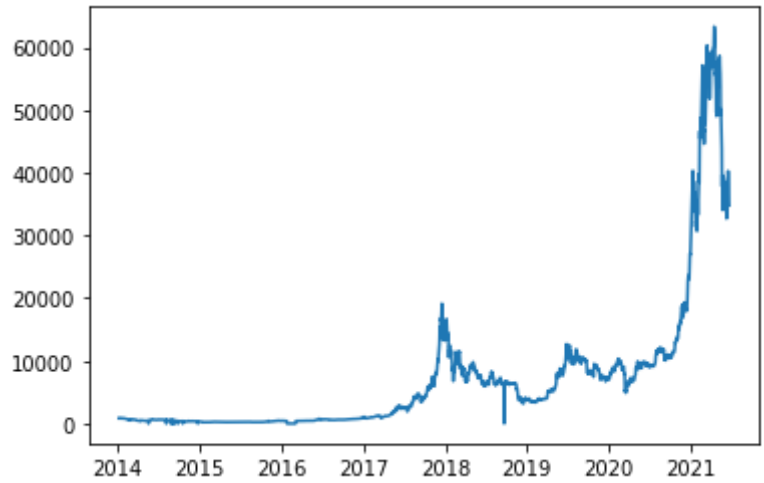
Date	Open	High	Low	Close	Volume (BTC)	Volume (Currency)	Weighted Price
2021-06-16	40167.3	40493.0	38120.0	38337.1	6487.206888	2.539206e+08	39141.737747
2021-06-17	38337.1	39561.4	37405.0	38078.2	6003.220618	2.307246e+08	38433.468618
2021-06-18	38078.2	38193.1	35126.0	35824.0	6558.468890	2.409217e+08	36734.445103

Repare que temos dados de abertura do pregão, fechamento, valor mais alto, valor mais baixo, volume diário do bitcoin e de todas as criptomoedas combinadas. E também, temos os preço ponderado pelos valores de compra/venda de um período, que em nosso caso é diário. Para facilitar, vamos usar o valor ponderado.

Plotando os dados

```
# imprima os dados
pyplot.plot(data['Weighted Price'])
```

```
[<matplotlib.lines.Line2D at 0x7f7c098801d0>]
```

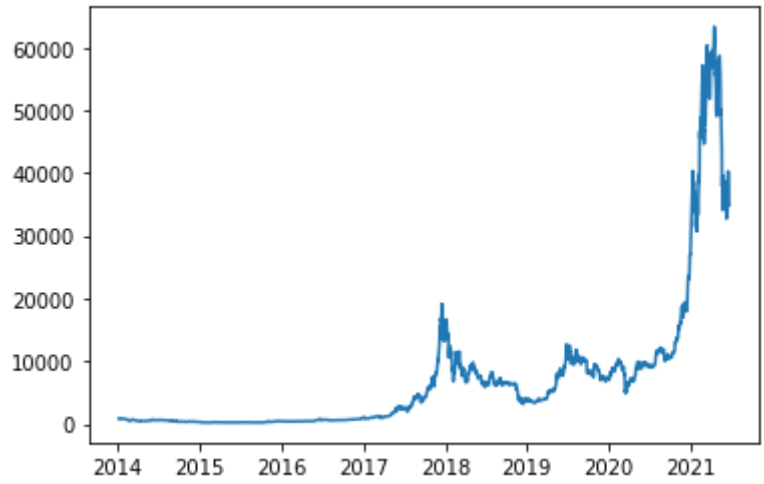


Pré-processamento dos dados

```
#existem alguns pontos com valor zero (outliers), vamos trocar por NaN e depois chamar um
data['Weighted Price'].replace(0, np.nan, inplace=True)
data['Weighted Price'].fillna(method='ffill', inplace=True)
```

```
# imprima novamente e observe que não existe mais estes outliers.
pyplot.plot(data['Weighted Price'])
```

```
[<matplotlib.lines.Line2D at 0x7f7c097fd690>]
```




```
# vamos usar o preço ponderado como entrada para nossa rede recorrente
# como já vimos, eh sempre bom normalizar os dados para ajudar na convergência do treiname
# Normaliza na faixa entre [0 e 1]
from sklearn.preprocessing import MinMaxScaler
```

```
values = data['Weighted Price'].values.reshape(-1,1)
values = values.astype('float32')
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)


# vamos deixar 70% para treino e 30% para teste. Observe que temos mais de 6 anos de dados
train_size = int(len(scaled) * 0.7)
test_size = len(scaled) - train_size
train, test = scaled[0:train_size,:], scaled[train_size:len(scaled),:]
print(len(train), len(test))
```

```
1899 814
```

Vamos considerar uma janela de um único dia para efetuar a predição. Para isso, use a função create_dataset(..) e deixe o parâmetro look_back=1. O parâmetro look_back controla a quantidade de dados que vai fazer parte da janela de entrada para a rede. Estude e entenda o que a função faz.

```
#função para criar os conjuntos de dados de treino
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    print(len(dataY))
    return np.array(dataX), np.array(dataY)


# entra com janela de 1 único valor
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

```
1898
813
```

```
trainX.shape
```

```
(1898, 1)
```

```
# reshape para formato de entrada da rede neural (instancias, 1, 1)
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))


trainX.shape
```

```
(1898, 1, 1)
```

Projeto de uma rede recorrente

Projete uma rede recorrente, usando alguma das camadas abaixo:

1. Item da lista
2. Item da lista

```
tf.keras.layers.LSTM
tf.keras.layers.GRU
tf.keras.layers.RNN
```

As camadas recorrentes (LSTM, GRU, RNN) podem ser bidirecionais ou simpels, por exemplo, uma camada LSTM com 32 unidades e bidirecional:

```
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32))
```

Você também pode usar dropout e camadas densas em seu modelo.

Experimente três arquiteturas (rasas e profundas) e pelo menos dois algoritmos de otimização. Documente os resultados em uma tabela e anexe.

Por exemplo, você pode usar um modelo raso como o abaixo:

```
np.random.seed(42)
tf.random.set_seed(42)



model_1 = Sequential([
  LSTM(128,input_shape=[None,1]),
  Dense(1)
])
```

Com uma função de custo **Mean Square Error** e o algoritmo de otimização **ADAM**:

```
model_1.compile(loss='mse',optimizer = 'adam')
```

Ou pode usar um modelo profundo, mais complexo como o abaixo:

```
model = Sequential()
model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)))
model.add(Dense(units = 64, activation='relu'))
model.add(Dropout(dropout_rate))
model.add(Dense(units = 1))
```

O **erro médio quadrático** deste último modelo, com o otimizador **ADAM** e **erro médio quadrático** como função de custo deve resultar em:

```
Test Root Mean Square Error (RMSE): 380.139
```

Observações

1. **Seu RMSE pode ser diferente devido aos dados usados.**
2. **Use modelos diferentes dos de exemplo!**

ToDo: Projetando os seus modelos (30pt)

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

Modelo 1:

```
tf.keras.layers.LSTM
tf.keras.layers.GRU
tf.keras.layers.RNN
```

```
keras.layers.recurrent.RNN
```

```
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32))
```

```
<keras.layers.wrappers.Bidirectional at 0x7f7c09244e50>
```

```
np.random.seed(42)
tf.random.set_seed(42)
```

```
model_1 = Sequential()
model_1.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)))
model_1.add(Dense(units = 64, activation='relu'))
#model_1.add(Dropout(dropout_rate))
model_1.add(Dense(units = 1))
#model_1.summary()
```

```
model_1.compile(loss='mse',optimizer = 'adam')
```

Modelo 2:

```
tf.keras.layers.Bidirectional(tf.keras.layers.GRU(32))
```

```
<keras.layers.wrappers.Bidirectional at 0x7f7c09866550>
```

```
tf.keras.layers.GRU
```

```
model_2 = keras.Sequential()
#model_2.add(layers.Embedding(input_dim=1000, output_dim=64))

# The output of GRU will be a 3D tensor of shape (batch_size, timesteps, 256)
model_2.add(layers.GRU(64, return_sequences=True))
#model_2.add(Dense(units = 64, activation='relu'))
# The output of SimpleRNN will be a 2D tensor of shape (batch_size, 128)
model_2.add(layers.SimpleRNN(64))

model_2.add(Dense(units = 1))

#model_2.summary()
```

```
model_2.compile(loss='mse',optimizer = 'adam')
```

Modelo 3:

```
model_3 = keras.Sequential()
```

```
model_3.add(Dense(units = 64, activation='relu'))

model_3.add(layers.SimpleRNN(128))
```

```
#model_3.add(layers.Dense(10))

model_3.add(Dense(units = 1))
```

```
#model_3.summary()
```

```
model_3.compile(loss='mse',optimizer = 'adam')
```

ToDo: Função de custo (10pt)

Como é um problema de regressão, usaremos funções de custo apropriadas. Você pode usar, por exemplo, *Mean Absolute Error* (mae) ou *Mean Squared Error* (mse).

ToDo: Estude as funções de custo MAE e MSE. Qual das duas funções você usaria. Justifique sua escolha. Repare que vamos avaliar os modelos pela métrica *Root Mean Square Error* (RMSE).

ToDo: Função para treinar o seu modelo (15pt)

```
# Função para treinar o modelo
def train_model(model, loss, optimizer, trainX, trainY):
  # Compile o modelo : atenção para a função de CUSTO. Abaixo um exemplo de uso da 'mae'
  model.compile(loss=loss, optimizer=optimizer)

  #treine o modelo
  history = model.fit(trainX,trainY,epochs=20)

  # plote a curva de custo
  pyplot.plot(history.history['loss'], label='train')
  #pyplot.plot(history.history['val_loss'], label='test')
  pyplot.legend()

  pyplot.show()
```

Função para avaliar o seu modelo

```
# Avaliando o modelo treinado
def evaluate_model(model, testX, testY):

  # plote as curvas, valor real e valor predito no mesmo gráfico
  yhat = model.predict(testX)
  pyplot.title('Curva do valor real e valor predito na escala usado no treino')
  pyplot.plot(yhat, label='predict')
  pyplot.plot(testY, label='true')
  pyplot.legend()
  pyplot.show()

  # os valores foream normalizados para o treinamento.
  # Veja que para fazer sentido, eles devem voltar para a escala original.
  # Volta para escala em US dollar :
  yhat_inverse = scaler.inverse_transform(yhat.reshape(-1, 1))
  testY_inverse = scaler.inverse_transform(testY.reshape(-1, 1))

  # calcula o RMSE
```

```
rmse = sqrt(mean_squared_error(testY_inverse, yhat_inverse))
print('Test RMSE: %.3f' % rmse)

# valor em US dollar
pyplot.title('Curva do valor real e valor predito em US dollar')
pyplot.plot(yhat_inverse, label='predict')
pyplot.plot(testY_inverse, label='actual', alpha=0.5)
pyplot.legend()
pyplot.show()
```

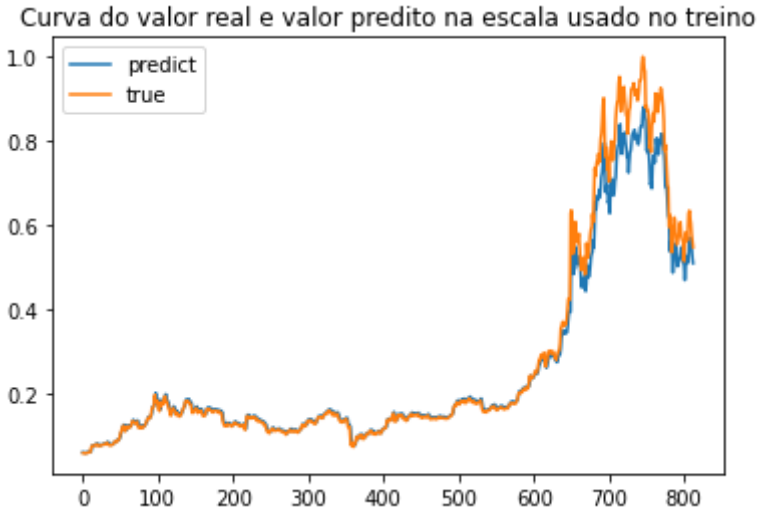
ToDo: Treinando e avaliando o seu modelo (15pt)

Modelo 1

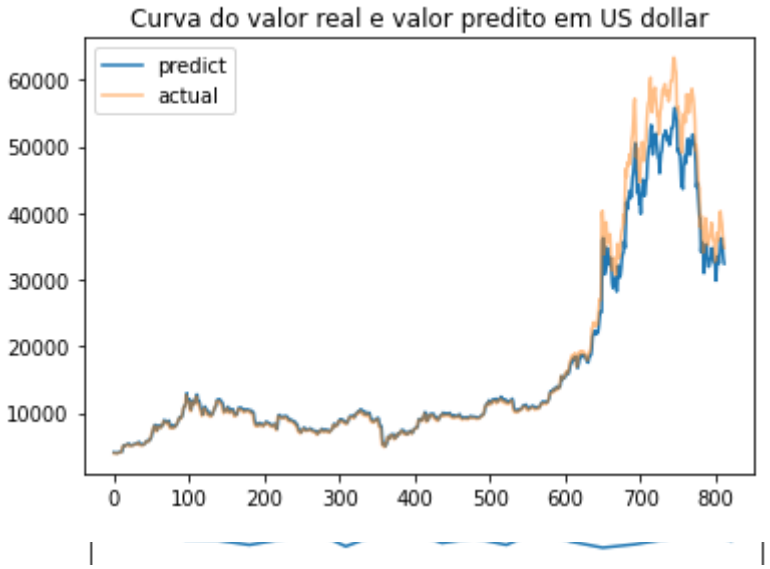
```
train_model(model_1,'mae','adam',trainX,trainY)
```

```
Epoch 1/20
60/60 [==============================] - 8s 7ms/step - loss: 0.0198
Epoch 2/20
60/60 [==============================] - 0s 8ms/step - loss: 0.0043
Epoch 3/20
60/60 [==============================] - 1s 9ms/step - loss: 0.0020
Epoch 4/20
60/60 [==============================] - 0s 7ms/step - loss: 0.0020
Epoch 5/20
60/60 [==============================] - 0s 7ms/step - loss: 0.0018
Epoch 6/20
60/60 [==============================] - 0s 8ms/step - loss: 0.0020
Epoch 7/20
60/60 [==============================] - 0s 8ms/step - loss: 0.0024
Epoch 8/20
60/60 [==============================] - 0s 6ms/step - loss: 0.0017
Epoch 9/20
```

```
evaluate_model(model_1,testX,testY)
```



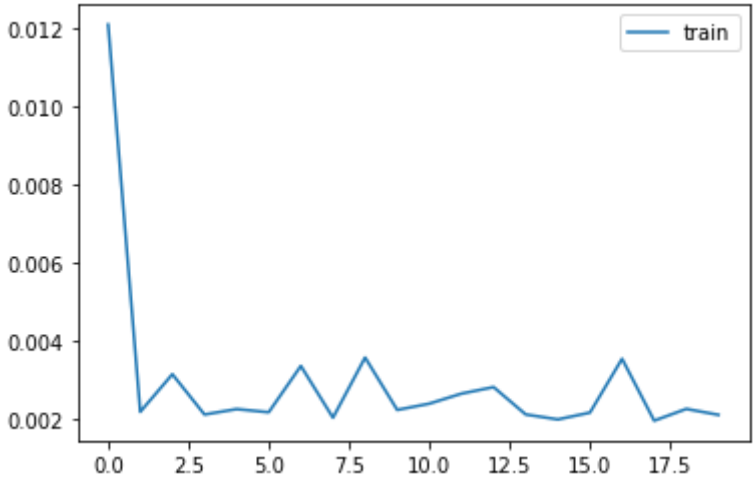

```
Test RMSE: 2571.262
```



Modelo 2

```
train_model(model_2,'mae','adam',trainX,trainY)
```

```
Epoch 1/20
60/60 [==============================] - 8s 7ms/step - loss: 0.0121
Epoch 2/20
60/60 [==============================] - 0s 7ms/step - loss: 0.0022
Epoch 3/20
60/60 [==============================] - 0s 8ms/step - loss: 0.0031
Epoch 4/20
60/60 [==============================] - 1s 9ms/step - loss: 0.0021
Epoch 5/20
60/60 [==============================] - 0s 6ms/step - loss: 0.0022
Epoch 6/20
60/60 [==============================] - 0s 7ms/step - loss: 0.0022
Epoch 7/20
60/60 [==============================] - 0s 7ms/step - loss: 0.0033
Epoch 8/20
60/60 [==============================] - 0s 7ms/step - loss: 0.0020
Epoch 9/20
60/60 [==============================] - 0s 5ms/step - loss: 0.0036
Epoch 10/20
60/60 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 11/20
60/60 [==============================] - 0s 4ms/step - loss: 0.0024
Epoch 12/20
60/60 [==============================] - 0s 4ms/step - loss: 0.0026
Epoch 13/20
60/60 [==============================] - 0s 4ms/step - loss: 0.0028
Epoch 14/20
60/60 [==============================] - 0s 4ms/step - loss: 0.0021
Epoch 15/20
60/60 [==============================] - 0s 4ms/step - loss: 0.0020
Epoch 16/20
60/60 [==============================] - 0s 4ms/step - loss: 0.0021
Epoch 17/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0035
Epoch 18/20
60/60 [==============================] - 0s 4ms/step - loss: 0.0019
Epoch 19/20
60/60 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 20/20
60/60 [==============================] - 0s 4ms/step - loss: 0.0021
```




```
evaluate_model(model_2,testX,testY)
```

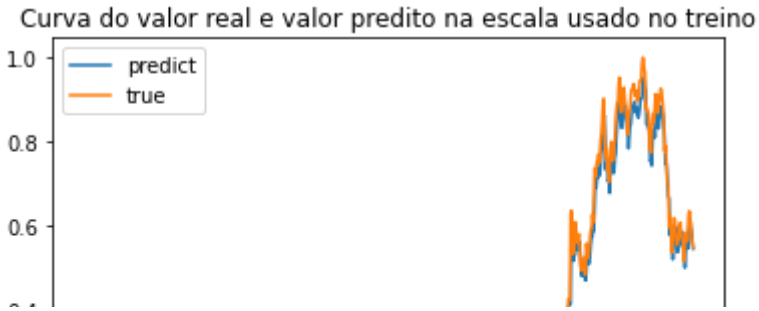

```
Test RMSE: 1032.817
```

Modelo 3

```
train_model(model_3,'mae','adam',trainX,trainY)
```

```
Epoch 1/20
60/60 [==============================] - 1s 3ms/step - loss: 0.0061
Epoch 2/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 3/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 4/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 5/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 6/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0023
Epoch 7/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 8/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0022
Epoch 9/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 10/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 11/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 12/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0017
Epoch 13/20
60/60 [==============================] - 0s 2ms/step - loss: 0.0021
Epoch 14/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0022
Epoch 15/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0022
Epoch 16/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 17/20
60/60 [==============================] - 0s 3ms/step - loss: 0.0020
```

```
evaluate_model(model_3,testX,testY)
```



Prevendo o próximo dia

```
# https://www.tensorflow.org/tutorials/structured_data/time_series
def create_time_steps(length):
  time_steps = []
  for i in range(-length, 0, 1):
    time_steps.append(i)
  return time_steps

def baseline(history):
  return np.mean(history)

def show_plot(plot_data, delta, title):
  labels = ['History', 'True Future', 'Model Prediction']
  marker = ['.-', 'rx', 'go']
  time_steps = create_time_steps(plot_data[0].shape[0])
  if delta:
    future = delta
  else:
    future = 0

  pyplot.title(title)
  for i, x in enumerate(plot_data):
    if i:
      pyplot.plot(future, plot_data[i], marker[i], markersize=10,
               label=labels[i])
    else:
      pyplot.plot(time_steps, plot_data[i].flatten(), marker[i], label=labels[i])
  pyplot.legend()
  pyplot.xlim([time_steps[0], (future+5)*2])
  pyplot.xlabel('Time-Step')
  return pyplot
```

ToDo: Função para predizer o próximo dia (15pt)

```
def predict_next_day(model, testX, testY):
  # os valores foream normalizados para o treinamento.
  # Veja que para fazer sentido, eles devem voltar para a escala original.
  # Volta para escala em US dollar :
  yhat_inverse = scaler.inverse_transform(model_1.predict(testX).reshape(-1, 1))
  testY_inverse = scaler.inverse_transform(testY.reshape(-1, 1))
```

```
# na base de teste, plote até a instância 200 e tente predizer a instância futura: 201
# use a função show_plot
show_plot([testY_inverse[0:200], yhat_inverse[201],
           baseline(testY_inverse[201])], 1, 'Predição do dia seguinte')

print('valor predito do dia 201: ', yhat_inverse[201])
print('Valor real do dia 201: ', testY_inverse[201])
```

Modelo 1:

```
predict_next_day(model_1,testX,testY)
```

```
valor predito do dia 201:  [8380.734]
Valor real do dia 201:  [8431.232]
```

Modelo 2:

```
predict_next_day(model_2,testX,testY)
```

```
valor predito do dia 201:  [8380.734]
Valor real do dia 201:  [8431.232]
```

Modelo 3:

```
predict_next_day(model_3,testX,testY)
```

```
valor predito do dia 201:  [8380.734]
Valor real do dia 201:  [8431.232]
```

ToDo: Resultados (15pt)

Coloque os valores dos modelos em uma tabela. Em cada coluna, informe qual a função de custo utilizada, qual otimizador e o erro na partição de teste em RMSE.

```
| Modelo   | Função de Custo | Otimizador |
| Modelo 1 |    mae          |     adam   |
| Modelo 2 |    mae          |     adam   |
| Modelo 3 |    mae          |     adam   |
```

✓ 0s conclusão: 13:10