time_series_prediction

April 8, 2022

1 LSTM for time series prediction

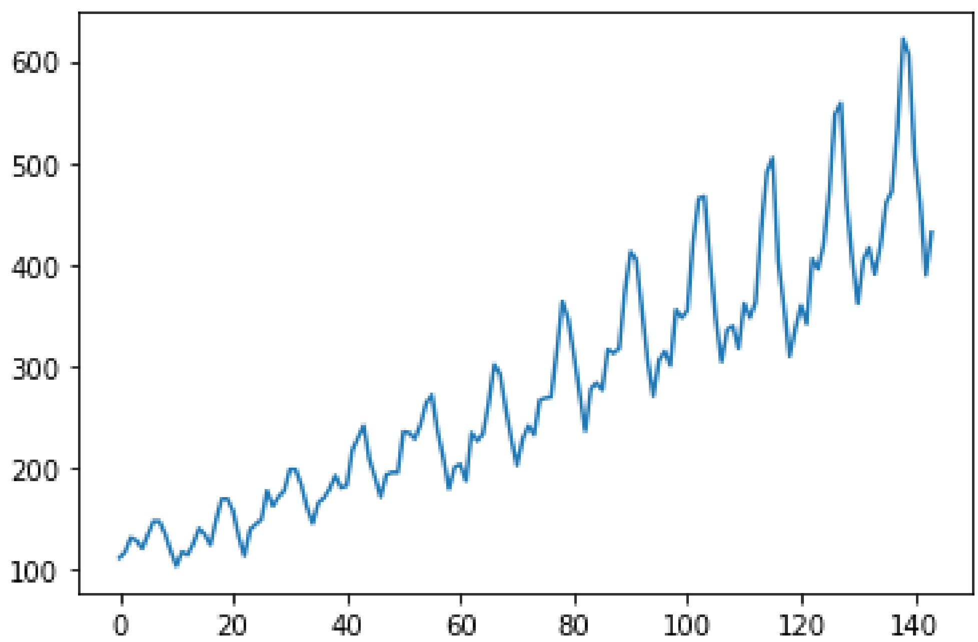
using this dataset and this guide

using an LSTM to learn time series sequences for airline passengers

1.1 intiail plot the data

[2]:
```
# interval separation of 1 month
import pandas as pd
import matplotlib.pyplot as plt

dataset = pd.read_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/
↪master/airline-passengers.csv", usecols=[1], engine='python')
plt.plot(dataset)
plt.show()
```



importing packages

[3]:
```
import numpy
import matplotlib.pyplot as plt
import pandas
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

# fix random seed for reproducibility
numpy.random.seed(7)
```

```
Using TensorFlow backend.
```

1.2 loading the data

[4]:
```
# load the dataset
dataframe = pandas.read_csv('https://raw.githubusercontent.com/jbrownlee/
 ↪Datasets/master/airline-passengers.csv', usecols=[1], engine='python')
dataset = dataframe.values
dataset = dataset.astype('float32')
```

1.3 scaling the data

sclaing the data to use a range of 0-1 (using MinMaxScaler)

[5]:
```
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

1.4 train test split

2/3 train, 1/3 test

[6]:
```
# 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[train_size:len(dataset),:]
# spreading operator and some complex indexing,
# selecting 0: train size and then selecting all columns
print(len(train), len(test))
```

```
96 48
```

1.5 creating lag variables

creating a function to create lag varaibles in the data this converts the data into a tabular form which has lag variables to predict off of

[7]:
```
# convert an array of values into a dataset matrix
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)
```

[8]:
```
# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

[9]:
```
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

1.6 creating the LSTM model

[10]:
```
# create and fit the LSTM network
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)
```

```
2022-04-08 13:54:26.507469: I tensorflow/core/platform/cpu_feature_guard.cc:145]
This TensorFlow binary is optimized with Intel(R) MKL-DNN to use the following
CPU instructions in performance critical operations:  SSE4.1 SSE4.2
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the
appropriate compiler flags.
2022-04-08 13:54:26.508840: I
tensorflow/core/common_runtime/process_util.cc:115] Creating new thread pool
with default inter op setting: 8. Tune using inter_op_parallelism_threads for
best performance.
```

```
Epoch 1/100
 - 1s - loss: 0.0409
Epoch 2/100
 - 0s - loss: 0.0197
Epoch 3/100
 - 0s - loss: 0.0142
Epoch 4/100
 - 0s - loss: 0.0127
Epoch 5/100
 - 0s - loss: 0.0117
```

```
Epoch 6/100
 - 0s - loss: 0.0106
Epoch 7/100
 - 0s - loss: 0.0097
Epoch 8/100
 - 0s - loss: 0.0087
Epoch 9/100
 - 0s - loss: 0.0076
Epoch 10/100
 - 0s - loss: 0.0065
Epoch 11/100
 - 0s - loss: 0.0057
Epoch 12/100
 - 0s - loss: 0.0048
Epoch 13/100
 - 0s - loss: 0.0041
Epoch 14/100
 - 0s - loss: 0.0035
Epoch 15/100
 - 0s - loss: 0.0030
Epoch 16/100
 - 0s - loss: 0.0027
Epoch 17/100
 - 0s - loss: 0.0025
Epoch 18/100
 - 0s - loss: 0.0023
Epoch 19/100
 - 0s - loss: 0.0022
Epoch 20/100
 - 0s - loss: 0.0021
Epoch 21/100
 - 0s - loss: 0.0021
Epoch 22/100
 - 0s - loss: 0.0021
Epoch 23/100
 - 0s - loss: 0.0021
Epoch 24/100
 - 0s - loss: 0.0020
Epoch 25/100
 - 0s - loss: 0.0020
Epoch 26/100
 - 0s - loss: 0.0021
Epoch 27/100
 - 0s - loss: 0.0020
Epoch 28/100
 - 0s - loss: 0.0020
Epoch 29/100
 - 0s - loss: 0.0020
```

```
Epoch 30/100
 - 0s - loss: 0.0021
Epoch 31/100
 - 0s - loss: 0.0020
Epoch 32/100
 - 0s - loss: 0.0020
Epoch 33/100
 - 0s - loss: 0.0021
Epoch 34/100
 - 0s - loss: 0.0021
Epoch 35/100
 - 0s - loss: 0.0021
Epoch 36/100
 - 0s - loss: 0.0020
Epoch 37/100
 - 0s - loss: 0.0021
Epoch 38/100
 - 0s - loss: 0.0020
Epoch 39/100
 - 0s - loss: 0.0021
Epoch 40/100
 - 0s - loss: 0.0020
Epoch 41/100
 - 0s - loss: 0.0020
Epoch 42/100
 - 0s - loss: 0.0020
Epoch 43/100
 - 0s - loss: 0.0021
Epoch 44/100
 - 0s - loss: 0.0020
Epoch 45/100
 - 0s - loss: 0.0021
Epoch 46/100
 - 0s - loss: 0.0020
Epoch 47/100
 - 0s - loss: 0.0020
Epoch 48/100
 - 0s - loss: 0.0020
Epoch 49/100
 - 0s - loss: 0.0020
Epoch 50/100
 - 0s - loss: 0.0020
Epoch 51/100
 - 0s - loss: 0.0020
Epoch 52/100
 - 0s - loss: 0.0020
Epoch 53/100
 - 0s - loss: 0.0020
```

```
Epoch 54/100
 - 0s - loss: 0.0020
Epoch 55/100
 - 0s - loss: 0.0021
Epoch 56/100
 - 0s - loss: 0.0020
Epoch 57/100
 - 0s - loss: 0.0020
Epoch 58/100
 - 0s - loss: 0.0020
Epoch 59/100
 - 0s - loss: 0.0020
Epoch 60/100
 - 0s - loss: 0.0020
Epoch 61/100
 - 0s - loss: 0.0021
Epoch 62/100
 - 0s - loss: 0.0020
Epoch 63/100
 - 0s - loss: 0.0020
Epoch 64/100
 - 0s - loss: 0.0020
Epoch 65/100
 - 0s - loss: 0.0020
Epoch 66/100
 - 0s - loss: 0.0020
Epoch 67/100
 - 0s - loss: 0.0020
Epoch 68/100
 - 0s - loss: 0.0021
Epoch 69/100
 - 0s - loss: 0.0020
Epoch 70/100
 - 0s - loss: 0.0021
Epoch 71/100
 - 0s - loss: 0.0020
Epoch 72/100
 - 0s - loss: 0.0020
Epoch 73/100
 - 0s - loss: 0.0020
Epoch 74/100
 - 0s - loss: 0.0021
Epoch 75/100
 - 0s - loss: 0.0021
Epoch 76/100
 - 0s - loss: 0.0020
Epoch 77/100
 - 0s - loss: 0.0021
```

```
Epoch 78/100
 - 0s - loss: 0.0019
Epoch 79/100
 - 0s - loss: 0.0022
Epoch 80/100
 - 0s - loss: 0.0020
Epoch 81/100
 - 0s - loss: 0.0020
Epoch 82/100
 - 0s - loss: 0.0020
Epoch 83/100
 - 0s - loss: 0.0020
Epoch 84/100
 - 0s - loss: 0.0020
Epoch 85/100
 - 0s - loss: 0.0021
Epoch 86/100
 - 0s - loss: 0.0021
Epoch 87/100
 - 0s - loss: 0.0020
Epoch 88/100
 - 0s - loss: 0.0020
Epoch 89/100
 - 0s - loss: 0.0020
Epoch 90/100
 - 0s - loss: 0.0020
Epoch 91/100
 - 0s - loss: 0.0020
Epoch 92/100
 - 0s - loss: 0.0020
Epoch 93/100
 - 0s - loss: 0.0021
Epoch 94/100
 - 0s - loss: 0.0021
Epoch 95/100
 - 0s - loss: 0.0020
Epoch 96/100
 - 0s - loss: 0.0020
Epoch 97/100
 - 0s - loss: 0.0020
Epoch 98/100
 - 0s - loss: 0.0020
Epoch 99/100
 - 0s - loss: 0.0020
Epoch 100/100
 - 0s - loss: 0.0020
```

[10]: <keras.callbacks.callbacks.History at 0x7f93532d4bd0>

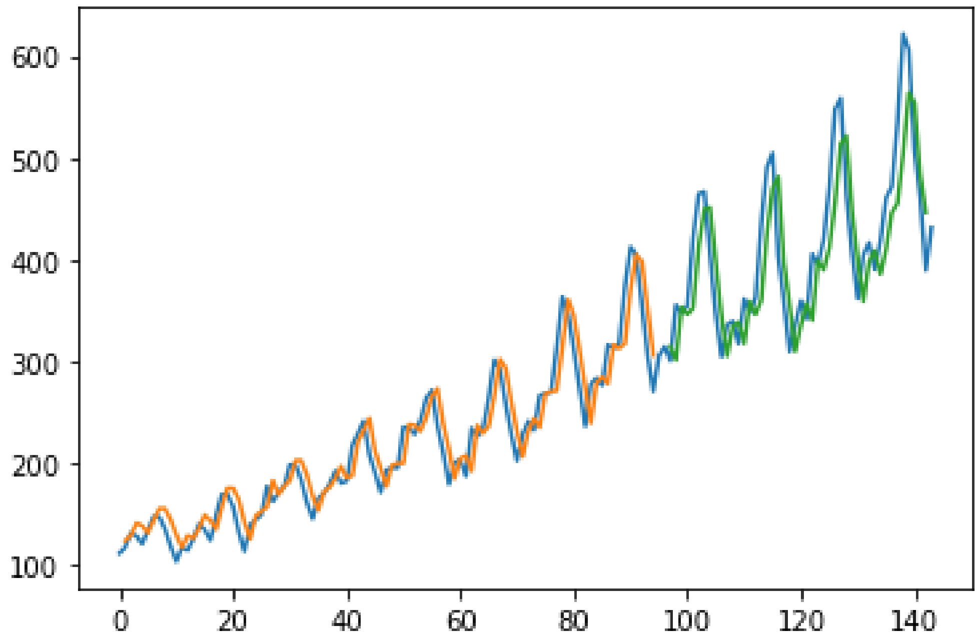
1.7 predicting

[11]:
```
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
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))
```

```
Train Score: 22.93 RMSE
Test Score: 47.60 RMSE
```

plotting predictions red is past information, green is predicted, blue is truth

[12]:
```
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] =␣
↪testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



1.8 LSTM for Regression Using the Window Method

using more lag variables rather than just one time step step back went from 1 -> 3

[13]:
```
# LSTM for international airline passengers problem with window regression␣
↪framing
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
# convert an array of values into a dataset matrix
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)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
```

```python
dataframe = read_csv('https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv', usecols=[1], engine='python')
dataset = dataframe.values
dataset = dataset.astype('float32')
# 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[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
# create and fit the LSTM network
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)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
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))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
```

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

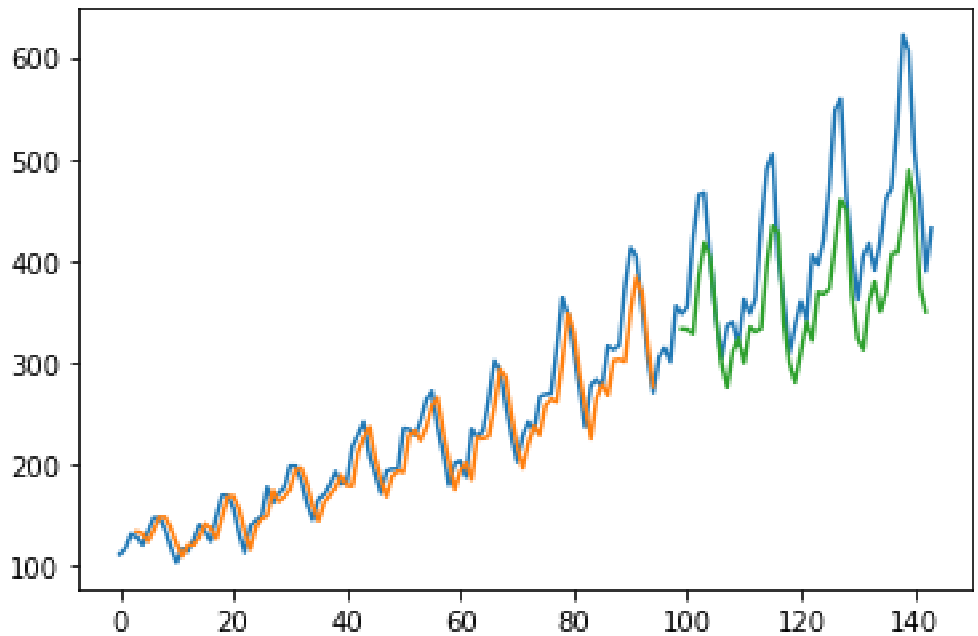
```
Epoch 1/100
 - 1s - loss: 0.0703
Epoch 2/100
 - 0s - loss: 0.0295
Epoch 3/100
 - 0s - loss: 0.0174
Epoch 4/100
 - 0s - loss: 0.0144
Epoch 5/100
 - 0s - loss: 0.0125
Epoch 6/100
 - 0s - loss: 0.0110
Epoch 7/100
 - 0s - loss: 0.0096
Epoch 8/100
 - 0s - loss: 0.0085
Epoch 9/100
 - 0s - loss: 0.0072
Epoch 10/100
 - 0s - loss: 0.0064
Epoch 11/100
 - 0s - loss: 0.0056
Epoch 12/100
 - 0s - loss: 0.0051
Epoch 13/100
 - 0s - loss: 0.0046
Epoch 14/100
 - 0s - loss: 0.0043
Epoch 15/100
 - 0s - loss: 0.0040
Epoch 16/100
 - 0s - loss: 0.0039
Epoch 17/100
 - 0s - loss: 0.0037
Epoch 18/100
 - 0s - loss: 0.0037
Epoch 19/100
 - 0s - loss: 0.0036
Epoch 20/100
 - 0s - loss: 0.0035
Epoch 21/100
 - 0s - loss: 0.0035
Epoch 22/100
```

```
 - 0s - loss: 0.0034
Epoch 23/100
 - 0s - loss: 0.0034
Epoch 24/100
 - 0s - loss: 0.0034
Epoch 25/100
 - 0s - loss: 0.0033
Epoch 26/100
 - 0s - loss: 0.0034
Epoch 27/100
 - 0s - loss: 0.0034
Epoch 28/100
 - 0s - loss: 0.0034
Epoch 29/100
 - 0s - loss: 0.0033
Epoch 30/100
 - 0s - loss: 0.0032
Epoch 31/100
 - 0s - loss: 0.0033
Epoch 32/100
 - 0s - loss: 0.0032
Epoch 33/100
 - 0s - loss: 0.0032
Epoch 34/100
 - 0s - loss: 0.0032
Epoch 35/100
 - 0s - loss: 0.0032
Epoch 36/100
 - 0s - loss: 0.0031
Epoch 37/100
 - 0s - loss: 0.0031
Epoch 38/100
 - 0s - loss: 0.0031
Epoch 39/100
 - 0s - loss: 0.0031
Epoch 40/100
 - 0s - loss: 0.0030
Epoch 41/100
 - 0s - loss: 0.0031
Epoch 42/100
 - 0s - loss: 0.0031
Epoch 43/100
 - 0s - loss: 0.0030
Epoch 44/100
 - 0s - loss: 0.0030
Epoch 45/100
 - 0s - loss: 0.0029
Epoch 46/100
```

```
 - 0s - loss: 0.0031
Epoch 47/100
 - 0s - loss: 0.0029
Epoch 48/100
 - 0s - loss: 0.0029
Epoch 49/100
 - 0s - loss: 0.0029
Epoch 50/100
 - 0s - loss: 0.0028
Epoch 51/100
 - 0s - loss: 0.0028
Epoch 52/100
 - 0s - loss: 0.0028
Epoch 53/100
 - 0s - loss: 0.0028
Epoch 54/100
 - 0s - loss: 0.0027
Epoch 55/100
 - 0s - loss: 0.0027
Epoch 56/100
 - 0s - loss: 0.0027
Epoch 57/100
 - 0s - loss: 0.0027
Epoch 58/100
 - 0s - loss: 0.0027
Epoch 59/100
 - 0s - loss: 0.0026
Epoch 60/100
 - 0s - loss: 0.0027
Epoch 61/100
 - 0s - loss: 0.0026
Epoch 62/100
 - 0s - loss: 0.0025
Epoch 63/100
 - 0s - loss: 0.0025
Epoch 64/100
 - 0s - loss: 0.0028
Epoch 65/100
 - 0s - loss: 0.0025
Epoch 66/100
 - 0s - loss: 0.0025
Epoch 67/100
 - 0s - loss: 0.0025
Epoch 68/100
 - 0s - loss: 0.0026
Epoch 69/100
 - 0s - loss: 0.0025
Epoch 70/100
```

```
 - 0s - loss: 0.0024
Epoch 71/100
 - 0s - loss: 0.0024
Epoch 72/100
 - 0s - loss: 0.0025
Epoch 73/100
 - 0s - loss: 0.0024
Epoch 74/100
 - 0s - loss: 0.0024
Epoch 75/100
 - 0s - loss: 0.0023
Epoch 76/100
 - 0s - loss: 0.0023
Epoch 77/100
 - 0s - loss: 0.0024
Epoch 78/100
 - 0s - loss: 0.0023
Epoch 79/100
 - 0s - loss: 0.0023
Epoch 80/100
 - 0s - loss: 0.0023
Epoch 81/100
 - 0s - loss: 0.0023
Epoch 82/100
 - 0s - loss: 0.0022
Epoch 83/100
 - 0s - loss: 0.0022
Epoch 84/100
 - 0s - loss: 0.0022
Epoch 85/100
 - 0s - loss: 0.0022
Epoch 86/100
 - 0s - loss: 0.0020
Epoch 87/100
 - 0s - loss: 0.0023
Epoch 88/100
 - 0s - loss: 0.0021
Epoch 89/100
 - 0s - loss: 0.0021
Epoch 90/100
 - 0s - loss: 0.0021
Epoch 91/100
 - 0s - loss: 0.0021
Epoch 92/100
 - 0s - loss: 0.0021
Epoch 93/100
 - 0s - loss: 0.0021
Epoch 94/100
```

```
 - 0s - loss: 0.0020
Epoch 95/100
 - 0s - loss: 0.0021
Epoch 96/100
 - 0s - loss: 0.0020
Epoch 97/100
 - 0s - loss: 0.0020
Epoch 98/100
 - 0s - loss: 0.0020
Epoch 99/100
 - 0s - loss: 0.0021
Epoch 100/100
 - 0s - loss: 0.0019
Train Score: 23.63 RMSE
Test Score: 69.74 RMSE
```



1.9 LSTM for Regression with Time Steps

models use time steps instead of observations. the time steps are not always equal

we do this by setting time steps as columns instead of observations

[14]:
```
# code that is changed from other examples
# reshape input to be [samples, time steps, features]
# trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
# testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))
```

[16]:

```python
# LSTM for international airline passengers problem with time step regression␣
↪framing
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
# convert an array of values into a dataset matrix
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)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('https://raw.githubusercontent.com/jbrownlee/Datasets/
↪master/airline-passengers.csv', usecols=[1], engine='python')
dataset = dataframe.values
dataset = dataset.astype('float32')
# 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[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
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()
model.add(LSTM(4, input_shape=(look_back, 1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
# make predictions
```

```python
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
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))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] =␣
 ↪testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

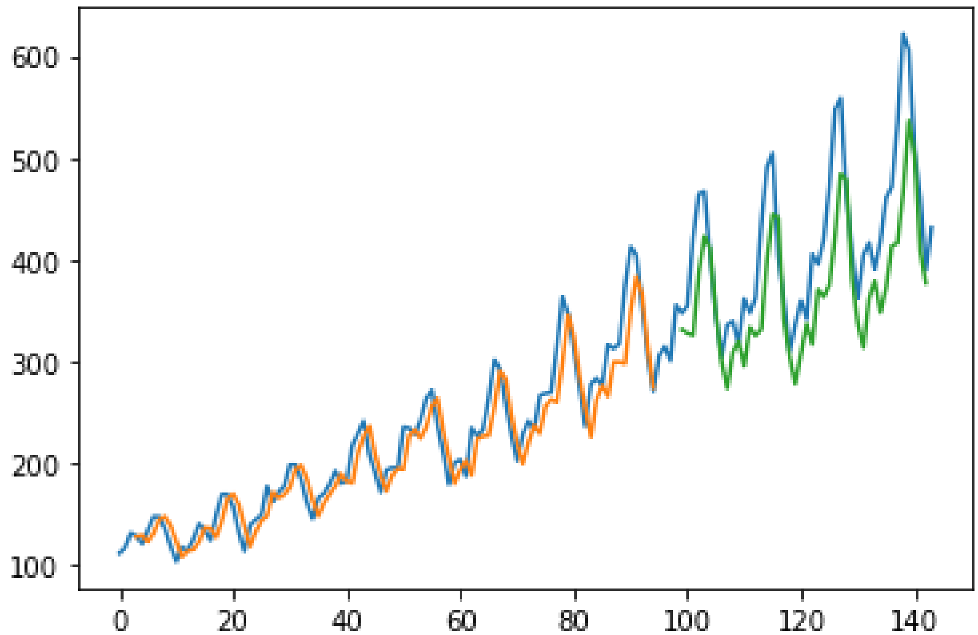
```
Epoch 1/100
 - 3s - loss: 0.0313
Epoch 2/100
 - 4s - loss: 0.0131
Epoch 3/100
 - 3s - loss: 0.0110
Epoch 4/100
 - 1s - loss: 0.0097
Epoch 5/100
 - 1s - loss: 0.0084
Epoch 6/100
 - 1s - loss: 0.0073
Epoch 7/100
 - 1s - loss: 0.0065
Epoch 8/100
 - 1s - loss: 0.0058
Epoch 9/100
 - 1s - loss: 0.0049
Epoch 10/100
 - 1s - loss: 0.0047
Epoch 11/100
```

```
 - 1s - loss: 0.0044
Epoch 12/100
 - 2s - loss: 0.0044
Epoch 13/100
 - 3s - loss: 0.0042
Epoch 14/100
 - 3s - loss: 0.0041
Epoch 15/100
 - 3s - loss: 0.0041
Epoch 16/100
 - 2s - loss: 0.0041
Epoch 17/100
 - 2s - loss: 0.0039
Epoch 18/100
 - 3s - loss: 0.0041
Epoch 19/100
 - 3s - loss: 0.0041
Epoch 20/100
 - 5s - loss: 0.0040
Epoch 21/100
 - 6s - loss: 0.0040
Epoch 22/100
 - 4s - loss: 0.0039
Epoch 23/100
 - 1s - loss: 0.0039
Epoch 24/100
 - 4s - loss: 0.0040
Epoch 25/100
 - 4s - loss: 0.0039
Epoch 26/100
 - 4s - loss: 0.0040
Epoch 27/100
 - 5s - loss: 0.0039
Epoch 28/100
 - 5s - loss: 0.0040
Epoch 29/100
 - 4s - loss: 0.0038
Epoch 30/100
 - 5s - loss: 0.0038
Epoch 31/100
 - 4s - loss: 0.0039
Epoch 32/100
 - 5s - loss: 0.0037
Epoch 33/100
 - 4s - loss: 0.0038
Epoch 34/100
 - 5s - loss: 0.0038
Epoch 35/100
```

```
 - 4s - loss: 0.0037
Epoch 36/100
 - 4s - loss: 0.0037
Epoch 37/100
 - 4s - loss: 0.0037
Epoch 38/100
 - 3s - loss: 0.0036
Epoch 39/100
 - 1s - loss: 0.0036
Epoch 40/100
 - 2s - loss: 0.0036
Epoch 41/100
 - 8s - loss: 0.0038
Epoch 42/100
 - 7s - loss: 0.0037
Epoch 43/100
 - 8s - loss: 0.0036
Epoch 44/100
 - 8s - loss: 0.0036
Epoch 45/100
 - 4s - loss: 0.0036
Epoch 46/100
 - 6s - loss: 0.0038
Epoch 47/100
 - 7s - loss: 0.0035
Epoch 48/100
 - 6s - loss: 0.0035
Epoch 49/100
 - 3s - loss: 0.0035
Epoch 50/100
 - 4s - loss: 0.0035
Epoch 51/100
 - 4s - loss: 0.0035
Epoch 52/100
 - 3s - loss: 0.0035
Epoch 53/100
 - 2s - loss: 0.0035
Epoch 54/100
 - 3s - loss: 0.0034
Epoch 55/100
 - 4s - loss: 0.0034
Epoch 56/100
 - 3s - loss: 0.0033
Epoch 57/100
 - 3s - loss: 0.0033
Epoch 58/100
 - 3s - loss: 0.0034
Epoch 59/100
```

```
 - 3s - loss: 0.0033
Epoch 60/100
 - 3s - loss: 0.0034
Epoch 61/100
 - 3s - loss: 0.0033
Epoch 62/100
 - 3s - loss: 0.0032
Epoch 63/100
 - 2s - loss: 0.0031
Epoch 64/100
 - 1s - loss: 0.0035
Epoch 65/100
 - 1s - loss: 0.0032
Epoch 66/100
 - 1s - loss: 0.0032
Epoch 67/100
 - 1s - loss: 0.0031
Epoch 68/100
 - 1s - loss: 0.0032
Epoch 69/100
 - 2s - loss: 0.0031
Epoch 70/100
 - 3s - loss: 0.0031
Epoch 71/100
 - 3s - loss: 0.0030
Epoch 72/100
 - 1s - loss: 0.0031
Epoch 73/100
 - 1s - loss: 0.0030
Epoch 74/100
 - 1s - loss: 0.0030
Epoch 75/100
 - 3s - loss: 0.0029
Epoch 76/100
 - 3s - loss: 0.0029
Epoch 77/100
 - 4s - loss: 0.0029
Epoch 78/100
 - 3s - loss: 0.0029
Epoch 79/100
 - 1s - loss: 0.0029
Epoch 80/100
 - 1s - loss: 0.0028
Epoch 81/100
 - 3s - loss: 0.0027
Epoch 82/100
 - 3s - loss: 0.0026
Epoch 83/100
```

```
 - 3s - loss: 0.0026
Epoch 84/100
 - 1s - loss: 0.0025
Epoch 85/100
 - 1s - loss: 0.0025
Epoch 86/100
 - 1s - loss: 0.0023
Epoch 87/100
 - 3s - loss: 0.0025
Epoch 88/100
 - 4s - loss: 0.0024
Epoch 89/100
 - 3s - loss: 0.0023
Epoch 90/100
 - 1s - loss: 0.0023
Epoch 91/100
 - 1s - loss: 0.0022
Epoch 92/100
 - 1s - loss: 0.0022
Epoch 93/100
 - 2s - loss: 0.0022
Epoch 94/100
 - 2s - loss: 0.0021
Epoch 95/100
 - 2s - loss: 0.0021
Epoch 96/100
 - 2s - loss: 0.0021
Epoch 97/100
 - 2s - loss: 0.0021
Epoch 98/100
 - 2s - loss: 0.0020
Epoch 99/100
 - 1s - loss: 0.0021
Epoch 100/100
 - 1s - loss: 0.0020
Train Score: 23.95 RMSE
Test Score: 63.33 RMSE
```



1.10 LSTM with Memory Between Batches

we should take advantage of the fact that LSTMs can remember info and train their own internal weights. instead of resetting the LSTM's between batches, we should keep those weights

[17]:
```
# LSTM for international airline passengers problem with memory
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
# convert an array of values into a dataset matrix
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)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
```

```
dataframe = read_csv('https://raw.githubusercontent.com/jbrownlee/Datasets/
↪master/airline-passengers.csv', usecols=[1], engine='python')
dataset = dataframe.values
dataset = dataset.astype('float32')
# 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[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
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
batch_size = 1
model = Sequential()
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
for i in range(100):
        model.fit(trainX, trainY, epochs=1, batch_size=batch_size, verbose=2,␣
↪shuffle=False)
        model.reset_states()
# make predictions
trainPredict = model.predict(trainX, batch_size=batch_size)
model.reset_states()
testPredict = model.predict(testX, batch_size=batch_size)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
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))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
```

```
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] =␣
↪testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

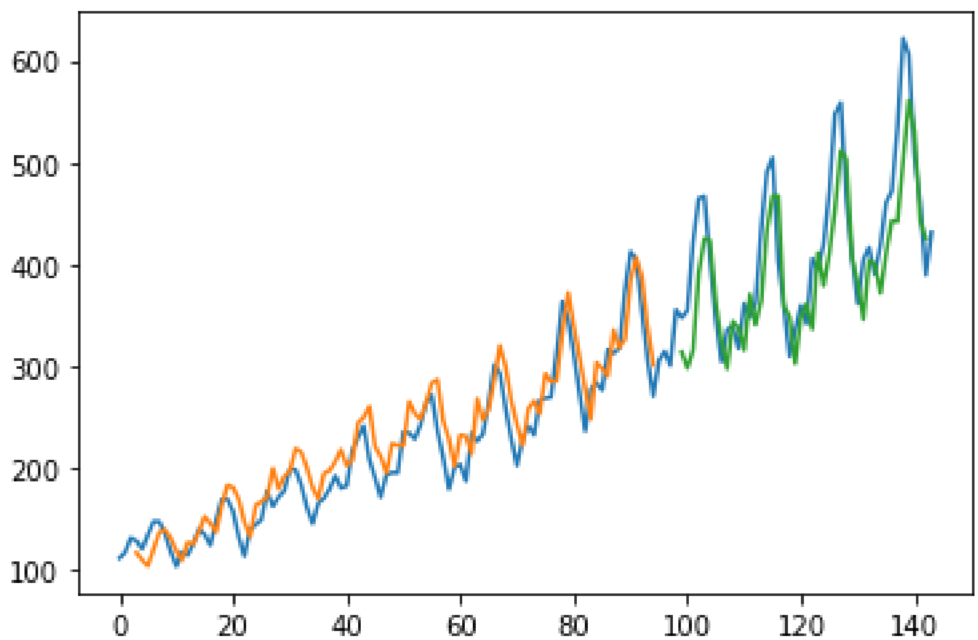
```
Epoch 1/1
 - 2s - loss: 0.0065
Epoch 1/1
 - 1s - loss: 0.0187
Epoch 1/1
 - 1s - loss: 0.0105
Epoch 1/1
 - 1s - loss: 0.0070
Epoch 1/1
 - 1s - loss: 0.0056
Epoch 1/1
 - 1s - loss: 0.0052
Epoch 1/1
 - 2s - loss: 0.0052
Epoch 1/1
 - 2s - loss: 0.0052
Epoch 1/1
 - 1s - loss: 0.0051
Epoch 1/1
 - 7s - loss: 0.0051
Epoch 1/1
 - 5s - loss: 0.0051
Epoch 1/1
 - 6s - loss: 0.0050
Epoch 1/1
 - 5s - loss: 0.0050
Epoch 1/1
 - 7s - loss: 0.0049
Epoch 1/1
 - 2s - loss: 0.0049
Epoch 1/1
 - 3s - loss: 0.0049
Epoch 1/1
 - 3s - loss: 0.0048
Epoch 1/1
 - 3s - loss: 0.0048
Epoch 1/1
 - 5s - loss: 0.0048
```

```
Epoch 1/1
 - 5s - loss: 0.0047
Epoch 1/1
 - 4s - loss: 0.0047
Epoch 1/1
 - 5s - loss: 0.0047
Epoch 1/1
 - 5s - loss: 0.0047
Epoch 1/1
 - 8s - loss: 0.0046
Epoch 1/1
 - 8s - loss: 0.0046
Epoch 1/1
 - 4s - loss: 0.0046
Epoch 1/1
 - 2s - loss: 0.0045
Epoch 1/1
 - 2s - loss: 0.0045
Epoch 1/1
 - 2s - loss: 0.0045
Epoch 1/1
 - 1s - loss: 0.0044
Epoch 1/1
 - 1s - loss: 0.0044
Epoch 1/1
 - 1s - loss: 0.0044
Epoch 1/1
 - 3s - loss: 0.0043
Epoch 1/1
 - 4s - loss: 0.0043
Epoch 1/1
 - 5s - loss: 0.0043
Epoch 1/1
 - 2s - loss: 0.0043
Epoch 1/1
 - 1s - loss: 0.0042
Epoch 1/1
 - 2s - loss: 0.0042
Epoch 1/1
 - 1s - loss: 0.0041
Epoch 1/1
 - 5s - loss: 0.0041
Epoch 1/1
 - 6s - loss: 0.0041
Epoch 1/1
 - 4s - loss: 0.0040
Epoch 1/1
 - 4s - loss: 0.0040
```

```
Epoch 1/1
 - 5s - loss: 0.0039
Epoch 1/1
 - 6s - loss: 0.0039
Epoch 1/1
 - 6s - loss: 0.0038
Epoch 1/1
 - 5s - loss: 0.0038
Epoch 1/1
 - 6s - loss: 0.0037
Epoch 1/1
 - 6s - loss: 0.0037
Epoch 1/1
 - 2s - loss: 0.0036
Epoch 1/1
 - 6s - loss: 0.0035
Epoch 1/1
 - 6s - loss: 0.0035
Epoch 1/1
 - 5s - loss: 0.0034
Epoch 1/1
 - 5s - loss: 0.0033
Epoch 1/1
 - 5s - loss: 0.0032
Epoch 1/1
 - 6s - loss: 0.0031
Epoch 1/1
 - 5s - loss: 0.0030
Epoch 1/1
 - 5s - loss: 0.0029
Epoch 1/1
 - 4s - loss: 0.0029
Epoch 1/1
 - 5s - loss: 0.0028
Epoch 1/1
 - 4s - loss: 0.0027
Epoch 1/1
 - 5s - loss: 0.0026
Epoch 1/1
 - 6s - loss: 0.0026
Epoch 1/1
 - 8s - loss: 0.0025
Epoch 1/1
 - 7s - loss: 0.0024
Epoch 1/1
 - 0s - loss: 0.0024
Epoch 1/1
 - 8s - loss: 0.0023
```

```
Epoch 1/1
 - 9s - loss: 0.0022
Epoch 1/1
 - 7s - loss: 0.0022
Epoch 1/1
 - 6s - loss: 0.0022
Epoch 1/1
 - 5s - loss: 0.0021
Epoch 1/1
 - 7s - loss: 0.0021
Epoch 1/1
 - 8s - loss: 0.0020
Epoch 1/1
 - 7s - loss: 0.0020
Epoch 1/1
 - 4s - loss: 0.0020
Epoch 1/1
 - 5s - loss: 0.0020
Epoch 1/1
 - 6s - loss: 0.0019
Epoch 1/1
 - 6s - loss: 0.0019
Epoch 1/1
 - 5s - loss: 0.0019
Epoch 1/1
 - 6s - loss: 0.0019
Epoch 1/1
 - 7s - loss: 0.0019
Epoch 1/1
 - 6s - loss: 0.0019
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 8s - loss: 0.0018
```

```
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 7s - loss: 0.0018
Epoch 1/1
 - 7s - loss: 0.0018
Epoch 1/1
 - 6s - loss: 0.0018
Epoch 1/1
 - 5s - loss: 0.0018
Epoch 1/1
 - 5s - loss: 0.0018
Epoch 1/1
 - 5s - loss: 0.0018
Train Score: 25.60 RMSE
Test Score: 48.98 RMSE
```



1.11 Stacked LSTMs with Memory Between Batches

now lets stack those stateful LSTM layers

[18]:
```
# model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1),␣
↪stateful=True, return_sequences=True))
# model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1),␣
↪stateful=True))
```

[19]:
```
# Stacked LSTM for international airline passengers problem with memory
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
# convert an array of values into a dataset matrix
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)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('https://raw.githubusercontent.com/jbrownlee/Datasets/
↪master/airline-passengers.csv', usecols=[1], engine='python')
dataset = dataframe.values
dataset = dataset.astype('float32')
# 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[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
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
batch_size = 1
model = Sequential()
```

```
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True,␣
↪return_sequences=True))
model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
for i in range(100):
        model.fit(trainX, trainY, epochs=1, batch_size=batch_size, verbose=2,␣
↪shuffle=False)
        model.reset_states()
# make predictions
trainPredict = model.predict(trainX, batch_size=batch_size)
model.reset_states()
testPredict = model.predict(testX, batch_size=batch_size)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
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))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] =␣
↪testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

```
Epoch 1/1
 - 5s - loss: 0.0059
Epoch 1/1
 - 3s - loss: 0.0133
Epoch 1/1
 - 3s - loss: 0.0077
Epoch 1/1
 - 4s - loss: 0.0060
Epoch 1/1
```

```
 - 5s - loss: 0.0058
Epoch 1/1
 - 5s - loss: 0.0059
Epoch 1/1
 - 4s - loss: 0.0059
Epoch 1/1
 - 5s - loss: 0.0059
Epoch 1/1
 - 7s - loss: 0.0059
Epoch 1/1
 - 7s - loss: 0.0059
Epoch 1/1
 - 5s - loss: 0.0059
Epoch 1/1
 - 6s - loss: 0.0059
Epoch 1/1
 - 1s - loss: 0.0058
Epoch 1/1
 - 2s - loss: 0.0058
Epoch 1/1
 - 1s - loss: 0.0058
Epoch 1/1
 - 1s - loss: 0.0058
Epoch 1/1
 - 1s - loss: 0.0058
Epoch 1/1
 - 1s - loss: 0.0058
Epoch 1/1
 - 1s - loss: 0.0058
Epoch 1/1
 - 1s - loss: 0.0058
Epoch 1/1
 - 2s - loss: 0.0057
Epoch 1/1
 - 5s - loss: 0.0057
Epoch 1/1
 - 4s - loss: 0.0057
Epoch 1/1
 - 4s - loss: 0.0057
Epoch 1/1
 - 5s - loss: 0.0057
Epoch 1/1
 - 3s - loss: 0.0057
Epoch 1/1
 - 2s - loss: 0.0056
Epoch 1/1
 - 3s - loss: 0.0056
Epoch 1/1
```

```
 - 3s - loss: 0.0056
Epoch 1/1
 - 4s - loss: 0.0056
Epoch 1/1
 - 6s - loss: 0.0056
Epoch 1/1
 - 1s - loss: 0.0055
Epoch 1/1
 - 7s - loss: 0.0055
Epoch 1/1
 - 5s - loss: 0.0055
Epoch 1/1
 - 3s - loss: 0.0055
Epoch 1/1
 - 4s - loss: 0.0054
Epoch 1/1
 - 7s - loss: 0.0054
Epoch 1/1
 - 3s - loss: 0.0054
Epoch 1/1
 - 3s - loss: 0.0054
Epoch 1/1
 - 3s - loss: 0.0053
Epoch 1/1
 - 2s - loss: 0.0053
Epoch 1/1
 - 3s - loss: 0.0052
Epoch 1/1
 - 4s - loss: 0.0052
Epoch 1/1
 - 3s - loss: 0.0052
Epoch 1/1
 - 3s - loss: 0.0051
Epoch 1/1
 - 4s - loss: 0.0050
Epoch 1/1
 - 4s - loss: 0.0050
Epoch 1/1
 - 4s - loss: 0.0049
Epoch 1/1
 - 2s - loss: 0.0049
Epoch 1/1
 - 3s - loss: 0.0048
Epoch 1/1
 - 4s - loss: 0.0047
Epoch 1/1
 - 4s - loss: 0.0046
Epoch 1/1
```

```
 - 3s - loss: 0.0045
Epoch 1/1
 - 5s - loss: 0.0044
Epoch 1/1
 - 4s - loss: 0.0043
Epoch 1/1
 - 4s - loss: 0.0041
Epoch 1/1
 - 3s - loss: 0.0040
Epoch 1/1
 - 2s - loss: 0.0038
Epoch 1/1
 - 2s - loss: 0.0037
Epoch 1/1
 - 4s - loss: 0.0035
Epoch 1/1
 - 4s - loss: 0.0034
Epoch 1/1
 - 3s - loss: 0.0032
Epoch 1/1
 - 4s - loss: 0.0031
Epoch 1/1
 - 2s - loss: 0.0029
Epoch 1/1
 - 4s - loss: 0.0028
Epoch 1/1
 - 3s - loss: 0.0027
Epoch 1/1
 - 2s - loss: 0.0026
Epoch 1/1
 - 5s - loss: 0.0025
Epoch 1/1
 - 3s - loss: 0.0025
Epoch 1/1
 - 6s - loss: 0.0024
Epoch 1/1
 - 6s - loss: 0.0024
Epoch 1/1
 - 2s - loss: 0.0023
Epoch 1/1
 - 4s - loss: 0.0023
Epoch 1/1
 - 3s - loss: 0.0022
Epoch 1/1
 - 2s - loss: 0.0022
Epoch 1/1
 - 3s - loss: 0.0022
Epoch 1/1
```

```
 - 5s - loss: 0.0021
Epoch 1/1
 - 2s - loss: 0.0021
Epoch 1/1
 - 3s - loss: 0.0021
Epoch 1/1
 - 6s - loss: 0.0020
Epoch 1/1
 - 4s - loss: 0.0020
Epoch 1/1
 - 3s - loss: 0.0020
Epoch 1/1
 - 2s - loss: 0.0020
Epoch 1/1
 - 3s - loss: 0.0019
Epoch 1/1
 - 4s - loss: 0.0019
Epoch 1/1
 - 3s - loss: 0.0019
Epoch 1/1
 - 3s - loss: 0.0021
Epoch 1/1
 - 3s - loss: 0.0024
Epoch 1/1
 - 3s - loss: 0.0030
Epoch 1/1
 - 3s - loss: 0.0024
Epoch 1/1
 - 4s - loss: 0.0021
Epoch 1/1
 - 4s - loss: 0.0023
Epoch 1/1
 - 2s - loss: 0.0026
Epoch 1/1
 - 4s - loss: 0.0020
Epoch 1/1
 - 4s - loss: 0.0019
Epoch 1/1
 - 4s - loss: 0.0019
Epoch 1/1
 - 7s - loss: 0.0019
Epoch 1/1
 - 7s - loss: 0.0020
Epoch 1/1
 - 7s - loss: 0.0019
Epoch 1/1
 - 9s - loss: 0.0017
Train Score: 21.52 RMSE
```

```
Test Score: 108.90 RMSE
```