UNIRIO

# TAA1

## Trabalho Final

**Breno P. M. Baronte**

# 1. Motivação

Nos últimos anos, a bolsa de valores brasileira apresentou um crescimento considerável e consecutivo no número de CPFs cadastrados. Passando de 619.625 em 2017 para 3.173.411 em 2020.

Devido a entrada de novos investidores na bolsa de valores, é comum que os novos ingressantes tenham um processo de aprendizado, que dure um tempo diferente para cada um, para investirem da melhor forma a partir de cada estratégia adotada.

Concomitantemente, é possível observar grandes avanços na área de aprendizagem profunda nos últimos anos. Em 2012 tivemos um grande marco na área de aprendizagem profunda, e mais especificamente na área de visão de máquina, com a rede neural AlexNet que foi desenvolvida utilizando o recém criado banco de imagens ImageNet obtendo uma acurácia nunca antes vista.

Após esse marco, a área de inteligência artificial e aprendizagem profunda voltaram a ganhar um destaque maior a ponto de que hoje em dia temos a aprendizagem profunda permeada nas nossas tecnologias de uso rotineiro. Alguns exemplos são: Amazon com a Alexa, Apple com a Siri,  Google Assistant, entre outros.

Utilizando as possibilidades que a aprendizagem profunda nos proporciona, a motivação deste trabalho é criar uma rede neural capacitada a ser mais uma fonte de suporte à decisão do investidor na hora operar no mercado de ações, tendo em vista que a rede neural não deve ser utilizada como única fonte de suporte para esta tomada de decisão já que é sabido que o mercado é muito complexo.

## 2. Definição da Tarefa

A categoria do problema deste trabalho se trata de um problema de regressão. A partir de valores do preço de uma ação em série, iremos tentar prever qual será o próximo valor.

Usaremos mais de um modelo para testarmos abordagens diferentes e posteriormente analisarmos os resultados. Também mostraremos como modelos iguais se comportam com diferentes treinamentos e quantidade de dados, e os efeitos dessa mudança nos resultados.

Para calcularmos o quão próximos os valores das predições dos modelos chegaram em relação aos valores reais iremos utilizar o desvio quadrático médio. O que nos ajudará a calcular a quantidade de “resíduo” da predição de cada modelo e tentaremos atingir o menor valor possível.

## 3. Análise Exploratória do Dataset

O dataset utilizado neste trabalho disponibiliza os dados das ações da Microsoft Corporation que são comercializadas na bolsa de valores americana chamada NASDAQ. Os dados datam de 02/01/1998 até o dia 02/01/2019 e somente os dias úteis apresentam dados já que não há operações na bolsa de valores em dias que não são úteis.

Os dados foram separados em seis colunas e faremos uma breve descrição de cada uma delas.

| Coluna | Descrição do dado armazenado |
|---|---|
| timestamp | Data dos registros para esta linha no formato ano-mês-dia. |
| open | O primeiro valor comercializado pela ação no dia. |
| high | Valor mais alto comercializado pela ação no dia. |
| low | Menor valor comercializado pela ação no dia. |
| close | Último valor comercializado pela ação no dia. |
| volume | Número de transações de compra e venda para a ação no dia. |

Para o treinamento dos nossos modelos iremos utilizar somente duas colunas. São elas a coluna “open” e “high”. O primeiro valor comercializado pela ação no dia é um bom indicador de preço momentâneo, mas também poderíamos utilizar o último valor comercializado no dia. Esta escolha foi experimental.

Além do valor de abertura, como é conhecido o “open”, utilizaremos o maior valor comercializado por aquela ação no dia pois esse valor nos mostra o quão “aquecido” ou movimentado o mercado estava naquele dia.

Para utilização do dataset nos nossos modelos, iremos dividir o dataset em dois. A primeira parte será com 65% dos dados e será o dataset utilizado para treinar o nosso modelo, enquanto a segunda parte ficará com os 35% restantes dos dados e servirá como dataset utilizado para testes.

## 4. Modelos Utilizados

Dois modelos foram criados para este trabalho. O primeiro modelo foi chamado de complexo e o segundo de simples, ou como estão descritos nos notebooks gerados na plataforma google colab, “complex” e “simple”.

Inicialmente mostraremos o modelo complexo e então mostraremos as diferenças entre este modelo e o modelo simples.

### 4.1 Modelo Complexo

A etapa inicial é a autenticação no Google drive para então obter os datasets de treinamento e de teste. Inicialmente obteremos somente o dataset de treinamento.

```
[ ]  # Step 0 - importing the dataset file
     !pip install pydrive
     from pydrive.auth import GoogleAuth
     from pydrive.drive import GoogleDrive
     from google.colab import auth
     from oauth2client.client import GoogleCredentials

     auth.authenticate_user()
     gauth = GoogleAuth()
     gauth.credentials = GoogleCredentials.get_application_default()
     drive = GoogleDrive(gauth)

     file_id = '12bWmhsQoeAOk2JLCOzlRsk0IMQaKD6s2'
     downloaded = drive.CreateFile({'id': file_id})
     downloaded.GetContentFile('dataset_daily_msft_training.csv')
```

Importaremos três bibliotecas. Numpy para operações e estruturas matemáticas que serão utilizadas. Matplot.pyplot para apresentarmos um gráfico do resultado da predição do nosso modelo e pandas para algumas operações como ler o arquivo do dataset em formato csv.

```
[ ]  import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
```

Iremos obter os dados do nosso dataset de treinamento e então selecionaremos as colunas que serão utilizadas no treinamento, as colunas “open” e “high”.

```
[ ]  dataset_train = pd.read_csv('dataset_daily_msft_training.csv')
     training_set = dataset_train.iloc[:, 1:2].values
```

Com o nosso dados para treinamento selecionados, iremos pré-processar esses dados de forma que os deixaremos em uma escala entre 0 e 1.

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
```

Deixaremos nossos dados no formato de um array de três dimensões já que este é o padrão usual para uma LSTM(Long short-term memory). O “60” abaixo utilizado é o número de dias que estaremos olhando para trás para treinar cada registro, enquanto 3435 é o número de linhas do nosso dataset de treinamento. Então criaremos primeiro dados com 60 dias e converteremos isso em um array e na hora de redimensionar deixaremos como: (dados dos 60 dias, 60, 1).

```
X_train = []
y_train = []

for i in range(60, 3435):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 0])

X_train, y_train = np.array(X_train), np.array(y_train)

X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

Com nossos dados pré-processados e redimensionados, podemos iniciar a criação do nosso modelo. Para isso iremos importar alguns módulos do Keras.

```
[ ]  from keras.models import Sequential
     from keras.layers import Dense
     from keras.layers import LSTM
     from keras.layers import Dropout
```

Nosso modelo contará com 4 camadas LSTM, contando que para cada camada LSTM será seguida de uma camada Dropout que desperdiçará os resultados das camadas LSTM em 20% para evitar possíveis overfitting ou underfitting. Nossa última camada será do tipo Dense, por questões experimentais, com somente uma unidade de saída.

```
regressor = Sequential()

regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

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

Nosso algoritmo de otimização será o Adam e o de perda será o do erro quadrático médio por questões experimentais.

```
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

O treinamento do modelo complexo contará com 100 epochs e o tamanho do batch de 32 unidades.

```
regressor.fit(X_train, y_train, epochs = 100, batch_size = 32)
```

Após o treinamento, iremos fazer o modelo predizer os valores do nosso dataset de teste e então iremos comparar os valores reais com os valores previstos pelo modelo. Para começar este processo precisamos carregar o arquivo do dataset de teste.

```
file_id = '10oiV6jvHGtiXgDDJmMByxBhiyYKjaQCw'
downloaded = drive.CreateFile({'id': file_id})
downloaded.GetContentFile('dataset_daily_msft_test.csv')
```

Em seguida leremos os valores deste arquivo, como feito com o dataset de treinamento e então o guardaremos em um array.

```
dataset_test = pd.read_csv('dataset_daily_msft_test.csv')
real_stock_price = dataset_test.iloc[:, 1:2].values
```

Com o dataset carregado já é possível previsões com o modelo, mas antes disso teremos que realizar o pré-processamento dos dados assim como foi feito com o dataset de treinamento. Após termos os dados do modelo, precisaremos reverter a transformação que fizemos para que eles ficassem na escala entre 0 e 1.

```
dataset_total = pd.concat((dataset_train['open'], dataset_test['open']), axis = 0)

inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)

X_test = []

for i in range(60, 1909): # 1909 is dataset_test x shape + 60, this 60 is days to look back
    X_test.append(inputs[i-60:i, 0])

X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

Com os dados reais e os dados previstos iremos identificar o quão efetiva nossa previsão foi ao calcularmos o valor do desvio quadrático médio. Lembrando que quanto menor o valor, melhor.

```
# Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(real_stock_price, predicted_stock_price))
```

Além disso, para termos uma visualização mais palpável, faremos o gráfico com os valores reais e os valores previstos. Faremos um ajuste antes de mostrar o gráfico para tornar seu tamanho maior já que como a quantidade de dados é muito grande isso foi necessário para melhorar a visualização.

```
plt.rcParams['figure.figsize'] = (25,15)
```

```
plt.plot(real_stock_price, color = 'black', label = 'MSFT Stock Price')
plt.plot(predicted_stock_price, color = 'orange', label = 'Predicted MSFT Stock Price', linestyle = 'solid')
plt.title('MSFT Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('MSFT Stock Price')
plt.legend()
plt.show()
```

## 4.2 Modelo Simples

O modelo simples difere muito pouco do modelo complexo quando estamos falando do código. As diferenças são, usamos cinco camadas no nosso modelo ao invés de nove.

```
regressor = Sequential()

regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

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

E o treinamento conta com 50 epochs ao invés de 100.

```
regressor.fit(X_train, y_train, epochs = 50, batch_size = 32)
```

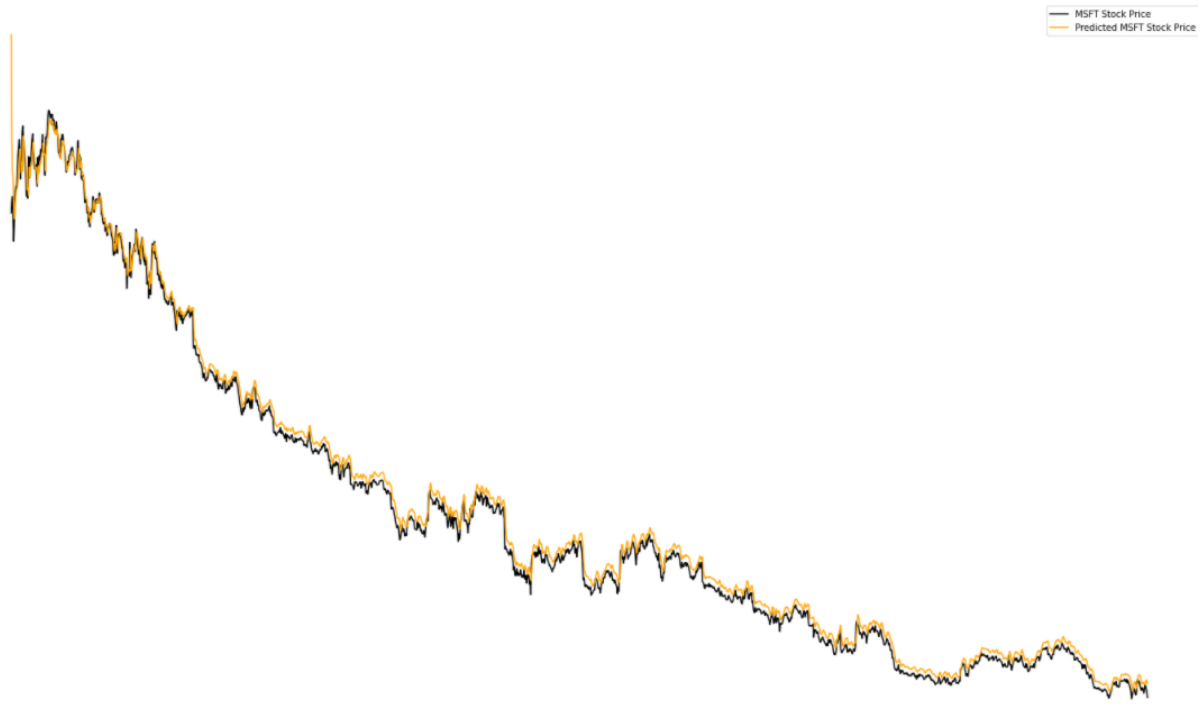
Esse segundo modelo foi criado para verificarmos se um número maior de camadas e de treinamento iria aumentar ou diminuir o nosso desvio quadrático médio para este caso específico.

## 5. Experimentos

Foram realizados quatro experimentos, utilizando os dois modelos citados anteriormente em conjunto com o dataset inteiro, mas também em conjunto com a metade do dataset, respeitando as proporções de treinamento e teste, para observarmos os efeitos de como a quantidade de dados influencia nossos resultados.

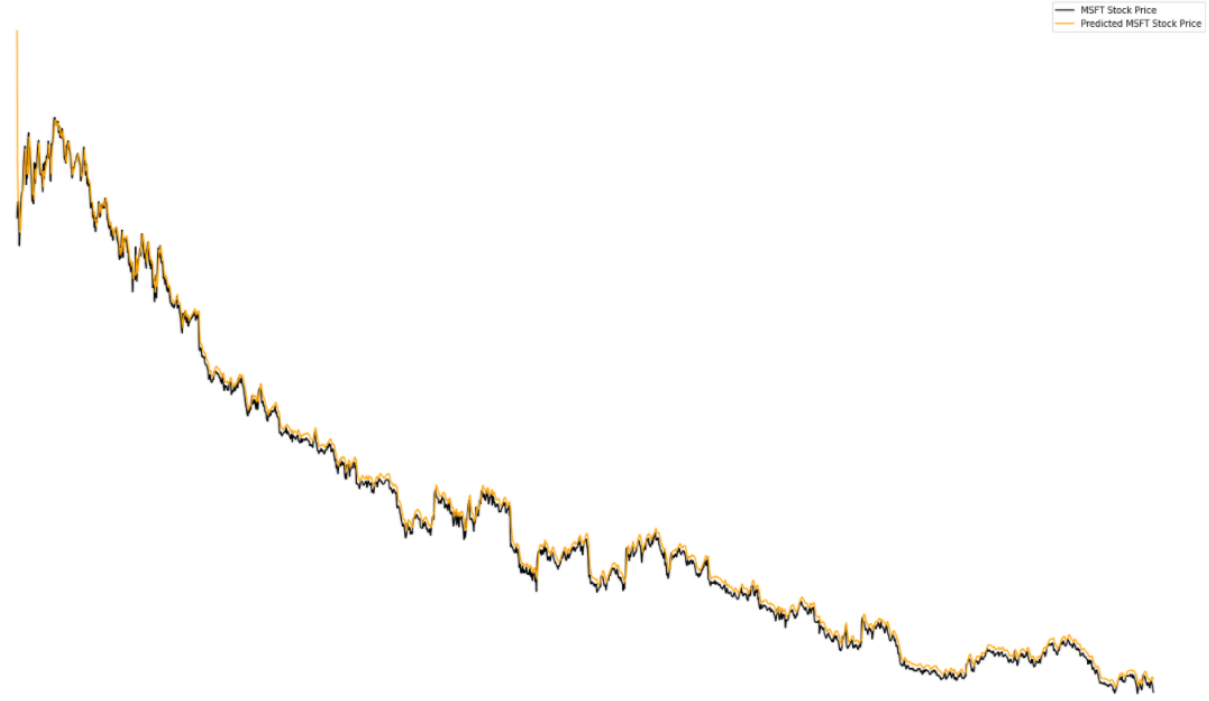
### 5.1 Modelo complexo com dataset inteiro:

https://colab.research.google.com/drive/1MreVESOO-B_VK1GRfXq4FvaEJ2asCsfs?usp=sharing



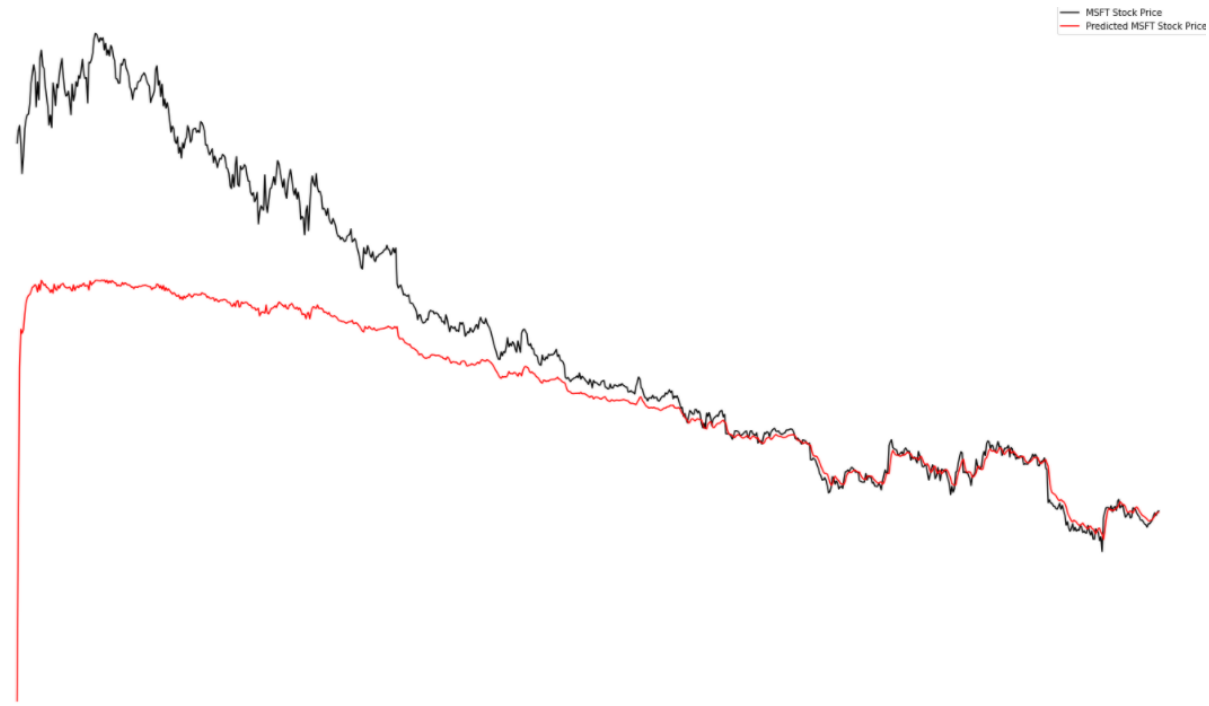
5.2 Modelo simples com dataset inteiro:

https://colab.research.google.com/drive/1M3e_QRPxWD_ZghpT0CBZOx0RlflBzujr?usp=sharing



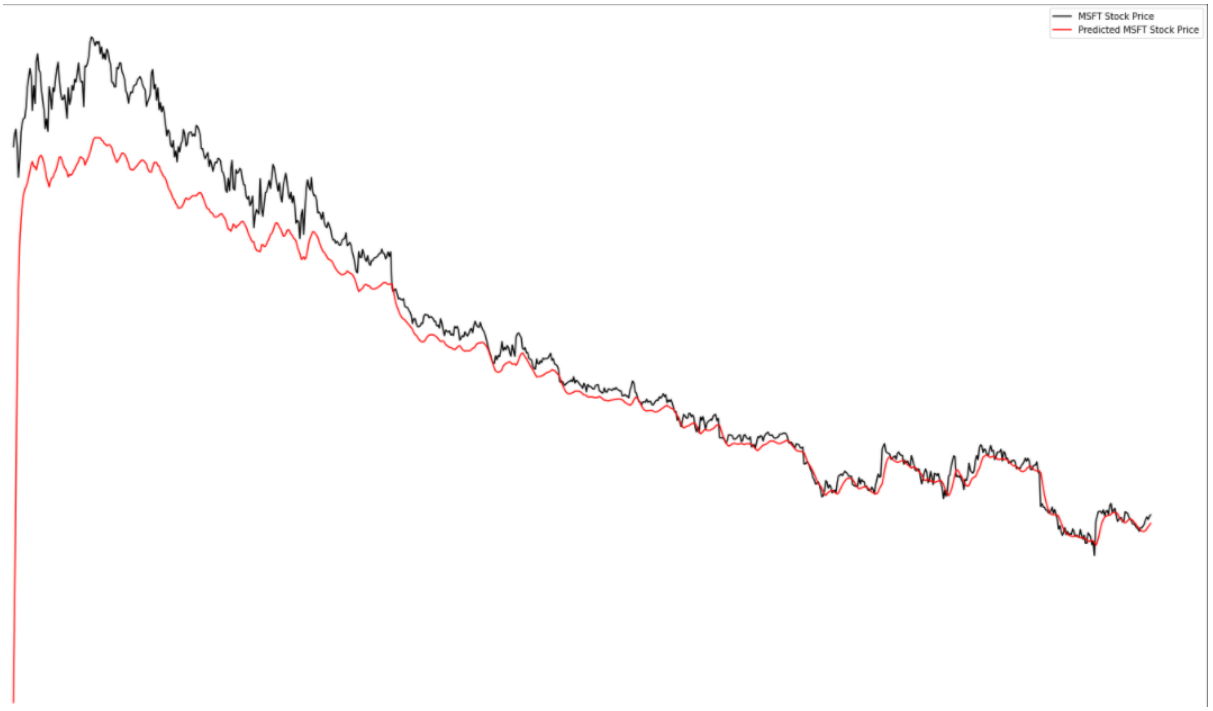

5.3 Modelo complexo com metade do dataset:

https://colab.research.google.com/drive/1mAM92FfJnSvvh_Fi_g7snfAob_sF-d29?usp=sharing



5.4 Modelo simples com metade do dataset:

https://colab.research.google.com/drive/18lPgPajV3MIYYUKPn_NiGyxAk0m8ZZJT?usp=sharing




Através dos modelos obtivemos:

| Modelo | Dataset utilizado | Erro quadrático médio |
|---|---|---|
| Complexo | inteiro | 1.5996510545553395 |
| Simples | inteiro | 1.4422432957359343 |
| Complexo | metade | 13.898225266553812 |
| Simples | metade | 6.677246417422629 |

## 6. Análise dos Experimentos

Em um primeiro momento, intuitivamente era achado que a rede complexa iria conseguir um melhor desempenho nos dois experimentos pois talvez ela poderia aprender padrões de subida e descida de forma mais eficaz. Todavia vimos que isso não foi o que aconteceu.

O melhor resultado, a julgar pelo desvio quadrático médio, foi o da rede simples utilizando o dataset inteiro. Foi também possível notar como o número de dados influencia na acurácia do nosso modelo proporcionalmente, o que já era esperado e foi confirmado.

## 7. Conclusão

Um número maior de epochs no treinamento combinado com um maior número de camadas da rede complexa não foi mais eficiente do que nossa rede simples, pelo contrário.

Algumas coisas ainda podem ser melhoradas no nosso modelo, como por exemplo as linhas verticais nos gráficos exibidos. Isso está ocorrendo por um motivo ainda não identificado, porém não é um comportamento esperado.

Mais experimentos também podem ser realizados utilizando as redes simples e complexas, por exemplo, fazendo com que as duas treinem com o mesmo número de epochs e compararmos seus desvios.

## 8. Referências Bibliográficas


https://www.kaggle.com/darkknight91/microsoft-stock-price-daily-19982018

https://www.youtube.com/watch?v=H6du_pfuznE

https://www.youtube.com/watch?v=QIUxPv5PJOY

https://towardsdatascience.com/adam-optimization-algorithm-1cdc9b12724a

https://www.youtube.com/watch?v=hJI0wZV7VnA

https://matplotlib.org/tutorials/colors/colors.html

https://matplotlib.org/3.1.0/gallery/lines_bars_and_markers/linestyles.html

https://www.youtube.com/watch?v=dKBKNOn3gCE&t=376s

https://www.kdnuggets.com/2018/11/keras-long-short-term-memory-lstm-model-predict-stock-prices.html

https://en.wikipedia.org/wiki/Root-mean-square_deviation

https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network

https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/