Cristóvão Augusto Pessanha de Souza Júnior

Projeto – Redes Neurais

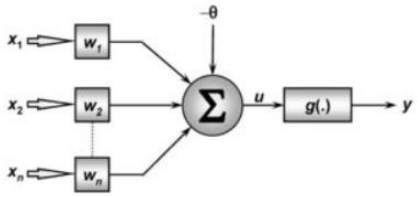
BI MASTER – PUC RIO

Sumário

1 – INTRODUÇÃO

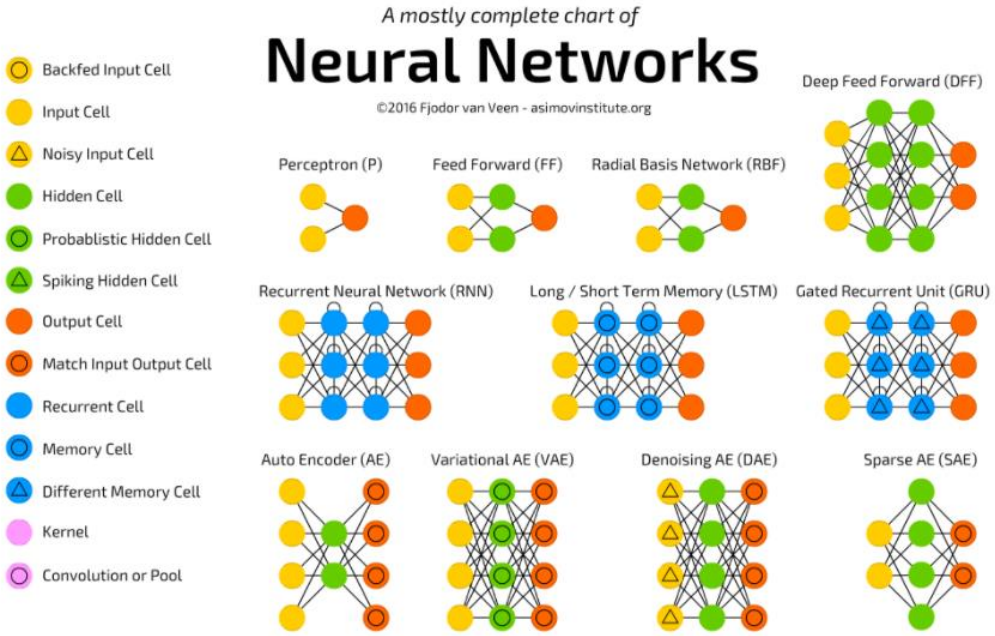
1.1 - O que são Redes Neurais e para que servem?

De forma breve e análoga é possível definir que redes neurais, são estruturas de algoritmos computacionais sofisticados, que utilizam linguagem de programação na construção de lógicas operacionais de interação e integração de dados, onde um modelo matemático irá desempenhar a função de um neurônio humano, como mostrado na imagem abaixo.



Esse conjunto de funções matemáticas formam o que chamamos de ARQUITETURAS de Redes Neurais, as quais simulam o comportamento de neurônios e desempenham algumas funções relacionadas a: ***VISÃO, AUDIÇÃO, TATO, OLFATO E FALA***, de uma forma mais rápida, precisa e objetiva, facilitando a vida tanto em um âmbito social ou como o de produção de bens de consumo, serviços diversos, medicina, educação, dentre outros em diferentes escalas e proporções.

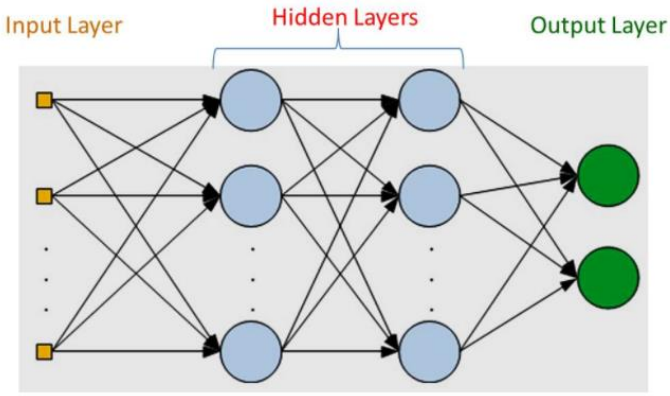
1.2 - Quais os tipos existentes de Arquiteturas de Redes Neurais conhecidas?



Existem diversas arquiteturas, onde cada uma delas é dedicada a tratar dados estruturados ou não estruturados da melhor forma, seja para classificação, regressão linear, previsão de séries temporais, reconhecimento de imagens, análise de vídeo, dentre outras. São divididas em três grandes grupos: FEED-FOWARD, RECORRENTES & HÍBRIDAS.

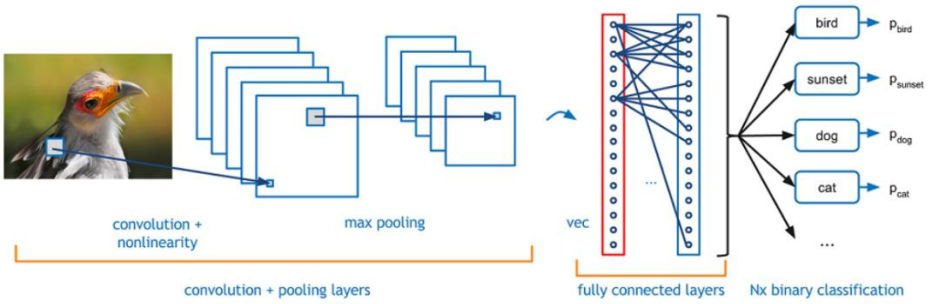
1.3 - As arquiteturas das Redes Neurais

1.3.1 - Redes Multilayer Perceptrons



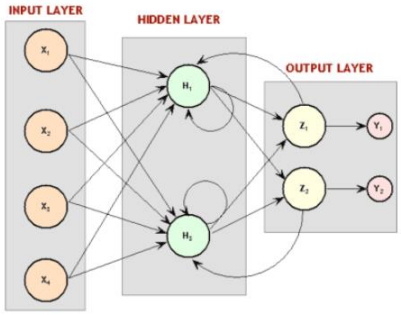
É um algoritmo simples destinado a realizar a classificação binária. Um Perceptron é um classificador linear; ou seja, é um algoritmo que classifica a entrada separando duas categorias com uma linha reta. A entrada geralmente é um vetor de recursos x multiplicado por pesos w e adicionado a um viés (ou bias) b. Aqui um exemplo do Perceptron: y = w * x + b. Um Perceptron produz uma única saída com base em várias entradas de valor real, formando uma combinação linear usando os pesos (e às vezes passando a saída através de uma função de ativação não linear).

1.3.2 - Redes Neurais Convolucionais



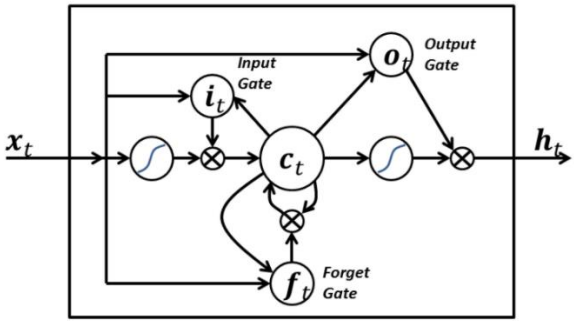
As Redes Neurais Convolucionais (ConvNets ou CNNs) são redes neurais artificiais profundas que podem ser usadas para classificar imagens, agrupá-las por similaridade (busca de fotos) e realizar reconhecimento de objetos dentro de cenas. São algoritmos que podem identificar rostos, indivíduos, sinais de rua, cenouras, ornitorrincos e muitos outros aspectos dos dados visuais.

1.3.3 - Redes Neurais Recorrentes



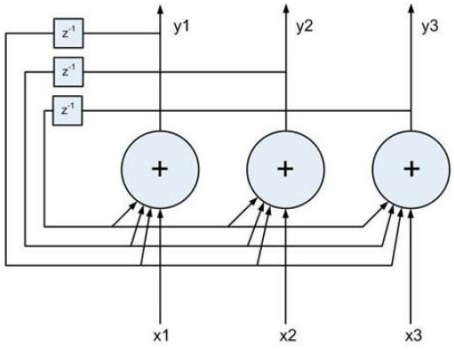
As redes recorrentes são um poderoso conjunto de algoritmos de redes neurais artificiais especialmente úteis para o processamento de dados sequenciais, como som, dados de séries temporais ou linguagem natural. Uma versão de redes recorrentes foi usada pelo DeepMind no projeto de videogames com agentes autônomos.

1.3.4 - Long Short-Term Memory (LSTM)



Os LSTM's possuem muitas aplicações práticas, incluindo processamento de linguagem natural, geração automática de texto e análise de séries temporais.

1.3.5 - Redes de Ropfield



Hopfield é simples, a rede pode ser descrita como uma rede auto associativa, tendo uma camada de neurônios totalmente conectada, onde apenas uma camada de nós existe e que todos estão conectados a todos. Significa que se a rede neural reconhecer um padrão e irá retorná-lo como output. Essas Redes neurais resolvem questões que envolvam problemas relacionados a classificação, predição e reconhecimento de padrões.

2 – DESENVOLVIMENTO

2.1 - Exploração e Análise de dados iniciais.

Para compreender a estrutura que compõe a base de dados original antes da criação do modelo de predição, a sequência de códigos demonstra quais são os principais atributos que contém a estrutura do arquivo a ser analisado, bem como definir qual será o “label”, ou seja, a variável a ser predita no estudo em questão.

```
dataframe = pandas.read_csv('ldeaths.csv', engine='python', sep = ',')
dataframe.head(5)
```

	index	value
0	1974 Jan	3035
1	1974 Feb	2552
2	1974 Mar	2704
3	1974 Apr	2554
4	1974 May	2014

```
[5] dataframe.tail(5)
```

	index	value
67	1979 Aug	1354
68	1979 Sep	1333
69	1979 Oct	1492
70	1979 Nov	1781
71	1979 Dec	1915

```
[4] dataframe.shape
(72, 2)
```

É possível entender a partir da base de dados, que a existe uma estrutura de série temporal relacionando a quantidade de pessoas com doenças pulmonares entre janeiro de 1974 a dezembro de 1979, onde essa quantidade está relacionada por mês de ocorrências. A base de dados contém 72 linhas e 2 colunas.

2.2 – Detalhamento da estrutura da Rede LSTM.

A estrutura original do projeto segue uma sequência de códigos que irão desempenhar as seguintes operações:

1) Carregar apenas a coluna com o total de doentes por mês.

```
[ ] # Carrega apenas a coluna com o total de doentes por mês
    dataframe = pandas.read_csv('ldeaths.csv', usecols=[1], engine='python', skipfooter=3, sep = ',')
    dataframe.head(3)
```

	value
0	3035
1	2552
2	2704

2) Ajustar a escala dos dados entre 0 e 1 e dividir a base de dados entre treino (67% primeiros meses) e teste (33% dos meses finais da série).

```
[ ] # Converte a coluna do dataframe pandas em um vetor numpy
    dataset = dataframe.values
    dataset = dataset.astype('float32')

    look_back = 12

    # Divite os dados de treino (2/3) e teste (1/3)
    # Note que a divisão não é aleatória, mas sim sequencial
    train_size = int(len(dataset) * 0.67)
    test_size = len(dataset) - train_size
    train, test = dataset[0:train_size,:], dataset[train_size-look_back-1:len(dataset),:]
```

3) Criação dos pares de dados X e Y, onde Yt = Xt−1 (para lookbak=1). Uma outra forma de pensar no valor de Y para um dado X é que ele é o próximo X na série temporal.

```
[ ] # Recebe uma série e converte em uma matriz com séries deslocadas.

    def create_dataset(dataset, look_back=1, std=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back):
            a = dataset[i:(i+look_back), 0]-dataset[i, 0]
            a /= std
            dataX.append(a)
            dataY.append(dataset[i + look_back, 0]-dataset[i + look_back-1, 0])
        return numpy.array(dataX), numpy.array(dataY)

    # reshape into X=t and Y=t+1
    std = train[:, 0].std()
    trainX, trainY = create_dataset(train, look_back, std)
    testX, testY = create_dataset(test, look_back, std)
    # shape is [samples, time steps, features]

    trainX = trainX.reshape(-1, look_back, 1)
    testX = testX.reshape(-1, look_back, 1)
    trainY = trainY / 30
    testY = testY / 30

    trainX.shape, testX.shape

    ((34, 12, 1), (24, 12, 1))
```

4) Criar uma rede LSTM com 32 recorrências e treinamento.

```
model = Sequential()
model.add(LSTM(32, input_shape=(look_back, 1), return_sequences=False)) # 16, 48
model.add(Dropout(0.2))
#model.add(LSTM(32, return_sequences=False))
model.add(Dense(1)) #2
model.compile(loss='mean_squared_error', optimizer='adam') #sgd
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 32)                4352
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0
_________________________________________________________________
```

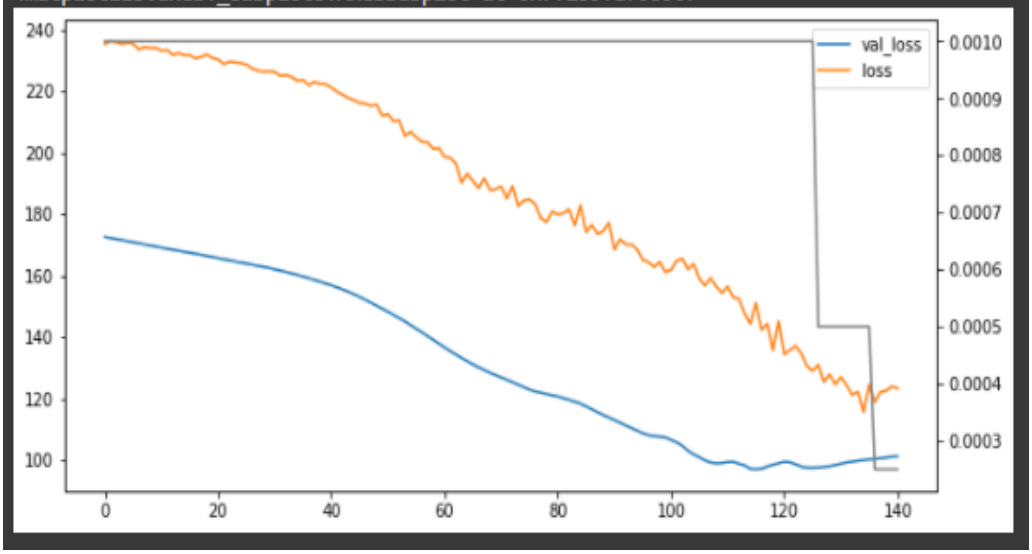
```
import absl.logging
absl.logging.set_verbosity(absl.logging.ERROR)

callbacks = [
    ReduceLROnPlateau(patience=10, factor=0.5, verbose=True),
    ModelCheckpoint('best.model', save_best_only=True),
    EarlyStopping(patience=25, verbose=True)
]

history = model.fit(trainX, trainY, epochs=5000, batch_size=24, validation_data=(testX, testY), #epochs
                    verbose=0, callbacks=callbacks)
```

```
df_history = pandas.DataFrame(history.history)
ax = df_history[['val_loss', 'loss']].plot(figsize=(10, 5))
df_history['lr'].plot(ax=ax.twinx(), color='gray')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f259fa79590>
```

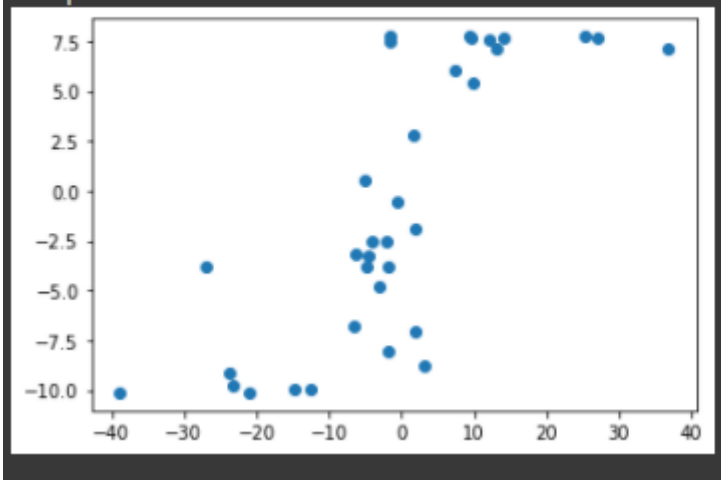


5) Realizar as previsões.

```
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
```

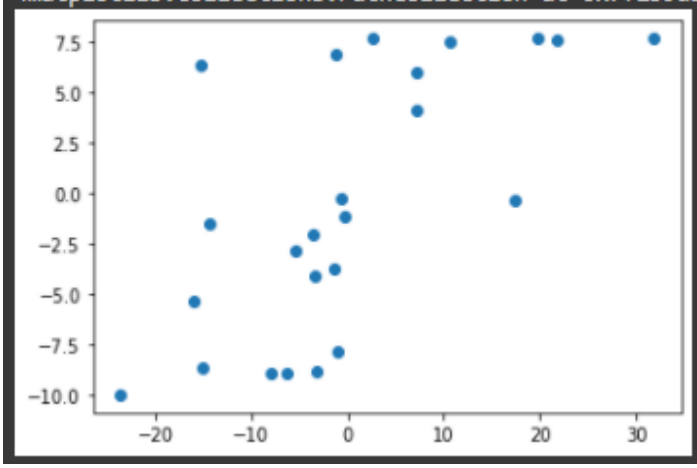
```
plt.scatter(trainY, trainPredict.ravel())
```

```
<matplotlib.collections.PathCollection at 0x7f25a398ca90>
```




```
plt.scatter(testY, testPredict.ravel())
```

```
<matplotlib.collections.PathCollection at 0x7f259d2200d0>
```



6) Reescalonar os dados para a escala original e calcular as métricas de RMSE.

```
# Calcula os erros de previsão
trainScore = math.sqrt(mean_squared_error(trainY, trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY, testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
```

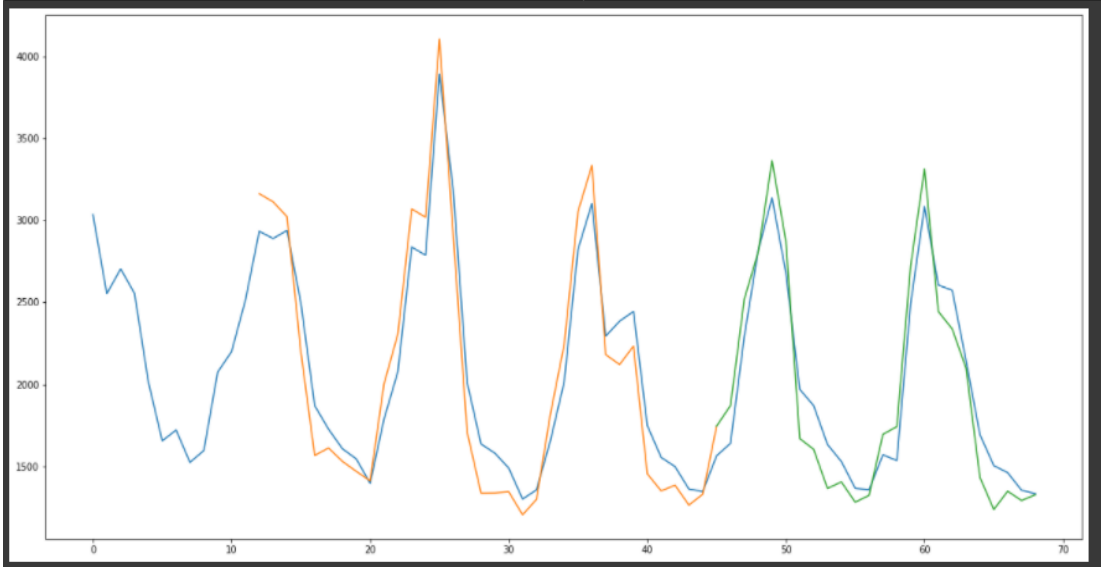
```
Train Score: 10.96 RMSE
Test Score: 10.07 RMSE
```

7) Imprimir o gráfico da previsão.

```
# shift train predictions for plotting
trainPredictPlot = (trainPredict.ravel() * 30) + dataset[look_back:len(trainPredict)+look_back, 0]

# shift test predictions for plotting
testPredictPlot = (testPredict.ravel() * 30) + dataset[len(trainPredict)+(look_back)-1:len(dataset), 0]

# plot baseline and predictions
plt.figure(figsize=(20, 10))
plt.plot(dataset)
plt.plot(look_back+numpy.arange(len(trainPredictPlot)), trainPredictPlot)
plt.plot(look_back+numpy.arange(len(testPredictPlot))+len(trainPredictPlot)-1, testPredictPlot)
plt.show()
```



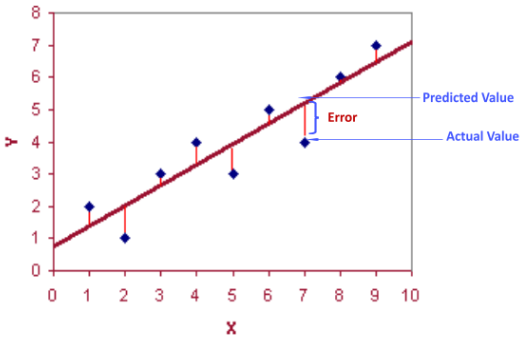
A principal fonte que será utilizada para avaliar o desempenho será o RMSE, e a partir dele será possível avaliar como ocorreu a mudança nos valores de RMSE do modelo original irá ser modificado a partir da alteração de alguns parâmetros no modelo original. Porém para caráter de entendimento do projeto como um todo, abaixo segue uma breve explicação do que vem a ser o RMSE.

2.3 - RMSE

É uma fórmula matemática utilizada para avaliar modelos de regressão linear medindo o desvio padrão dos resíduos observados.

$$RMSE = \sqrt{\frac{\sum (y_i - y_p)^2}{n}}$$

y_i = actual value
y_p = predicted value
n = number of observations/rows

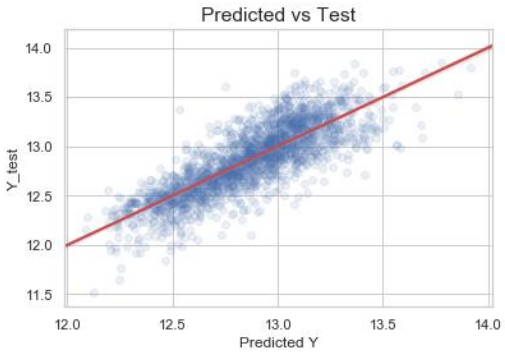


Desvio Padrão: O desvio padrão é uma medida de quão dispersos estão os números. Sua fórmula é a raiz quadrada da Variância. A variância é definida como a média das diferenças quadradas da média.

Resíduos: Resíduos são uma medida de quão longe os pontos de dados da linha de regressão estão. Os resíduos nada mais são do que um erro de previsão, podemos encontrá-lo subtraindo o valor previsto do valor real.

O RMSE é uma medida de quão espalhados são esses resíduos. Em outras palavras, ele informa o quão concentrados os dados estão em torno da linha de melhor ajuste.

Como os erros são elevados ao quadrado antes de serem calculados, o RMSE atribui um peso relativamente alto aos erros grandes. Isso significa que o RMSE é mais útil quando erros grandes são particularmente indesejáveis.



2.4 – Desenvolvimento das alterações no código original.

A ideia do projeto é, através da criação e análise de uma série temporal utilizando redes neurais LSTM, prever futuros casos, modificando um notebook original nos seguintes parâmetros: ***OTIMIZADOR ADAM POR SGD, QUANTIDADE DE CAMADAS DA LSTM (#16, #48) ACRESCENTAR UM NOVA CAMADA DENSE E MODIFICAR A QUANTIDADE DE ÉPOCAS DUAS VEZES.***

A partir da verificação do comportamento dos resultados dos cinco novos valores de RMSE, poderá ser decidido, qual é a melhor estrutura de LSTM para a previsão do "label": quantidade de pessoas com doenças pulmonares.

2.4.1 - Modificação do otimizador ADAM para SGD.

```
"""É nesse trecho de código que a mudança do otimizador ADAM para SGD foi realizada."""

model = Sequential()
model.add(LSTM(32, input_shape=(look_back, 1), return_sequences=False)) # 16, 48
model.add(Dropout(0.2))
#model.add(LSTM(32, return_sequences=False))
model.add(Dense(1)) #2
model.compile(loss='mean_squared_error', optimizer='sgd') #adam
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 32)                4352
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0
_________________________________________________________________
```

Após a alteração do otimizador esse é o novo valor de RMSE encontrado.

```
Train Score: 7.55 RMSE
Test Score: 8.65 RMSE
```

2.4.2 - Mudança no número de camadas para 16.

```
"""É nesse trecho de código que foram feitas as mudanças  no otimizador
para ADAM para SGD e 16 recorrências."""

model = Sequential()
model.add(LSTM(16, input_shape=(look_back, 1), return_sequences=False))
model.add(Dropout(0.2))
#model.add(LSTM(32, return_sequences=False))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='sgd')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 16)                1152
_________________________________________________________________
dropout (Dropout)            (None, 16)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 17
=================================================================
Total params: 1,169
Trainable params: 1,169
Non-trainable params: 0
_________________________________________________________________
```

Após a alteração na quantidade de recorrência para 16, esse são os novos valores de RMSE para treino e teste.

```
Train Score: 10.16 RMSE
Test Score: 9.43 RMSE
```

2.4.3 - Mudança no número de camadas para 48.

```
"""É nesse trecho de código onde houve mudança do otimizador para ADAM,
e onde a quantidade de recorrências são 48."""

model = Sequential()
model.add(LSTM(48, input_shape=(look_back, 1), return_sequences=False))
model.add(Dropout(0.2))
#model.add(LSTM(32, return_sequences=False))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='sgd')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 48)                9600
_________________________________________________________________
dropout (Dropout)            (None, 48)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 49
=================================================================
Total params: 9,649
Trainable params: 9,649
Non-trainable params: 0
_________________________________________________________________
```

Após a alteração da quantidade de recorrências para 48, este são os novos valores de RMSE.

```
Train Score: 8.09 RMSE
Test Score: 9.22 RMSE
```

2.4.4 - A adição de uma segunda camada (Dense).

```
"""É nesse trecho de código temos o otimizador SGD, 48 recorrências e a adição
de outra camada DENSE."""

model = Sequential()
model.add(LSTM(48, input_shape=(look_back, 1), return_sequences=False))
model.add(Dropout(0.2))
#model.add(LSTM(32, return_sequences=False))
model.add(Dense(1))
model.add(Dense(1)) #2
model.compile(loss='mean_squared_error', optimizer='sgd')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 48)                9600
_________________________________________________________________
dropout (Dropout)            (None, 48)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 49
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 2
=================================================================
Total params: 9,651
Trainable params: 9,651
Non-trainable params: 0
_________________________________________________________________
```

Após adição de uma segunda camada DENSE, mantido as 48 recorrências e o otimizador SGD, estes são os novos valores de RMSE.

```
Train Score: 7.68 RMSE
Test Score: 8.97 RMSE
```

2.4.5 - A mudança do valor no número de épocas (#600).

Nessa fase do projeto, segue se a etapa na modificação no número de épocas, onde 2 novos valores ótimos devem ser encontrados a fim de comparação com as modificações já feias anteriormente, no otimizador, quantidade de recorrências (16 & 48), adição de outra camada dense, e por fim o número de épocas.

Foi entendido que deveria ser feito uma análise com 5 diferentes valores de épocas e então entre essas 5 decidir quais são as 2 ótimas a constituírem a conclusão do projeto, logo, segue abaixo a relação desses 5 valores de RMSE de treino e teste com a seguinte quantidade de épocas: 500, 600, 700, 800 e 900.

500

```
Train Score: 9.33 RMSE
Test Score: 9.56 RMSE
```

600

```
Train Score: 9.99 RMSE
Test Score: 8.81 RMSE
```

700

```
Train Score: 6.34 RMSE
Test Score: 8.77 RMSE
```

800

```
Train Score: 6.38 RMSE
Test Score: 9.46 RMSE
```

900

```
Train Score: 8.43 RMSE
Test Score: 8.89 RMSE
```

A partir dessas informações ficou decidido que as épocas escolhidas são: 700 e 800.

3 - CONCLUSÃO

MODIFICAÇÃO	OTIMIZADOR	LSTM	DENSE	ÉPOCA	RMSE_TREINO	RMSE_TESTE
ORIGINAL	ADAM	32	1	5000	387,68	327,24
OTIMIZADOR	SGD	32	1	5000	7,55	8,65
LSTM	SGD	16	1	5000	10,16	9,43
LSTM	SGD	48	1	5000	8,09	9,22
DENSE	SGD	48	2	5000	7,68	8,97
ÉPOCA	SGD	48	2	700	6,34	8,77
ÉPOCA	SGD	48	2	800	6,38	9,46

De acordo com a tabela acima, contendo os valores de RMSE obtidos após desenvolvimento dos modelos e suas respectivas modificações, é possível determinar que, o notebook contendo o ***Otimizador SGD, 48 Recorrências, duas camadas DENSE e 700 épocas*** é o que possui menores valores de RMSE, sendo assim considerado o melhor em termos de previsões dos dados para análise de Série Temporal.