Published in Towards Data Science

George V Jose Follow

Sep 20, 2019 · 5 min read · Listen

Predicting Sequential Data using LSTM: An Introduction

Predicting the future of sequential data like stocks using Long Short Term Memory (LSTM) networks.

Photo by Chris Liverani on Unsplash

Forecasting is the process of predicting the future using current and previous proven to be better in understanding the patterns in both structured and unstructured data.

To understand the patterns in a long sequence of data, we need networks to analyse patterns across time. Recurrent Networks is the one usually used for learning such data. They are capable of understanding long and short term dependencies or temporal differences.

Alright, no more intro…

This post will show you how to implement a forecasting model using LSTM networks in **Keras** and with some cool visualizations. We'll be using the stock price of Google from yahoo finance but feel free to use any stock data that you like.

Implementation

I have used colab to implement this code to make the visualizations easier, you can use your preferred method. We'll start off with importing necessary libraries:

```
import pandas as pd
import numpy as np
import keras
import tensorflow as tf
from keras.preprocessing.sequence import TimeseriesGenerator
```

TS_import.py hosted with ❤ by GitHub view raw

After you've downloaded your .csv file from yahoo finance or your source, load the data using pandas.

```
print(df.info())
```

TS_data_read.py hosted with ❤ by GitHub view raw

The info of dataframe shows somewhat like this:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3797 entries, 0 to 3796
Data columns (total 7 columns):
Date         3797 non-null object
Open         3797 non-null float64
High         3797 non-null float64
Low          3797 non-null float64
Close        3797 non-null float64
Adj Close    3797 non-null float64
Volume       3797 non-null int64
dtypes: float64(5), int64(1), object(1)
memory usage: 207.7+ KB
```

For this tutorial, we require only Date and Close columns, everything else can be dropped.

```python
df['Date'] = pd.to_datetime(df['Date'])
df.set_axis(df['Date'], inplace=True)
df.drop(columns=['Open', 'High', 'Low', 'Volume'], inplace=True)
```

TS_df_preprocess.py hosted with ❤ by GitHub view raw

Before we do the training and predictions, let's see how the data looks like. For all the visualizations, I'm using the Plotly python library. Why Plotly?… cause its simply the best graphing library and it can produce some good looking graphs.

With plotly, we can define a trace and the layout and it does everything else.

The graph is oscillating from 2018 and the sequence is not smooth… Moving on.

Data Preprocessing

For our analysis, let train the model on the first 80% of data and test it on the remaining 20%.

```python
close_data = df['Close'].values
close_data = close_data.reshape((-1,1))

split_percent = 0.80
split = int(split_percent*len(close_data))

close_train = close_data[:split]
close_test = close_data[split:]

date_train = df['Date'][:split]
date_test = df['Date'][split:]

print(len(close_train))
print(len(close_test))
```

TS_data_preprocess1.py hosted with ❤ by GitHub view raw

trained as a supervised model. Thus we need to convert the data from sequence to supervised data 😱.

Let me explain, training a neural network of any machine learning model requires the data to be in {<*features*>,<*target*>} format. Similarly, we need to convert the given data into this format. Here, we introduce a concept of a look back.

> *Look back is nothing but the number of previous days' data to use, to predict the value for the next day. For example, let us say look back is 2; so in order to predict the stock price for tomorrow, we need the stock price of today and yesterday.*

Coming back to the format, at a given day *x(t)*, the *features* are the values of *x(t-1), x(t-2), …., x(t-n)* where n is look back.

So if our data is like this,

```
[2, 3, 4, 5, 4, 6, 7, 6, 8, 9]
```

the required data format (n=3) would be this:

```
[2, 3, 4] -> [5]
[3, 4, 5] -> [4]
[4, 5, 4] -> [6]
[5, 4, 6] -> [7]
[4, 6, 7] -> [6]
[6, 7, 6] -> [8]
[7, 6, 8] -> [9]
```

Phew 😓.

```python
look_back = 15

train_generator = TimeseriesGenerator(close_train, close_train, length=look_back, batch_size=20)
test_generator = TimeseriesGenerator(close_test, close_test, length=look_back, batch_size=1)
```

TS_data_preprocess2.py hosted with ❤ by GitHub view raw

I've set look_back as 15, but you can play around with that value.

Neural Network

Now that our data is ready, we can move on to creating and training our network.

```python
from keras.models import Sequential
from keras.layers import LSTM, Dense

model = Sequential()
model.add(
    LSTM(10,
        activation='relu',
        input_shape=(look_back,1))
)
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

num_epochs = 25
model.fit_generator(train_generator, epochs=num_epochs, verbose=1)
```

TS_nn.py hosted with ❤ by GitHub view raw

A simple architecture of LSTM units trained using Adam optimizer and Mean Squared Loss function for 25 epochs. Note that instead of using model.fit(), we use model.fit_generator() because we have created a data generator.

To know more about LSTM network, see this awesome blog post.

Prediction

```python
prediction = model.predict_generator(test_generator)

close_train = close_train.reshape((-1))
close_test = close_test.reshape((-1))
prediction = prediction.reshape((-1))

trace1 = go.Scatter(
    x = date_train,
    y = close_train,
    mode = 'lines',
    name = 'Data'
)
trace2 = go.Scatter(
    x = date_test,
    y = prediction,
    mode = 'lines',
    name = 'Prediction'
)
trace3 = go.Scatter(
    x = date_test,
    y = close_test,
    mode='lines',
    name = 'Ground Truth'
)
layout = go.Layout(
    title = "Google Stock",
    xaxis = {'title' : "Date"},
    yaxis = {'title' : "Close"}
)
fig = go.Figure(data=[trace1, trace2, trace3], layout=layout)
fig.show()
```

TS_prediction_plot.py hosted with ❤ by **GitHub** **view raw**

Rather than computing loss between predicted and actual values, we can plot it.

From the graph, we can see that prediction and the actual value(ground truth) somewhat overlap. But if you zoom in, the fit is not perfect. We should expect this because it is inevitable as we are performing prediction.

Forecasting

Our testing shows the model is somewhat good. So we can move on to predicting the future or forecasting.

> *Foreshadowing: Since we are attempting to predict the future, there will be a great amount of uncertainty in the prediction.*

Predicting the future is easy… To predict tomorrow's value, feed into the model the past *n*(look_back) days' values and we get tomorrow's value as output. To get the day after tomorrow's value, feed-in past n-1 days' values along with tomorrow's value and the model output day after tomorrow's value.

Forecasting for longer duration is not feasible. So, let's forecast a months stock

```python
def predict(num_prediction, model):
    prediction_list = close_data[-look_back:]

    for _ in range(num_prediction):
        x = prediction_list[-look_back:]
        x = x.reshape((1, look_back, 1))
        out = model.predict(x)[0][0]
        prediction_list = np.append(prediction_list, out)
    prediction_list = prediction_list[look_back-1:]

    return prediction_list

def predict_dates(num_prediction):
    last_date = df['Date'].values[-1]
    prediction_dates = pd.date_range(last_date, periods=num_prediction+1).tolist()
    return prediction_dates

num_prediction = 30
forecast = predict(num_prediction, model)
forecast_dates = predict_dates(num_prediction)
```

TS_forecast.py hosted with ❤ by GitHub view raw

Now plotting the future values,

When predicting the future, there is a good possibility that model output is uncertain to a great extent. The model's output is fed back into it as input. This causes the model's noise and uncertainty to be repeated and amplified.

But still, we have created a model that gives us a trend of the graphs and also the range of values that might be in the future.

Conclusion

With this model, we have created a rudimentary model that is able to forecast to a certain extent. Even though the model is not perfect, we have one that can approximate to the past data pretty well. But for new data, we would require more parameter tuning.

552 25

If there exists any problem during the training of the model, try tuning these parameters:

- *look_back*

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

Get this newsletter