LSTM

目标

读完LSTM 和 GRU 的原理解释，理解为何LSTM可以找到Long-term dependency, 并用代码或者已有的工具验证理解。

Article: Illustrated Guide to LSTM's and GRU's: A step by step explanation
https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

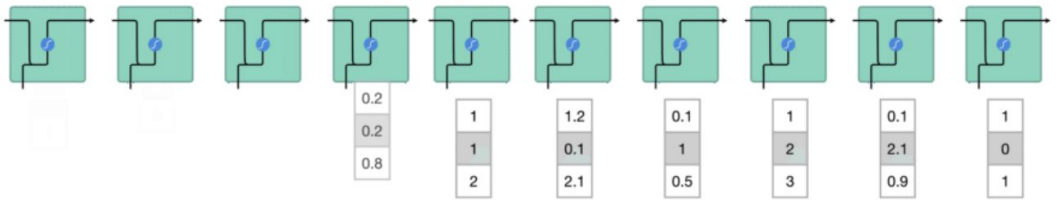
理解

首先我们还是来看看概念的问题，先了解 RNN是什么。RNN 循环神经网络（Recurrent neural network，循环神经网络），RNN是一系列能够处理序列数据的神经网络的总称。

RNN包含如下三个特性：

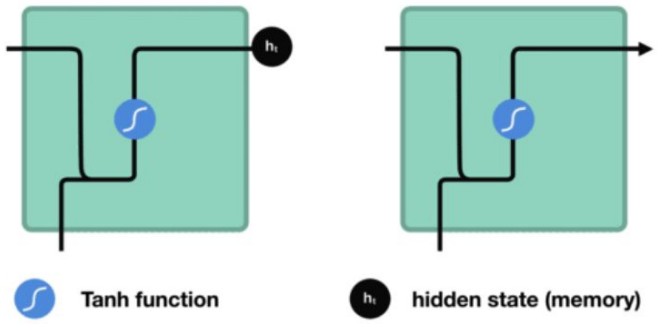
1. 循环神经网络能够在每个时间节点产生一个输出，且隐单元间的连接是循环的；
2. 循环神经网络能够在每个时间节点产生一个输出，且该时间节点上的输出仅与下一时间节点的隐单元有循环连接；
3. 循环神经网络包含带有循环连接的隐单元，且能够处理序列数据并输出单一的预测。

用作者图像化的解释如下，RNN 将信息一个连一个的处理，



Processing sequence one by one

将隐藏记忆状态和输入数据经过 Tanh 函数输出作为下一步的隐藏状态输入。



Passing hidden state to next time step

看到这里我们有个问题，RNN 和 LSTM 的关系是什么呢? 为什么要有 LSTM
一般一个新网络或者技术算法的出现是为了解决之前网络所存在的不足，LSTM 肯定也不例外，RNN 的问题是什么。

其实作者在文章开头已经说了

The Problem, Short-term Memory

Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.

RNN 一直饱受短期记忆之苦，比如一个句子很长，那么 RNN 网络很难把最开始比较重要的信息保留到最后，导致理解的误差，所以 LSTM 的出现是为了解决这一类问题的。
为了解决上述的问题最成功应用最广泛的就是门限RNN（Gated RNN），而LSTM就是门限RNN中最著名的一种。

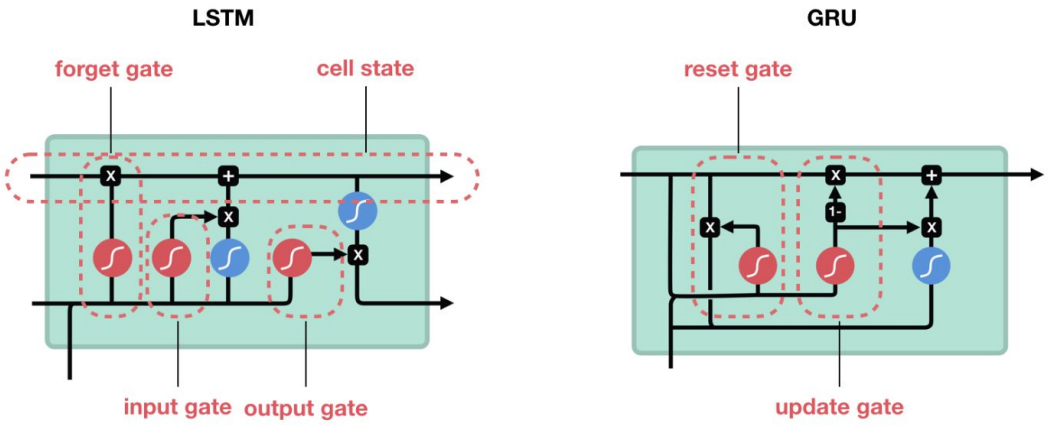
这样一来我们对 LSTM 的理解更加深入了，知道它为什么产生，解决什么样的问题(短期记忆的问题)。

OK，那接下来中重要的就是 LSTM 是怎么解决 Short-term Memroy Problem 的。

文章作者提到了解决 Shrot-term Memory 的两个方案 LSTM 和 GRU

LSTM's and GRU's as a solution

LSTM 's and GRU's were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information.

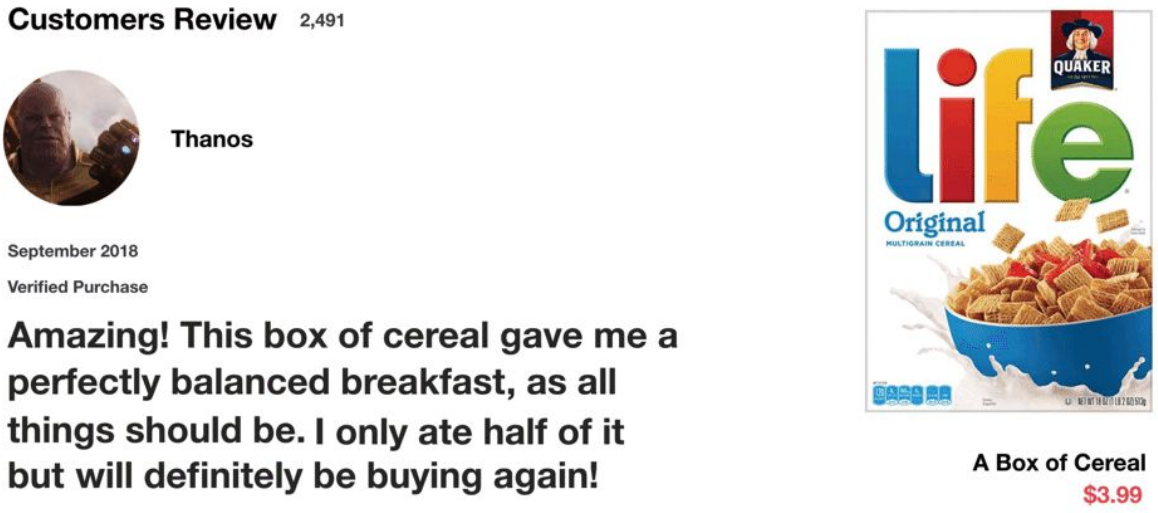


从上图可以看出 GRU 与 LSTM 的区别在于使用同一个门限来代替输入门限和遗忘门限，即通过一个update gate来控制 cell 的状态，该做法的好处是计算得以简化，同时模型的表达能力也很强，所以GRU也因此越来越流行。但是其实我们深入的剖析理解 LSTM 的工作原理就够了，GRU 类似。

另外这几个门限用的函数分别是 sigmoid 函数和 tanh 函数，数据之间有乘法、加法以及向量合并。

作者用现实生活中的情况举了个例子。比如我们想买一些小零食，我们看大家对零食的评价，其实那么一长段话如果快速看的话，我们很难一个一个一个字看，最多是扫描一下一些关键词，比如Amazing，perfectly，buy again 之类的。包括比如大多数人就算一个字一个字看，那么过几天后你大概率能够记得这个描述的就是还不错，挺好吃的，我可以推荐给其他朋友。

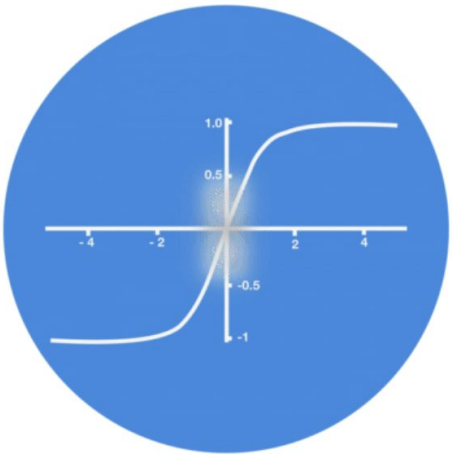
所以其实我们大脑阅读的过程也类似 LSTM，记住关键信息，最后统一合并输出。 如果是普通 RNN 的话，可能最开始的 Amazing 到最后一步的时候就模糊不清了，这个是有问题的理解。



我们继续分析 LSTM 内部实现逻辑，首先来看下为什么用 Tanh 激活函数和Sigmoid 函数

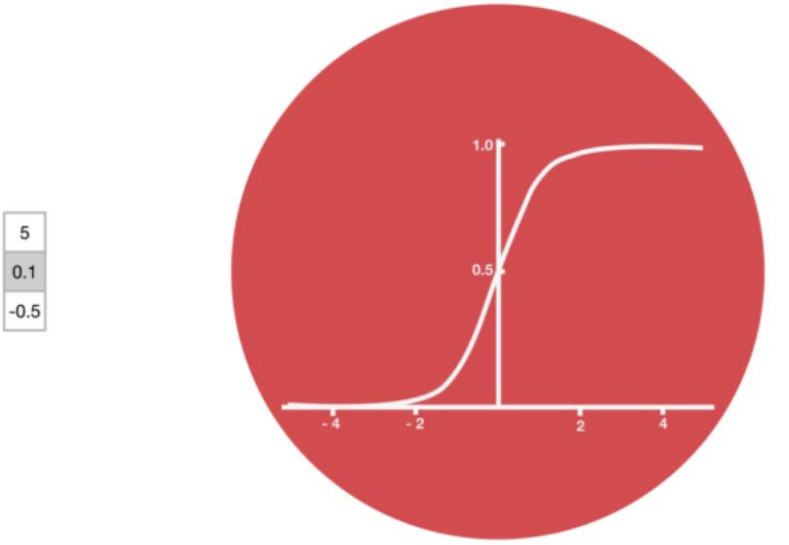
Tanh activation

The tanh activation is used to help regulate the values flowing through the network. The tanh function squishes values to always be between -1 and 1.



Tanh squishes values to be between -1 and 1

Tanh 激活函数的输出在[-1,1]之间，因此相当于把输入的均值调整为0，便于后续处理。不如用3x 函数的话，会发现处理之后的数据由最开始的差异一点，最后被放大到有巨大的落差，这不是我们想看到的，而且最后数据处理的误差也很大。但是使用 Tanh 函数会把数据输出到 -1 1 之间，而且均值为0，方便后续的数据处理。

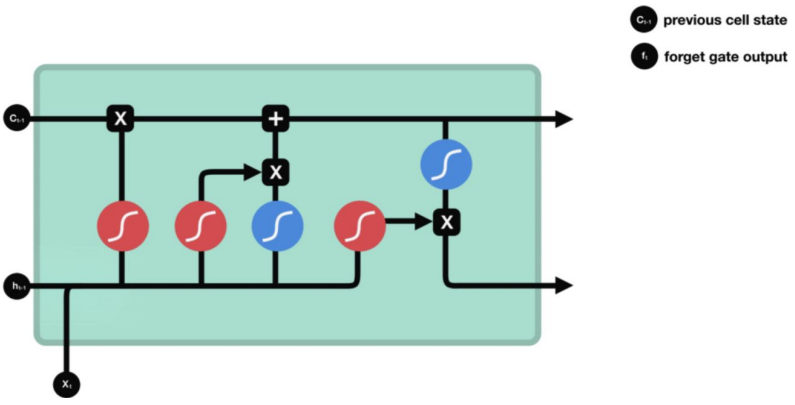


Sigmoid squishes values to be between 0 and 1

Sigmoid 激活函数输出范围是[0,1] 这是一个天然的逻辑门，0 or 1，非常适合作为记忆或者遗忘处理。

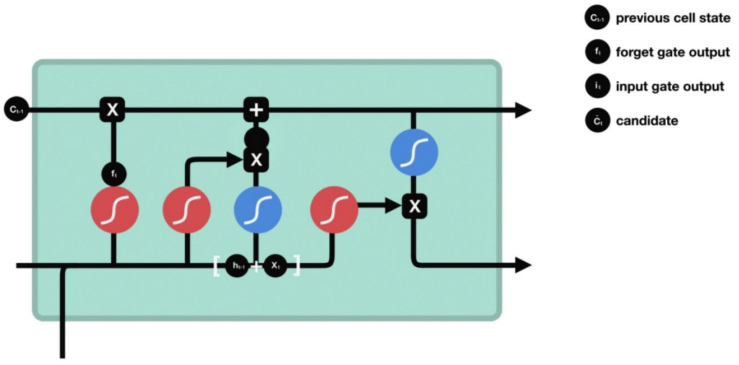
LSTM 有四个重要的概念
Forget Gate、Input Gate、Cell State、Output Gate

遗忘门：



遗忘门会把之前的隐藏状态h和当前信息 x 合并通过第一个 sigmoid 函数，这个函数就能够判断是遗忘度。接近0 意味着 forget 接近1 意味着记住。

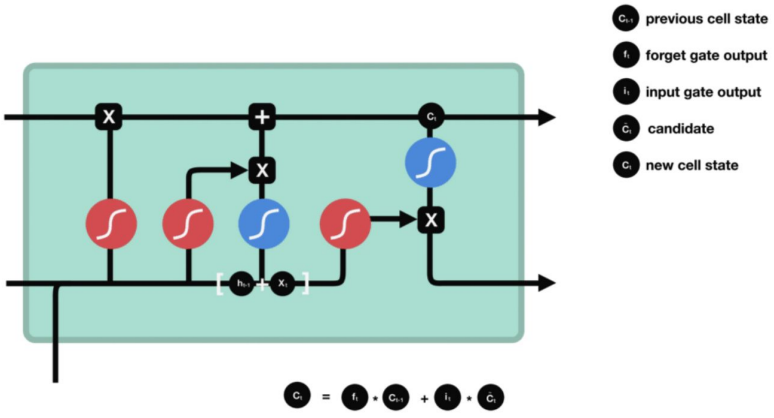
输入门：



Input gate operations

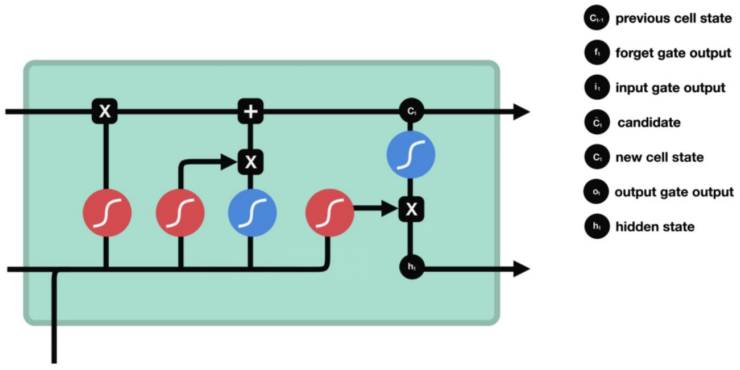
为了更新 Cell 的状态，我们将隐藏状态(和前一个节点相关的) 和 input 先通过一个 sigmoid 函数，这样是为了区分出两者的重要程度，0 不重要，1 重要。另外将 隐藏状态和 input 输入一个 Tanh 函数，主要用来调整网络，最后两者相乘。sigmoid 的意义在于约束 Tanh 函数输出的重要性。

细胞状态：



首先细胞状态和遗忘门相乘来决定重要性，之后和输入门相加来更新细胞状态，最后输出一个新的细胞状态。

输出门：



output gate operations

输出门决定了下一个细胞的隐藏状态，所以隐藏状态包含了上一个细胞的信息，可以用来做预测。首先我们将之前的隐藏状态和现在的输入经过一个 sigmoid 函数，随后将更改细胞状态后的 C1 通过 Tanh 函数之后相乘输出 h1，也就是下一个函数的隐藏状态。那么这输出包含了当前的细胞状态与输入，通过 sigmoid 函数筛选出了之前的信息重要还是当前的输入信息重要。这个也就是为什么LSTM 可以解决 short-term memory 的问题。它将之前重要的信息都通过各种门的筛选保留的下来。

总结

对于 LSTM 里面的状态和各种门，作者也给出了他自己的总结

> To review, the Forget gate decides what is relevant to keep from prior steps. The input gate decides what information is relevant to add from the current step. The output gate determines what the next hidden state should be.

Top highlight

遗忘门用来保持和之前信息的相关性，输入门决定了哪种更相关的信息应该被加入到当前网络，输出门决定了下一个隐藏状态是什么。

对于整个 LSTM 网络的理解就是关键就是细胞状态，水平线在图上方贯穿运行。通过精心设计的称作为“门”的结构来去除或者增加信息到细胞状态的能力，让信息选择式通过从而达到 Long-Term dependency。

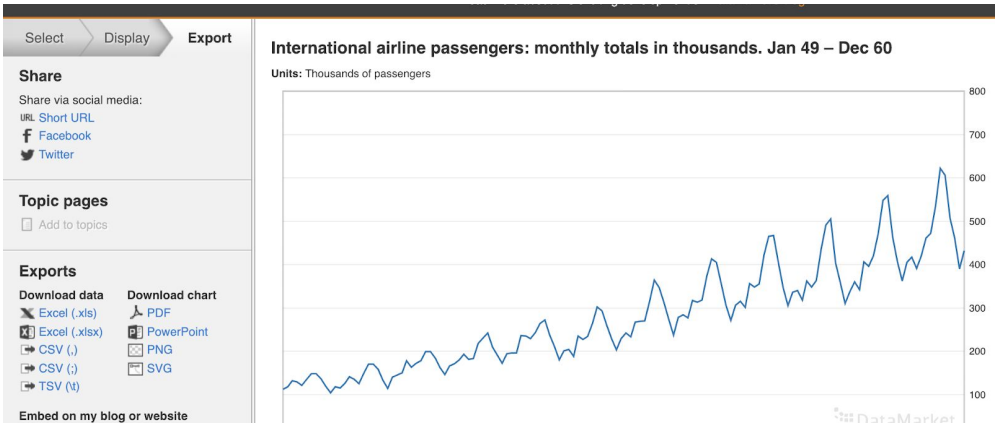
实验

为了增加进一步理解，从网上查找了一些资料做了一个关于 LSTM 对时间序列的预测

https://datamarket.com/data/set/22u3/international-airline-passengers-monthly-totals-in-thousands-jan-49-dec-60#!ds=22u3&display=line

上面地址是历年国际航班的每个月乘客人数，那么根据之前的历史来去预测后面可能几个月的人员走势。

首先我们在 Export 中把历史数据导出为 csv 格式。



接下来编写代码
导入我们使用LSTM所需要的库

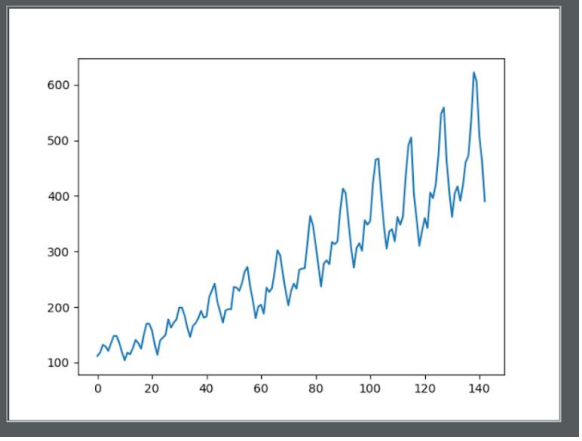
```
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

加载本地 csv， 并绘制出图像

```
dataframe = read_csv('international-airline-passengers.csv', usecols=[1], engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
plt.plot(dataset)
plt.show()
```

可以看出和网上展示的数据集走势基本一致

```
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
numpy.random.seed(7)


dataframe = read_csv('international-airline-passenge
dataset = dataframe.values
dataset = dataset.astype('float32')
plt.plot(dataset)
plt.show()

# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)

# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[trai

# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
```



接下来我们要做的验证就是使用前100个月的数据通过 LSTM 做训练，来去预测后面40个月，并且做以验证。

由于 LSTM 使用 Tanh 和 Sigmoid 函数对数据比较敏感，我们先将数据做统一化处理，把值归一化到 0-1之间

```
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

进行训练集和测试集的划分

```
train_size = int(len(dataset) * 0.7)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
```

接下来我们使用 create_dataset 函数将数据集进行训练和测试集转化，将一列的数据变为前后关联。X 是 month， Y 是 month + 1 的值

X	Y
112	118
118	132
132	129
129	121
121	135

```
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

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

创建 LSTM 模型并进行训练

```
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
```

紧接着我们开始预测

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

trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

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

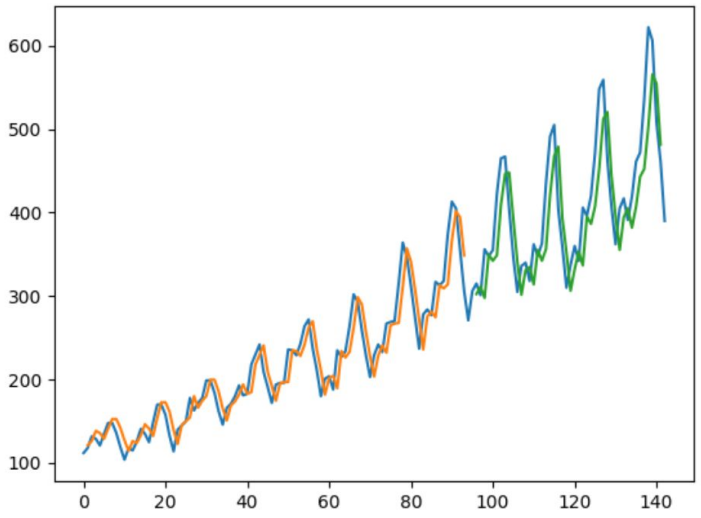
并且求出预测均方根误差

最后使用图表展示出

```
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict

plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

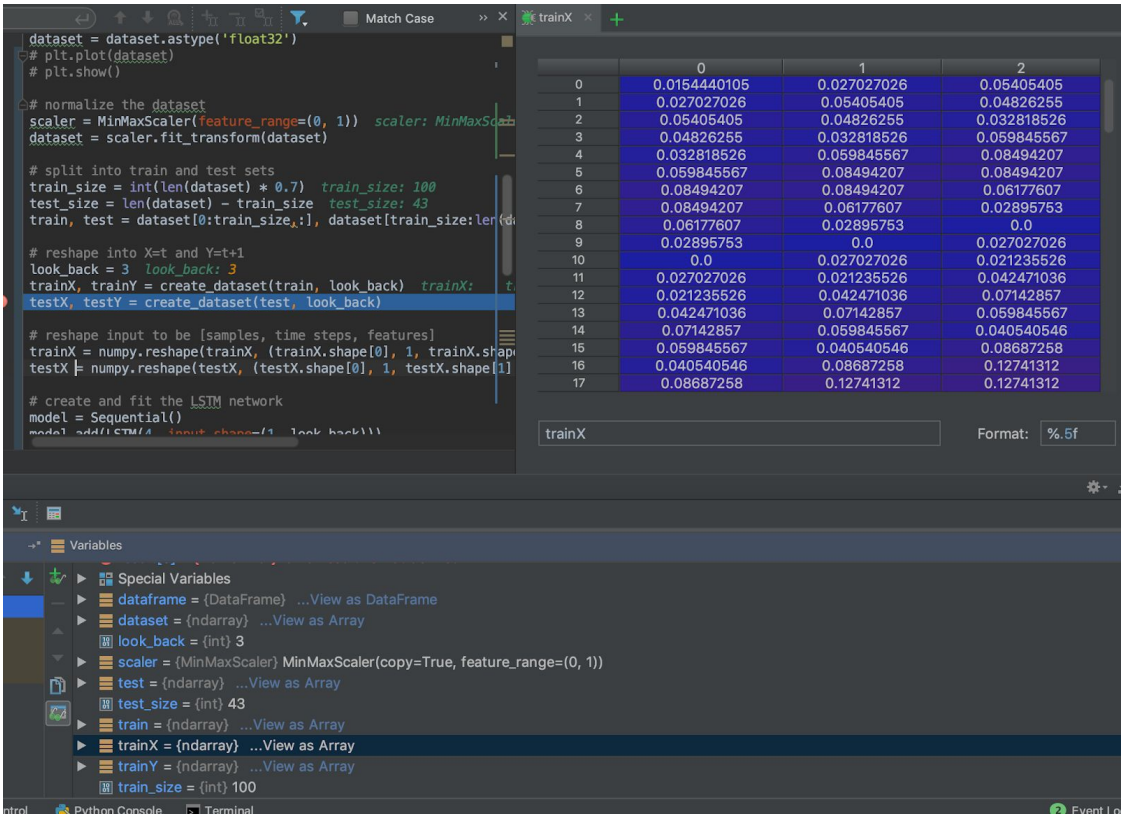
最终结果

```
Epoch 100/100
 - 0s - loss: 0.0021
Train Score: 23.52 RMSE
Test Score: 49.96 RMSE
```



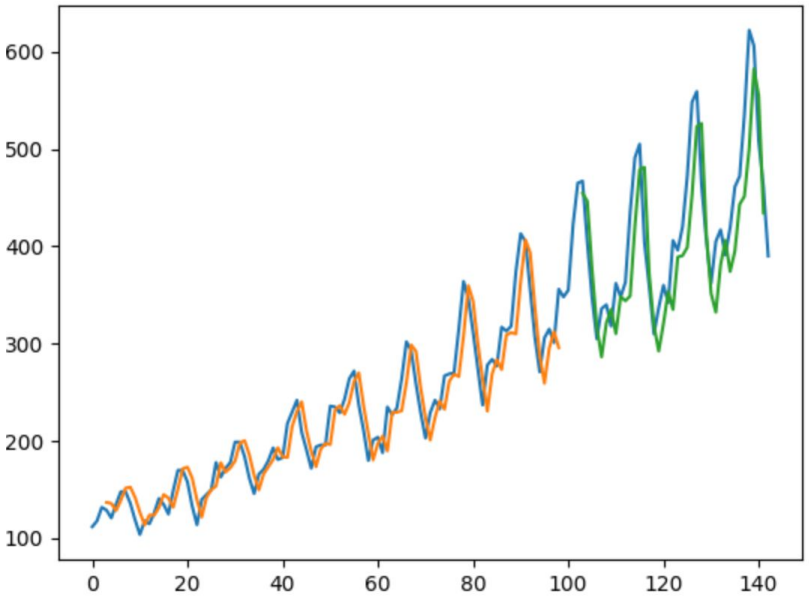
蓝色为原数据，绿色为训练集的预测值，红色为测试集的预测值，可以看出 LSTM 根据前序序列的值预测结果还不错。

之前 look back 为 1，我们可以改为3 那么LSTM 的 trianx 为



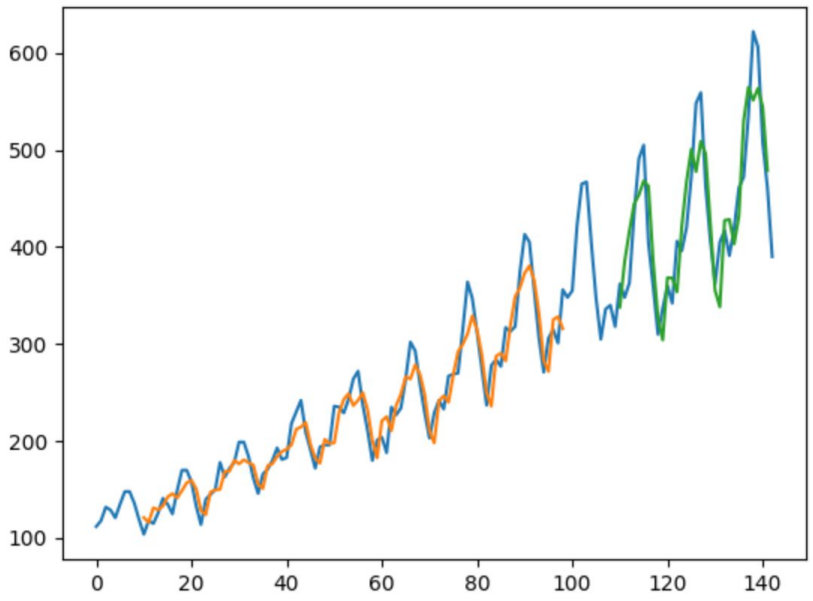
```
Epoch 100/100
 - 0s - loss: 0.0022
Train Score: 24.01 RMSE
Test Score: 49.91 RMSE

Process finished with exit code 0
```

可以看出 Train Score 有部分提升，Test Score 相差不大，图表展示如下



但 look back 改为10的话，出现了过拟合情况
无论从数据表现和图表预测度来看，都偏差较大，应为航班预测更相关的可能是前两年的数据，数据拉大太长，前期参考意义可能不大。

```
 - 0s - loss: 0.0015
Train Score: 19.49 RMSE
Test Score: 39.10 RMSE
```


600
500
400
300
200
100
0
20
40
60
80
100
120
140