16 - Implementing RNN

Ludwig Krippahl

Recurrent Netwoks

Summary

- Univariate time series prediction with LSTM
- Optional: Multivariate time series

Time series prediction

Time Series

- Based on tutorial by Jason Brownlee

https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

- Imports:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

- Load data (passengers per year) normalize and split

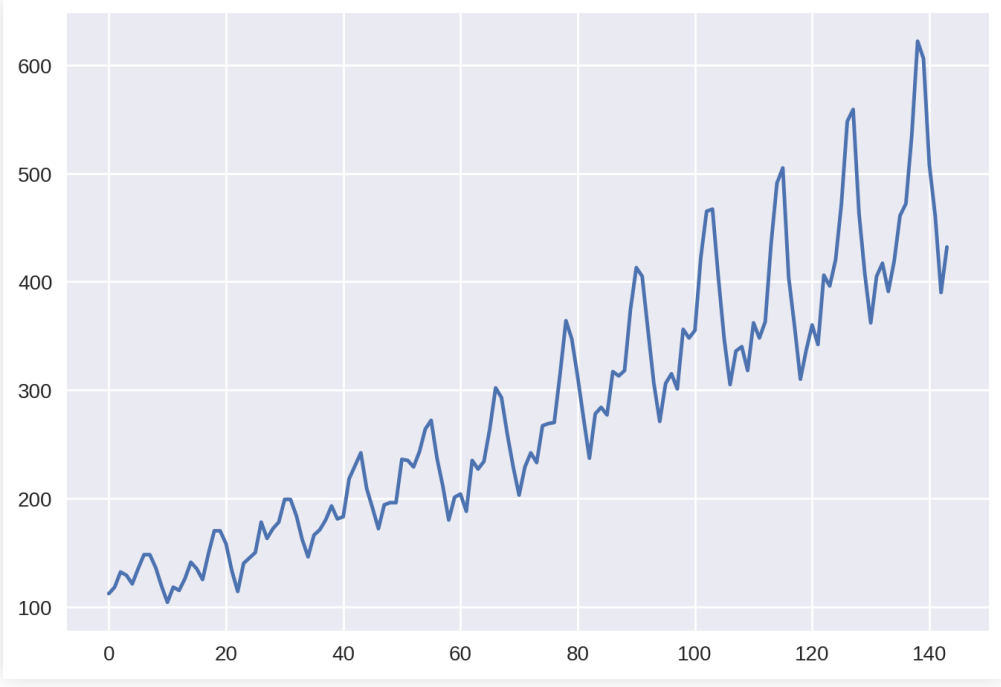
```
df = pd.read_csv('passengers.csv')
dataset = df['Passengers'].values.astype('float32')
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset.reshape(-1,1)).reshape((-1,))
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size], dataset[train_size:len(dataset)]
```

Time Series

- Time series with number of airline passengers per month

- Box & Jenkins, monthly passengers (thousands), 1949 to 1960.

```
"Month","Passengers"
"1949-01",112
"1949-02",118
"1949-03",132
"1949-04",129
"1949-05",121
"1949-06",135
"1949-07",148
"1949-08",148
"1949-09",136
"1949-10",119
"1949-11",104
"1949-12",118
"1950-01",115
"1950-02",126
```

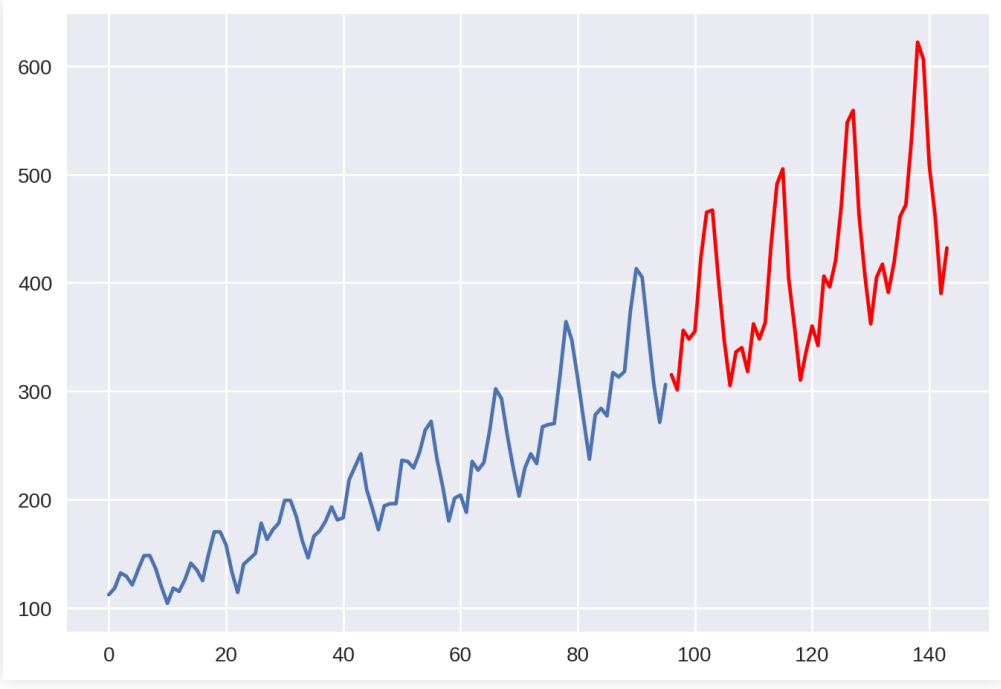


Time Series

- Split 67% for training, 33% for testing

- In time series prediction it is best to leave last for testing
- Sampling at random would not be realistic

```
"Month","Passengers"
"1949-01",112
"1949-02",118
"1949-03",132
"1949-04",129
"1949-05",121
"1949-06",135
"1949-07",148
"1949-08",148
"1949-09",136
"1949-10",119
"1949-11",104
"1949-12",118
"1950-01",115
"1950-02",126
```



Time Series

- Create samples of sequences (e.g. length 3)

- Note: we use `look_back` points to predict the next one

```
def create_dataset(dataset, look_back=1):
    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 np.array(dataX), np.array(dataY)
```

```
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

- Reshape to [samples, time steps, features] (-1, 3, 1)

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

Time Series

- Create model and fit. We will use a stateful LSTM,

- This way it preserves states across batches
- We cannot shuffle the samples and must reset the states after each epoch

```
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()
```

- LSTM layer returns only last output of 4 cells

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (1, 4)                    96
_________________________________________________________________
dense_3 (Dense)              (1, 1)                    5
=================================================================
```

Time Series

- Prediction and rescaling:

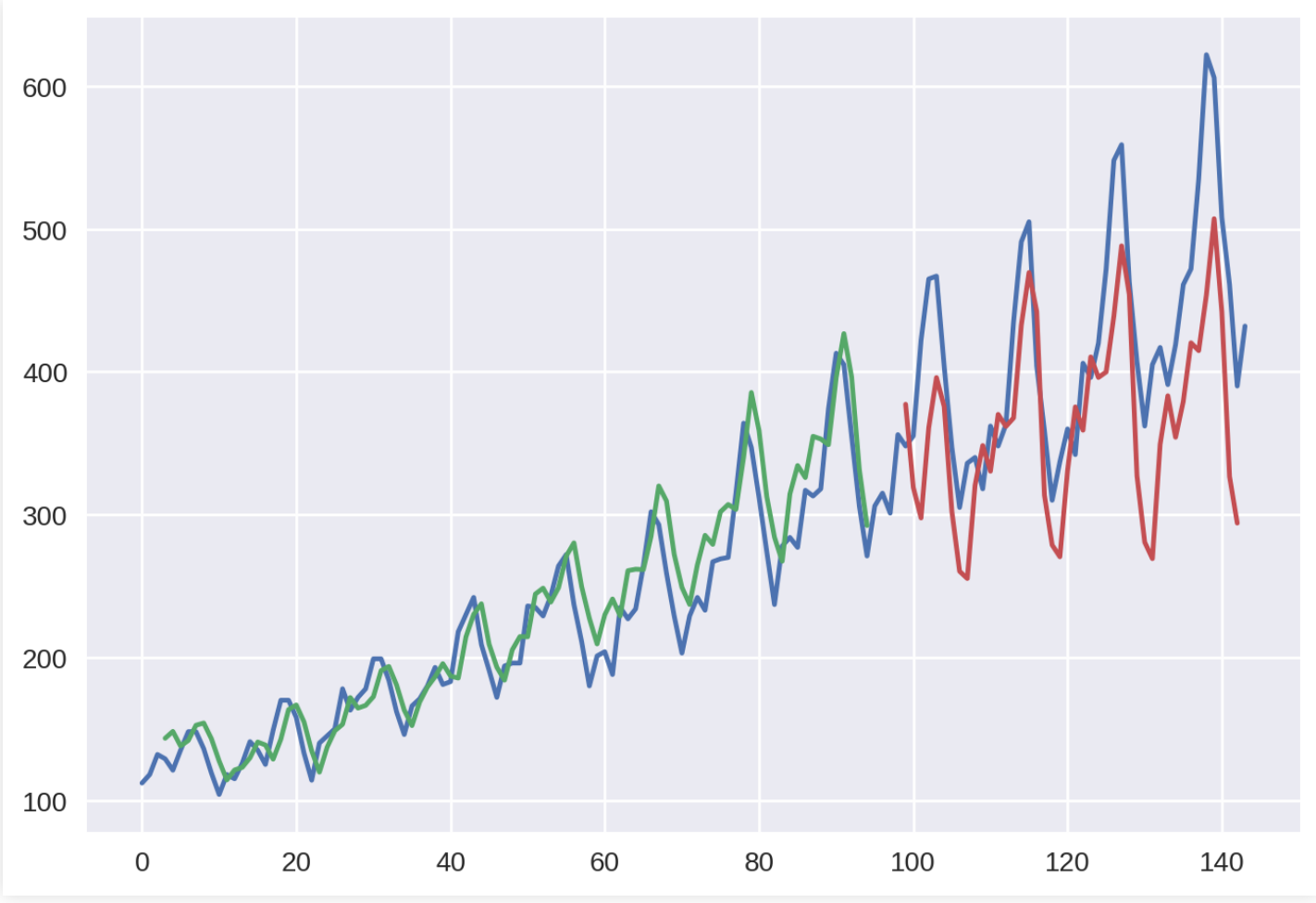
```
trainPredict = model.predict(trainX, batch_size=batch_size)
model.reset_states()
testPredict = model.predict(testX, batch_size=batch_size)
trainPredict = scaler.inverse_transform(trainPredict).reshape((-1,))
trainY = scaler.inverse_transform(trainY.reshape(-1,1)).reshape((-1,))
testPredict = scaler.inverse_transform(testPredict).reshape((-1,))
testY = scaler.inverse_transform(testY.reshape(-1,1)).reshape((-1,))
trainScore = mean_squared_error(trainY, trainPredict)**0.5
testScore = mean_squared_error(testY, testPredict)**0.5
```

- Plotting the predictions:

```
trainPredictPlot = np.empty_like(dataset)
trainPredictPlot[:] = np.nan
trainPredictPlot[look_back:len(trainPredict)+look_back] = trainPredict
testPredictPlot = np.empty_like(dataset)
testPredictPlot[:] = np.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1] = testPredict
plt.plot(scaler.inverse_transform(dataset.reshape(-1,1)))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
```

Time Series

- Plotting the predictions:



Time Series

Stacking LSTM layers

- In this case you will probably just get more overfitting

```
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')
```

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_6 (LSTM)                (1, 3, 4)                 96
_________________________________________________________________
lstm_7 (LSTM)                (1, 4)                    144
_________________________________________________________________
dense_5 (Dense)              (1, 1)                    5
=================================================================
```

Optional exercise

- Multivariate time series

- (more of the same)

- Jason Brownlee:

https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/