RECURRENT NEURAL NETWORKS (RNN)

BIM470 NEURAL NETWORKS

TERM PROJECT

Yunus Nihat ÖZPOLAT

Abstract

Recent work in deep machine learning has led to more powerful artificial neural network designs, including Recurrent Neural Networks (RNN) that can process input sequences of arbitrary length. We focus on a special kind of RNN known as a Long-Short-Term-Memory (LSTM) network. LSTM networks have enhanced memory capability, creating the possibility of using them for learning and generating music and language. Moreover, predicting stock prices is an uncertain task which is modelled using machine learning to predict the return on stocks. There are a lot of methods and tools used for the purpose of stock market prediction. The stock market is considered to be very dynamic and complex in nature. An accurate prediction of future prices may lead to a higher yield of profit for investors through stock investments. As per the predictions, investors will be able to pick the stocks that may give a higher return.

This project focuses on the Long-Short-Term Memory (LSTM) Recurrent Neural Network, one of the popular deep learning models, used in stock market prediction.

Contents

Introduction

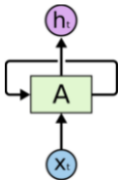
RNNs are a powerful and robust type of neural network, and belong to the most promising algorithms in use because it is the only one with an internal memory.

Like many other deep learning algorithms, recurrent neural networks are relatively old. They were initially created in the 1980's, but only in recent years have we seen their true potential. An increase in computational power along with the the massive amounts of data that we now have to work with, and the invention of long short-term memory (LSTM) in the 1990s, has really brought RNNs to the foreground.

Because of their internal memory, RNN's can remember important things about the input they received, which allows them to be very precise in predicting what's coming next. This is why they're the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more. Recurrent neural networks can form a much deeper understanding of a sequence and its context compared to other algorithms.

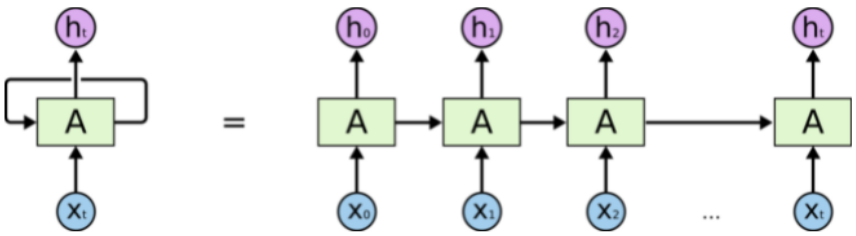
WHAT IS A RECURRENT NEURAL NETWORK (RNN)?

Recurrent neural networks (RNN) are a class of neural networks that are helpful in modeling sequence data. Derived from feedforward networks, RNNs exhibit similar behavior to how human brains function. Simply put: recurrent neural networks produce predictive results in sequential data that other algorithms can't.



Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, AA, looks at some input xtxt and outputs a value htht. A loop allows information to be passed from one step of the network to the next. These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:



An unrolled recurrent neural network.

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

How Recurrent Neural Networks Work

To understand RNNs properly, working knowledge of "normal" feedforward neural networks and sequential data should be known.

Sequential data is basically just ordered data in which related things follow each other. Examples are financial data or the DNA sequence. The most popular type of sequential data is perhaps time series data, which is just a series of data points that are listed in time order.

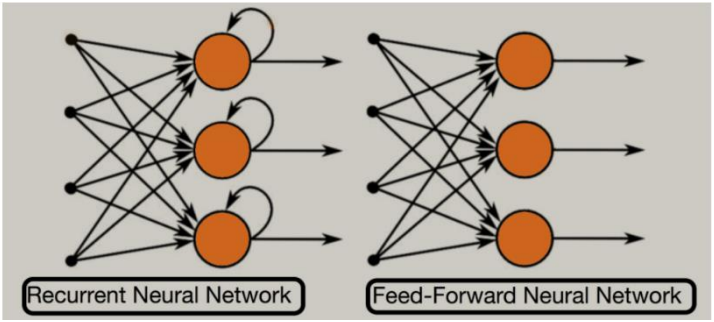
RNN VS. Feed-Forward Neural Networks

RNN's and feed-forward neuralnetworks get their names from the way they channel information.
In a feed-forward neural network, the information only moves in one direction from the input layer, through the hidden layers, to the output layer. The information moves straight through the network and never touches a node twice.

Feed-forward neural networks have no memory of the input they receive and are bad at predicting what's coming next. Because a feed-forward network only considers the current input, it has no notion of order in time. It simply can't remember anything about what happened in the past except its training.

In a RNN the information cycles through a loop. When it makes a decision, it considers the current input and also what it has learned from the inputs it received previously.

The two images below illustrate the difference in information flow between a RNN and a feed-forward neural network.



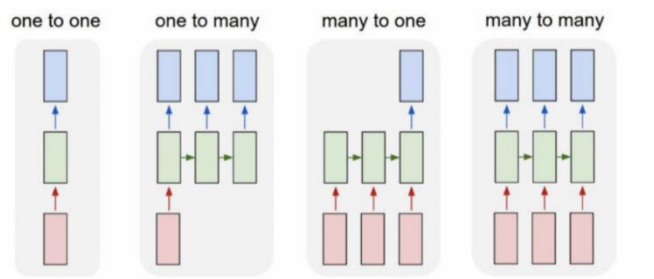
A usual RNN has a short-term memory. In combination with a LSTM they also have a long-term memory.

A recurrent neural network, however, is able to remember those characters because of its internal memory. It produces output, copies that output and loops it back into the network.

Simply put: recurrent neural networks add the immediate past to the present.

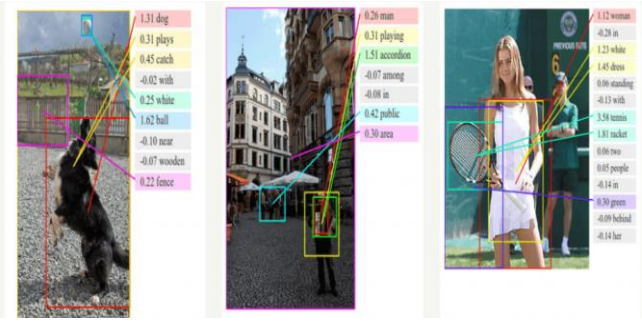
Types Of RNNs

- One to One
- One to Many
- Many to One
- Many to Many



Applications of RNN's

- Stock Price Forecasting
- Anomaly Detection
- Sentiment Analysis
- Semantic Search
- Speech recognition
- Visual Search, Face detection, OCR
- Machine Translation
- Text Generation



LSTM Networks

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

Long Short-Term Memory (LSTM) networks are an extension of RNN that extend the memory. LSTM are used as the building blocks for the layers of a RNN. LSTMs assign data "weights" which helps RNNs to either let new information in, forget information or give it importance enough to impact the output.

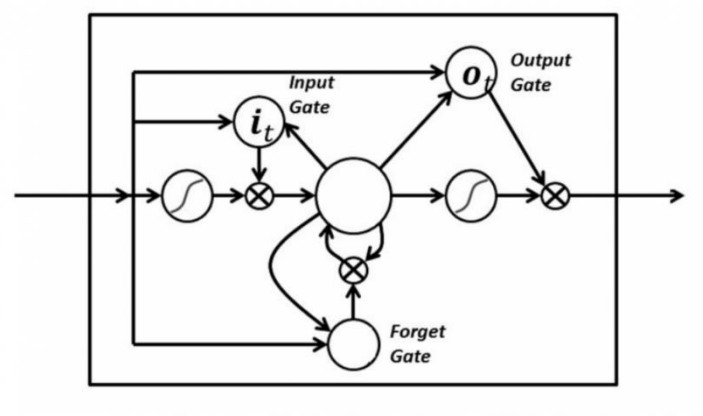
The units of an LSTM are used as building units for the layers of a RNN, often called an LSTM network.

LSTMs enable RNNs to remember inputs over a long period of time. This is because LSTMs contain information in a memory, much like the memory of a computer. The LSTM can read, write and delete information from its memory.

This memory can be seen as a gated cell, with gated meaning the cell decides whether or not to store or delete information (i.e., if it opens the gates or not), based on the importance it assigns to the information. The assigning of importance happens through weights, which are

also learned by the algorithm. This simply means that it learns over time what information is important and what is not.

In an LSTM you have three gates: input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn't important (forget gate), or let it impact the output at the current timestep (output gate). Below is an illustration of a RNN with its three gates:

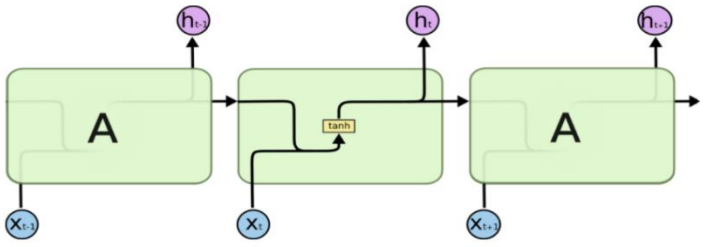


The gates in an LSTM are analog in the form of sigmoids, meaning they range from zero to one. The fact that they are analog enables them to do backpropagation.

The problematic issues of vanishing gradients is solved through LSTM because it keeps the gradients steep enough, which keeps the training relatively short and the accuracy high.

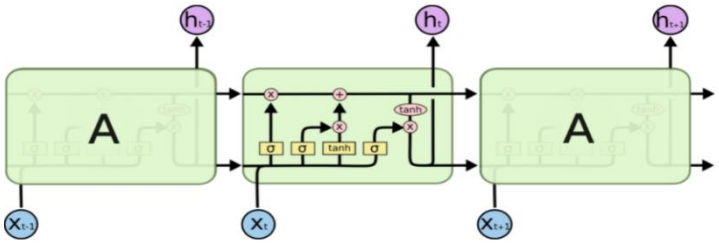
How to work LSTM ?

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

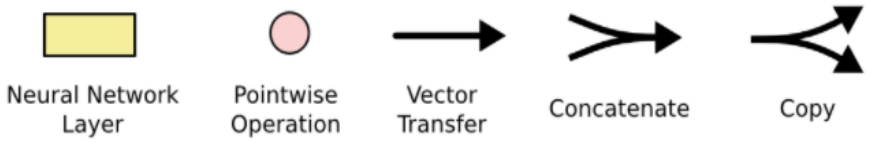


The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

LSTM Recurrent Neural Network For Stock Market Prediction (Pyhton Code)

Over the years, various machine learning techniques have been used in stock market prediction, but with the increased amount of data and expectation of more accurate prediction, the deep learning models are being used nowadays which have proven their advantage over traditional machine learning methods in terms of accuracy and speed of prediction. In this project, we will fetch the historical data of stock automatically using python libraries and fit the LSTM model on this data to predict the future prices of the stock.

In this project, the future stock prices of State Bank of India (SBIN) are predicted using the LSTM Recurrent Neural Network. Our task is to predict stock prices for a few days, which is a time series problem. The LSTM model is very popular in time-series forecasting, and this is the reason why this model is chosen in this task. The historical prices of SBIN are collected automatically using the nsepy library of python. We have used 6 years of historical price data, from 01.01.2013 to 31.12.2018.

This data set contains 1483 observations with 12 attributes. After preprocessing, only dates and OHLC (Open, High, Low, Close) columns, a total of 5 columns, are taken as these columns have main significance in the dataset. The LSTM model is trained on this entire dataset, and for the testing purpose, a new dataset is fetched for the duration between 01.01.2019 to 18.09.2019. The stock prices for this new duration will be predicted by the already trained LSTM model, and the predicted prices will be plotted against the original prices to visualise the model's accuracy.

```
#Importing the libraries
from nsepy import get_history as gh
import datetime as dt
from matplotlib import pyplot as plt
from sklearn import model_selection
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

1.Step

Firstly, I imported the required libraries.TensorFlow will be used as a backend for LSTM model,
and nsepy will be used to fetch the historical stock data.

2.Step

We will fetch 6 years of historical prices of SBIN from 01.01.2013 to 31.12.2018. So we need to set the start and end dates and pass these parameters to the function for fetching the data.

```
#Setting start and end dates and fetching the historical data
start = dt.datetime(2013,1,1)
end = dt.datetime(2018,12,31)
stk_data = gh(symbol='SBIN',start=start,end=end)
```

```
#Visualizing the fetched data
plt.figure(figsize=(14,14))
plt.plot(stk_data['Close'])
plt.title('Historical Stock Value')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.show()
```

3.Step

We can visualise the fetched data in the above step. For simplicity, only the day-wise closing prices are visualised.

4.Step

There are 12 columns in the fetched data. Many of the columns are not of our interest so only significant columns are selected to create the main dataset.

```
#Data Preprocessing
stk_data['Date'] = stk_data.index
data2 = pd.DataFrame(columns = ['Date', 'Open',
                                'High', 'Low', 'Close'])
data2['Date'] = stk_data['Date']
data2['Open'] = stk_data['Open']
data2['High'] = stk_data['High']
data2['Low'] = stk_data['Low']
data2['Close'] = stk_data['Close']
```

```
train_set = data2.iloc[:, 1:2].values
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(train_set)
X_train = []
y_train = []
for i in range(60, 1482):
  X_train.append(training_set_scaled[i-60:i, 0])
  y_train.append(training_set_scaled[i, 0])
  X_train, y_train = np.array(X_train), np.array(y_train)
  X_train = np.reshape(X_train, (X_train.shape[0],
                                 X_train.shape[1], 1))
```

5.Step

We preprocess the data in order to prepare it for the LSTM model. The data fetched in step one is used for training purpose only. For testing purpose, different data will be fetched later.

6.Step

We defined the LSTM Recurrent Neural Network. Here, we can add more LSTM layers and adjust the dropout in order to improve the accuracy of the model.

```
#Defining the LSTM Recurrent Model
regressor = Sequential()
regressor.add(LSTM(units = 50, return_sequences = True,
                   input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))
regressor.add(Dense(units = 1))
```

```
#Compiling and fitting the model
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
regressor.fit(X_train, y_train, epochs = 15, batch_size = 32)
```

7.Step

We compiled and trained the model defined in the above step. Iteratively, it can be increased or decreased the epochs and batch size to get more accuracy.

8.Step

Now, our model is trained and needs to be tested on the testing data. For this purpose, fetch the new data for a different period. Preprocessing steps are similar as we have done with training data.

```
#Fetching the test data and preprocessing
testdataframe = gh(symbol='SBIN',start=dt.datetime(2019,1,1),
                   end=dt.datetime(2019,9,18))
testdataframe['Date'] = testdataframe.index
testdata = pd.DataFrame(columns = ['Date', 'Open', 'High',
                                   'Low', 'Close'])
testdata['Date'] = testdataframe['Date']
testdata['Open'] = testdataframe['Open']
testdata['High'] = testdataframe['High']
testdata['Low'] = testdataframe['Low']
testdata['Close'] = testdataframe['Close']
real_stock_price = testdata.iloc[:, 1:2].values
dataset_total = pd.concat((data2['Open'], testdata['Open']), axis = 0)
inputs = dataset_total[len(dataset_total) - len(testdata) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(60, 235):
  X_test.append(inputs[i-60:i, 0])
  X_test = np.array(X_test)
  X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
#Making predictions on the test data
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

9.Step

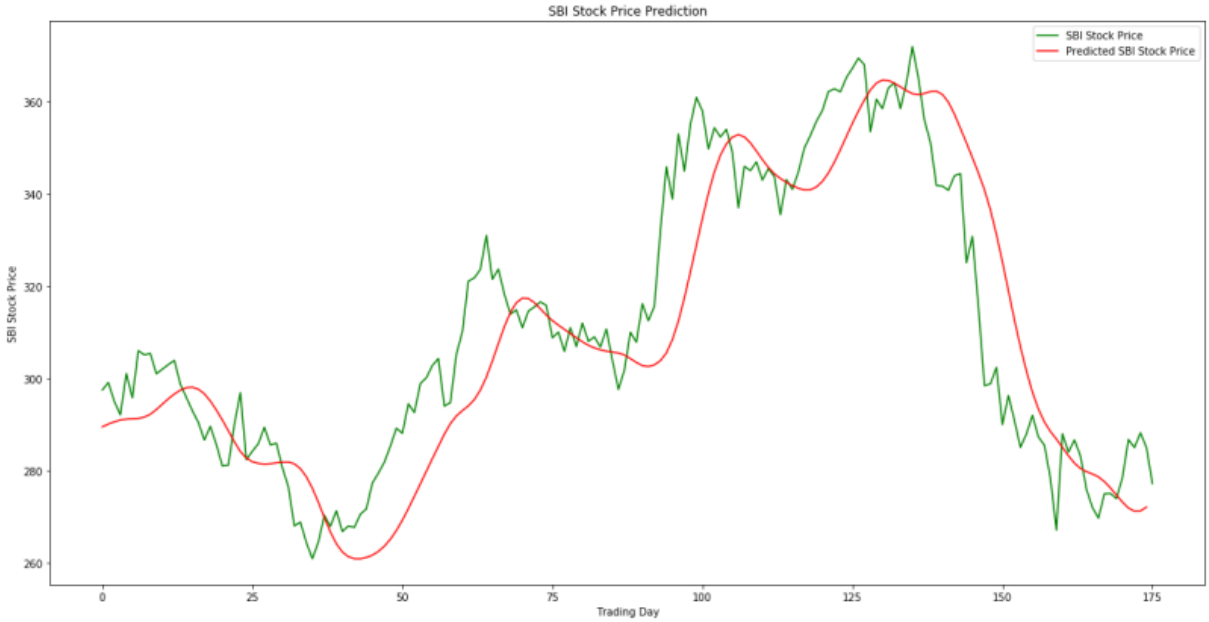
Test the LSTM model on the new dataset.

10.Step

Finally, we need to visualize the predicted stock prices with original stock prices.

```
#Visualizing the prediction
plt.figure(figsize=(20,10))
plt.plot(real_stock_price, color = 'green', label = 'SBI Stock Price')
plt.plot(predicted_stock_price, color = 'red', label = 'Predicted SBI Stock Price')
plt.title('SBI Stock Price Prediction')
plt.xlabel('Trading Day')
plt.ylabel('SBI Stock Price')
plt.legend()
plt.show()
```

Actually, the graph that should come out is as follows, but we couldn't run it due to some errors we got in the coding part.



We could not add the Tensorflow library in PyCharm. When we tried to install it with the help of Anaconda, we were able to install Tensorflow, but this time other libraries failed and we were faced with the problem of not being able to install them.

Conclusion

Today, the popularity of stock markets and cryptocurrencies is increasing day by day. In our research, we saw that the applications, projects and resources that could make price predictions using historical data thanks to RNN were quite sufficient, and we wanted to work on this subject in our term project.

I learned that RNN and LSTM are used in many indicators that help stock market and cryptocurrency analysis. As I was knowledgeable about these indicators, accessing the source codes and getting help from them increased my interest and knowledge in the project. Although we use different and even online compilers to solve the library problem we are experiencing, we do not get a positive result, but we think that we have increased our skills in the Neural Network course we took as part of the Artificial Intelligence and Machine Learning Minor Program.

REFERENCES

- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://towardsdatascience.com/stock-prediction-using-recurrent-neural-networks-c03637437578
- https://www.sciencedirect.com/science/article/pii/S1877050920304865
- https://mospace.umsystem.edu/xmlui/bitstream/handle/10355/56058/research.pdf?isAllowed=y&sequence=2
- https://www.youtube.com/watch?v=H6du_pfuznE
- https://github.com/krishnaik06/Stock-MArket-Forecasting/blob/master/Untitled.ipynb
- https://analyticsindiamag.com/how-to-code-your-first-lstm-network-in-keras/