LSTM代码实现

https://blog.csdn.net/comli_cn/article/details/104523867
https://blog.csdn.net/weixin_40002336/article/details/111158688

时间序列预测

头文件

```
import numpy
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import  pandas as pd
import  os
from keras.models import Sequential, load_model
from sklearn.preprocessing import MinMaxScaler
```

加载数据

```
dataframe = pd.read_csv('./international-airline-passengers.csv', usecols=[1], engine='python',
skipfooter=3)
dataset = dataframe.values
# 将整型变为float
dataset = dataset.astype('float32')
# 归一化 在下一步会讲解
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# LSTM可以不进行归一化的操作，但是这样会让训练模型的loss下降很慢。如果不进行归一化，100次
迭代后loss还是很高
train_size = int(len(dataset) * 0.65)
# 时序数据的前65%为训练数据 后35%为测试数据
trainlist = dataset[:train_size]
testlist = dataset[train_size:]
```

根据前timestep步预测后面的数据
timestep为3，即根据前三个的数据预测后一个数据的值

对数据进行处理

```
def create_dataset(dataset, look_back):
#这里的look_back与timestep相同
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(dataset[i + look_back])
    return numpy.array(dataX),numpy.array(dataY)
#训练数据太少 look_back并不能过大
look_back = 1
trainX,trainY  = create_dataset(trainlist,look_back)
testX,testY = create_dataset(testlist,look_back)
```

导入LSTM模型

timesteps为步数，features为维度 这里我们的数据是1维的

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

# create and fit the LSTM network
model = Sequential()
# important
model.add(LSTM(4, input_shape=(None,1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
model.save(os.path.join("DATA","Test" + ".h5"))
# make predictions
```

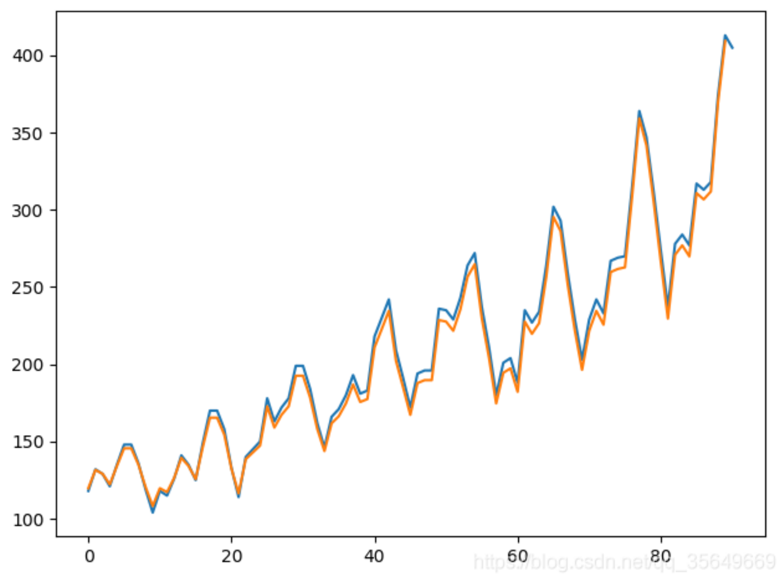
进行预测

```
#model = load_model(os.path.join("DATA","Test" + ".h5"))
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

#反归一化
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform(trainY)
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform(testY)
```

输出

```
plt.plot(trainY)
plt.plot(trainPredict[1:])
plt.show()
plt.plot(testY)
plt.plot(testPredict[1:])
plt.show()
```


400
350
300
250
200
150
100
0
20
40
60
80