@Odemir Depieri Jr

# Prevendo preço de ações da Tesla


## com Redes Neurais Recorrentes

Keras

scikit learn

pandas

plotly

NumPy

@Odemir Depieri Jr

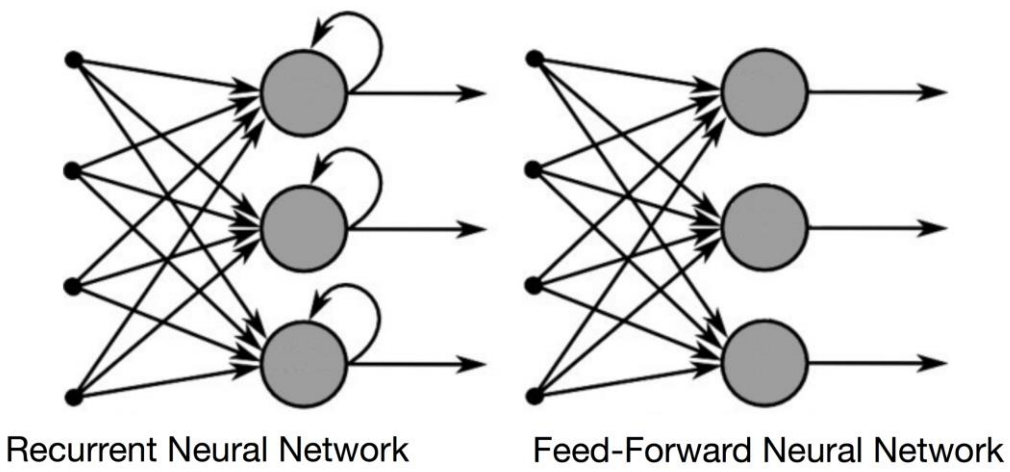
# Redes Neurais Recorrentes

## RRN

As **redes recorrentes** são um tipo de **rede neural artificial** projetada para reconhecer padrões em sequências de dados, como texto, genomas, caligrafia, palavra falada ou dados de séries numéricas que emanam de sensores, bolsas de valores e agências governamentais. Esses algoritmos consideram tempo e sequência, eles têm uma dimensão temporal.

As redes recorrentes, por outro lado, tomam como entrada **não apenas o exemplo de entrada atual** que veem, mas também o que **perceberam anteriormente** no tempo.

*Diferença entre uma rede neural vs rede recorrente*




## Artigos

https://www.deeplearningbook.com.br/redes-neurais-recorrentes/

https://www.monolitonimbus.com.br/modelo-sequencial-do-keras/

https://keras.io/api/optimizers/

https://qastack.com.br/programming/38714959/understanding-keras-lstms

https://medium.com/luisfredgs/an%C3%A1lise-de-sentimentos-com-redes-neurais-recorrentes-lstm-a5352b21e6aa

## Vídeos

https://www.youtube.com/watch?v=bDDP0m4jjH0

https://www.youtube.com/watch?v=bIcadBu--u8&t=4597s

**Vamos utilizar uma base de dados da Kaggle**

https://www.kaggle.com/varpit94/tesla-stock-data-updated-till-28jun2021

**Vamos importar as bibliotecas necessárias**

```
# Lib para modelagem de Dados
import pandas as pd
# Lib para uso de vetores
import numpy as np
# Lib para funções matematicas
import math
# Lib para visualização gráfica
import plotly.graph_objects as Dash
# Libs para uso de Machine Learning do Keras
from keras.models import Sequential
from keras.layers import Dense, LSTM
# Lib para pre-processamento
from sklearn.preprocessing import MinMaxScaler
```

**Importar os dados**

```
[2] # Lendo a Base de Dados
    Base_Dados = pd.read_csv('TSLA.csv')
    # Verificando os primeiros registros
    Base_Dados.head()
```

| | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| **0** | 2010-06-29 | 3.800 | 5.000 | 3.508 | 4.778 | 4.778 | 93831500 |
| **1** | 2010-06-30 | 5.158 | 6.084 | 4.660 | 4.766 | 4.766 | 85935500 |
| **2** | 2010-07-01 | 5.000 | 5.184 | 4.054 | 4.392 | 4.392 | 41094000 |
| **3** | 2010-07-02 | 4.600 | 4.620 | 3.742 | 3.840 | 3.840 | 25699000 |
| **4** | 2010-07-06 | 4.000 | 4.000 | 3.166 | 3.222 | 3.222 | 34334500 |

Nessa base de dados temos os dados das ações da Tela:

- Data
- Abertura
- Maior
- Menor
- Fechamento
- Volume

## Vamos incluir a data como index da nossa tabela

```
[3]  # Definir a coluna Data como Index do DataSet
     Base_Dados = Base_Dados.set_index('Date')

     # Retirando qualquer valor nulo existente nos Dados
     Base_Dados = Base_Dados.dropna()

     # Verificando
     Base_Dados.head()
```

| Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| **2010-06-29** | 3.800 | 5.000 | 3.508 | 4.778 | 4.778 | 93831500 |
| **2010-06-30** | 5.158 | 6.084 | 4.660 | 4.766 | 4.766 | 85935500 |
| **2010-07-01** | 5.000 | 5.184 | 4.054 | 4.392 | 4.392 | 41094000 |
| **2010-07-02** | 4.600 | 4.620 | 3.742 | 3.840 | 3.840 | 25699000 |
| **2010-07-06** | 4.000 | 4.000 | 3.166 | 3.222 | 3.222 | 34334500 |

## Plotar esses dados e verificar as informações das ações

```
[4]  # Criando um Gráfico Dinâmico
     # No gráfico é possivel filtrar pela legenda a informação
     # Utlizar zoons

     # Definindo uma figura
     Figura = Dash.Figure()

     # Incluindo o Eixo no Gráfico - Abertura
     Figura.add_trace(Dash.Scatter(x = Base_Dados.index, y = Base_Dados.Open,
                         mode='lines',
                         name='Abertura',
                         marker_color = '#FF7F0E',
                         visible = "legendonly"))

     # Incluindo o Eixo no Gráfico - Maior
     Figura.add_trace(Dash.Scatter(x = Base_Dados.index, y = Base_Dados.High,
                         mode='lines',
                         name='Maior',
                         marker_color = '#2CA02C',
                         visible = "legendonly"))

     # Incluindo o Eixo no Gráfico - Menor
     Figura.add_trace(Dash.Scatter(x = Base_Dados.index, y = Base_Dados.Low,
                         mode='lines',
                         name='Menor',
                         marker_color = '#D62728',
                         visible = "legendonly"))
```

[4]
```
# Incluindo o Eixo no Gráfico - Fechamento
Figura.add_trace(Dash.Scatter(x = Base_Dados.index, y = Base_Dados.Close,
                    mode='lines',
                    name='Fechamento',
                    marker_color = '#1F77B4'))

# Modificando o Layout do Gráfico
Figura.update_layout(
    title='Histórico de Preço', # Titulo
    titlefont_size = 28, # Tamanho da Fonte

    # Parametros para mexer no eixo X
    xaxis = dict(
        title='Período Histórico', # Titulo do Eixo x
        titlefont_size=16, # Tamanho fonte do Titulo
        tickfont_size=14), # Tamanho da fonte do eixo

    # Tamanho do Grafico
    height = 500,

    # Parametros para mexer no eixo y
    yaxis=dict(
        title='Preço em Rupia Indiana (₹)', # Titulo do Eixo y
        titlefont_size=16, # Tamanho fonte do Titulo
        tickfont_size=14), # Tamanho da fonte do eixo

    # Parametros para mexer na legenda
    legend=dict(
        y=1, x=1, # Posição da Legenda
        bgcolor='rgba(255, 255, 255, 0)', # Cor de fundo
        bordercolor='rgba(255, 255, 255, 0)')) # Cor da Bornda

# Mostrando o Gráfico
Figura.show()
```

## Vamos ajustar a base de dados para o modelo

```
[5]  # Filtrando os dados de Fechamento
     Dados_Fechamento = Base_Dados.filter(['Close'])
     # Filtrando apenas o valores
     Dados_Fechamento_Valores = Dados_Fechamento.values
     # Arredondando o numero para cima usando o 'math.ceil'
     Dados_Fechamento_Valores_Tamanho = math.ceil(len(Dados_Fechamento_Valores) * .8)

     Dados_Fechamento_Valores_Tamanho
```

```
2242
```

```
[6]  # Aplicando escalonamento nos dados
     # Definindo os parametros da função de escalonamento
     Funcao_Escalonamento_01 = MinMaxScaler()
     # Aplicando nos dados de Fechamento
     Dados_Escalonados_Fechamento = Funcao_Escalonamento_01.fit_transform(Dados_Fechamento_Valores)

     Dados_Escalonados_Fechamento
```

```
array([[0.00183878],
       [0.00182515],
       [0.00140011],
       ...,
       [0.80081368],
       [0.81721271],
       [0.8114395 ]])
```

```
[7]  # Definindo os dados de treinamento
     Dados_Treino = Dados_Escalonados_Fechamento

     # Listas para receber os dados
     x_treinamento = []
     y_treinamento = []

     # Loop para separar os dados de treino e teste
     # Nesse Loop vamos seprar os dados em blocos de 60 valores
     for Loop in range(60, len(Dados_Treino)):

         # Separando os dados de treinamento x
         Filtrando_Amostra_Treinamento_x = Dados_Treino[Loop-60:Loop,0]
         x_treinamento.append( Filtrando_Amostra_Treinamento_x )

         # Separando os dados de treinamento y
         Filtrando_Amostra_Treinamento_y = Dados_Treino[Loop, 0]
         y_treinamento.append( Filtrando_Amostra_Treinamento_y )
```

```
# Transformando as listas em Array
x_treinamento, y_treinamento = np.array(x_treinamento), np.array(y_treinamento)
# Convertendo o array para Matriz
x_treinamento = np.reshape(x_treinamento, (x_treinamento.shape[0], x_treinamento.shape[1], 1))
# Verificando a demisão da nossa matriz
x_treinamento.shape
```

```
(2742, 60, 1)
```

## Vamos treinar nosso modelo [ Vai demorar uns segundos ]

```
[9]  # Definindo a função do Keras
     # Essa função é uma pilha linear de camadas do Keras
     Modelo = Sequential()

     # Adicinando as camadas e parametros para nossa rede neural
     # Treinamento da Rede Neural Recorrente

     # LSTM - Long Short-Term Memory
     Modelo.add(LSTM(50, return_sequences = True,
                     input_shape = (x_treinamento.shape[1], 1)))
     Modelo.add(LSTM(50, return_sequences = False))

     # Adicionando as camadas na rede neural
     Modelo.add(Dense(25))
     Modelo.add(Dense(1))
     Modelo.compile(optimizer = 'adam', loss = 'mean_squared_error')
     # Treinando o modelo
     Modelo.fit(x_treinamento, y_treinamento, batch_size = 1, epochs = 1)
```

```
2742/2742 [==============================] - 79s 28ms/step - loss: 0.0017
<keras.callbacks.History at 0x7f5179888b10>
```

```
[24] # Definindo amostra para ser testada
     Dados_Teste = Dados_Escalonados_Fechamento[Dados_Fechamento_Valores_Tamanho - 60: , :]
     # Lista para recever os dados de teste
     x_teste = []
     # Lista com os dados de teste
     y_Teste = Dados_Fechamento_Valores[Dados_Fechamento_Valores_Tamanho:, :]

     # Loop para fixar amostra para teste
     for Loop in range (60, len(Dados_Teste)):
         x_teste.append(Dados_Teste[Loop - 60:Loop, 0])

     # Transformando os dados em um array
     x_teste = np.array(x_teste)
     # Convertendo o array para Matriz
     x_teste = np.reshape(x_teste, (x_teste.shape[0], x_teste.shape[1], 1))

     # Aplicando as Previsões
     Previsoes = Modelo.predict(x_teste)
     # Invertendo para escalas reais
     Previsoes = Funcao_Escalonamento_01.inverse_transform(Previsoes)

     # Calculando o erro quadrático médio
     rsme = np.sqrt(np.mean(Previsoes - y_Teste) ** 2)
     print('Erro Quadrático Médio:', rsme )
```

```
Erro Quadrático Médio: 20.11408321346174
```

https://medium.com/turing-talks/como-avaliar-seu-modelo-de-regress%C3%A3o-c2c8d73dab96

## Ajustes Finais

```
[11] # Atribuindo as previsões no DataSet
     Validação['Previsões'] = Previsoes
```

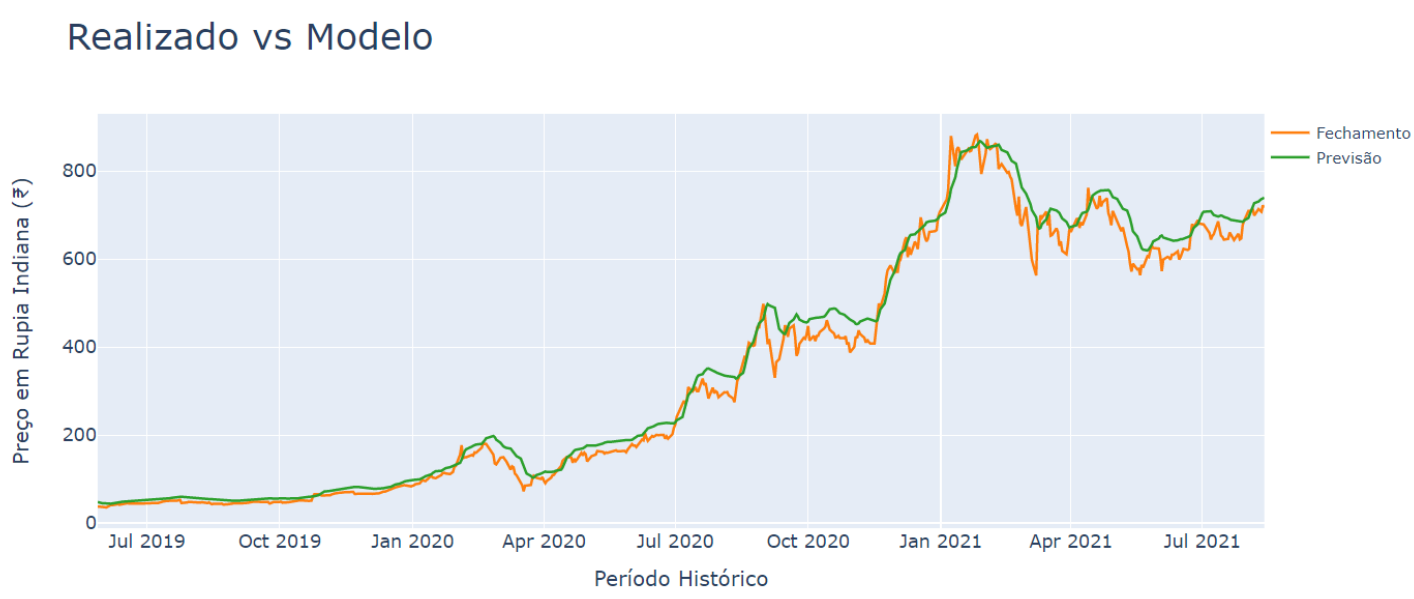
```
[23] # Verificando o real vs modelo
     Validação[['Close','Previsões']].head()
```

| Date | Close | Previsões |
|---|---|---|
| **2019-05-28** | 37.740002 | 48.013565 |
| **2019-05-29** | 37.972000 | 46.954281 |
| **2019-05-30** | 37.644001 | 46.118889 |
| **2019-05-31** | 37.032001 | 45.456936 |
| **2019-06-03** | 35.793999 | 44.882004 |

## Plotagem final

```
[27] # Criando um Gráfico Dinâmico
     # No gráfico é possivel filtrar pela legenda a informação
     # Utlizar zoons

     # Definindo uma figura
     Figura = Dash.Figure()

     # Incluindo o Eixo no Gráfico - Fechamento
     Figura.add_trace(Dash.Scatter(x = Validação.index, y = Validação.Close,
                         mode='lines',
                         name='Fechamento',
                         marker_color = '#FF7F0E',
                         ))

     # Incluindo o Eixo no Gráfico - Previsão
     Figura.add_trace(Dash.Scatter(x = Validação.index, y = Validação.Previsões,
                         mode='lines',
                         name='Previsão',
                         marker_color = '#2CA02C',
                         ))

     # Modificando o Layout do Gráfico
     Figura.update_layout(
         title='Realizado vs Modelo', # Titulo
         titlefont_size = 28, # Tamanho da Fonte

         # Parametros para mexer no eixo X
         xaxis = dict(
             title='Período Histórico', # Titulo do Eixo x
             titlefont_size=16, # Tamanho fonte do Titulo
             tickfont_size=14), # Tamanho da fonte do eixo
```

```
        # Tamanho do Grafico
        height = 500,

        # Parametros para mexer no eixo y
        yaxis=dict(
            title='Preço em Rupia Indiana (₹)', # Titulo do Eixo y
            titlefont_size=16, # Tamanho fonte do Titulo
            tickfont_size=14), # Tamanho da fonte do eixo

        # Parametros para mexer na legenda
        legend=dict(
            y=1, x=1, # Posição da Legenda
            bgcolor='rgba(255, 255, 255, 0)', # Cor de fundo
            bordercolor='rgba(255, 255, 255, 0)')) # Cor da Bornda

    # Mostrando o Gráfico
    Figura.show()
```




***Acurácia Aceitável***
***hahahah !!!***

## Final

Esse guia foi elaborada para demostrar como prever preço de fechamento de ações.

Link do Colab
https://colab.research.google.com/drive/1TtYHd30hPRaHPludDStDhdR2DmzBYS37?usp=sharing

**Odemir Depieri Jr**

Data Intelligence Analyst Sr
Tech Lead
Specialization AI