Special topic report

Stock Price Prediction

Introduction

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present:

1. Input gate - Discover which value from input should be used to modify the memory. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1.
2. Forget gate - Discover what details to be discarded from the block. It is decided by the **sigmoid function.** it looks at the previous state(**ht-1**) and the content input (**Xt**) and outputs a number between **0(***omit this*) and **1(***keep this***)** for each number in the cell state **Ct–1**.
3. Output gate - The input and the memory of the block is used to decide the output. **Sigmoid** function decides which values to let through **0,1.** and **tanh** function gives weightage to the values which are passed deciding their level of importance ranging from**-1** to **1** and multiplied with output of **Sigmoid.**

As financial institutions begin to embrace artificial intelligence, machine learning is increasingly utilized to help make trading decisions. Although there is an abundance of stock data for machine learning models to train on, a high noise to signal ratio and the multitude of factors that affect stock prices are among the several reasons that predicting the market difficult. At the same time, these models don't

need to reach high levels of accuracy because even 60% accuracy can deliver solid returns. One method for predicting stock prices is using a long short-term memory neural network (LSTM) for times series forecasting.

The stock market is a vast array of investors and traders who buy and sell stock, pushing the price up or down. The prices of stocks are governed by the principles of demand and supply, and the ultimate goal of buying shares is to make money by buying stocks in companies whose perceived value (i.e., share price) is expected to rise. Stock markets are closely linked with the world of economics —the rise and fall of share prices can be traced back to some Key Performance Indicators (KPI's). The five most commonly used KPI's are the opening stock price (`Open'), end-of-day price (`Close'), intra-day low price(`Low'), intra-day peak price (`High'), and total volume of stocks traded during the day (`Volume'). Economics and stock prices are mainly reliant upon subjective perceptions about the stock market. It is near-impossible to predict stock prices to the T, owing to the volatility of factors that play a major role in the movement of prices. However, it is possible to make an educated estimate of prices. Stock prices never vary in isolation: the movement of one tends to have an avalanche effect on several other stocks as well. This aspect of stock price movement can be used as an important tool to predict the prices of many stocks at once. Due to the sheer volume of money involved and number of transactions that take place every minute, there comes a trade-off between the accuracy and the volume of predictions made; as such, most stock prediction systems are implemented in a distributed, parallelized fashion. These are some of the considerations and challenges faced in stock market analysis.

Literature survey

The initial focus of our literature survey was to explore generic online learning algorithms and see if they could be adapted to our use case i.e., working on real-time stock price data. These included Online AUC Maximization, Online Transfer Learning and Online Feature Selection. However, as we were unable to find any potential adaptation of these for stock price prediction, we then decided to look at the existing systems, analyse the major drawbacks of the same, and see if we could improve upon them. We zeroed in on the correlation between stock data (in the form of dynamic, long-term temporal dependencies between stock prices) as the key issue that we wished to solve. A brief search of generic solutions to the above problem led us to RNN's and LSTM. After deciding to use an LSTM neural network to perform stock prediction, we consulted a number of papers to study the concept of gradient descent and its various types. We concluded our literature survey by looking at how gradient descent can be used to tune the weights of an LSTM network and how this process can be optimized.

Existing systems and their drawbacks

Traditional approaches to stock market analysis and stock price prediction include fundamental analysis, which looks at a stock's past performance and the general credibility of the company itself, and statistical analysis, which is solely concerned with number crunching and identifying patterns in stock price variation. The latter is commonly achieved with the help of Genetic Algorithms (GA) or Artificial Neural Networks (ANN's), but these fail to capture correlation between stock prices in the form of long-term temporal dependencies. Another major issue with using simple ANNs for stock prediction is the phenomenon of exploding / vanishing gradient, where the weights of a large network either become too large or too small (respectively), drastically slowing their convergence to the optimal value. This is typically caused by two factors: weights are initialized randomly, and the weights closer to the end of the network also tend to change a lot more than those at the beginning. An alternative approach to stock market analysis is to reduce

the dimensionality of the input data and apply feature selection algorithms to shortlist a core set of features (such as GDP, oil price, inflation rate, etc.) that have the greatest impact on stock prices or currency exchange rates across markets. However, this method does not consider long-term trading strategies as it fails to take the entire history of trends into account; furthermore, there is no provision for outlier detection.

Advantages of LSTM

The main advantage of an LSTM is its ability to learn context-specific temporal dependence. Each LSTM unit remembers information for either a long or a short period of time (hence the name) without explicitly using an activation function within the recurrent components. An important fact to note is that any cell state is multiplied only by the output of the forget gate, which varies between 0 and 1. That is, the forget gate in an LSTM cell is responsible for both the weights and the activation function of the cell state. Therefore, information from a previous cell state can pass through a cell unchanged instead of increasing or decreasing exponentially at each time-step or layer, and the weights can converge to their optimal values in areas on able amount of time. This allows LSTM's to solve the vanishing gradient problem – since the value stored in a memory cell isn't iteratively modified, the gradient does not vanish when trained with backpropagation. Additionally, LSTM's are also relatively insensitive to gaps (i.e., time lags between input data points) compared to other RNN's.

Explanation of the code

i. Imports: We import NumPy for making scientific computations, Pandas for loading and modifying datasets and Matplotlib for plotting graphs.

```
[ ]  1 import numpy as np
     2 import matplotlib.pyplot as plt
     3 import pandas as pd
```

ii. Loading the data: We load the stock data into the program. From the data we select the values of the first and second columns as out training dataset.

```
[ ]  1 url = r'/content/TSLA train.csv'
     2 dataset_train = pd.read_csv(url)
     3 training_set = dataset_train.iloc[:, 1:2].values
```

iii. Checking the data: We check the 'head' of the data which shows us the first five rows of out uploaded data.

```
[ ]  1 dataset_train.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	01-04-2015	188.699997	192.300003	186.050003	187.589996	187.589996	3794600
1	02-04-2015	190.229996	193.229996	190.000000	191.000000	191.000000	5010400
2	06-04-2015	198.000000	207.750000	197.500000	203.100006	203.100006	12455800
3	07-04-2015	202.509995	205.059998	201.139999	203.250000	203.250000	4347900
4	08-04-2015	208.199997	210.899994	205.869995	207.669998	207.669998	6303100

iv. Data Normalization: We normalize the data, i.e., change the values of the numeric columns in the dataset to a common scale. This improves the performance of our model. To scale the data, we use MinMaxScaler with numbers between 0 & 1.

```
[ ]  1 from sklearn.preprocessing import MinMaxScaler
     2 sc = MinMaxScaler(feature_range=(0,1))
     3 training_set_scaled = sc.fit_transform(training_set)
```

v. Timesteps: We input our data in the form of a 3D array to the model. We create data in 60 timesteps and then use NumPy to convert it into an array.

```
X_train = []
y_train = []
for i in range(60, 896):
  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))
```

vi. More imports: We import Sequential to initialize the neural network, LSTM to add the LSTM layers, Dropout to prevent overfitting and Dense to add a densely connected neural network layer.

```
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import LSTM
from tensorflow.python.keras.layers import Dropout
from tensorflow.python.keras.layers import Dense
```

vii. The Model: 50 units is the dimensionality of the output space. return_sequences = True is necessary for stacking LSTM layers so the consequent LSTM layer has a three-dimensional sequence input. input_shape is the shape of the training dataset. Specifying 0.2 in the Dropout layer means that 20% of the layers will be dropped. We add the Dense layer that specifies an output of one unit. To compile, we use the Adam optimizer and set the loss as the mean_squared_error. After that, we fit the model to run for 50 epochs.

```
model = Sequential()

model.add(LSTM(units=50,return_sequences=True,
               input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))

model.add(LSTM(units=50,return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=50,return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(units=1))

model.compile(optimizer='adam',loss='mean_squared_error')
model.fit(X_train,y_train,epochs=50,batch_size=100)
```

viii. Making predictions: To make predictions, we import the test dataset. In this program, test set is 30% of the total dataset. The remaining 70% was used to train the model.

```
url = r'/content/TSLA test.csv'
dataset_test = pd.read_csv(url)
real_stock_price = dataset_test.iloc[:, 1:2].values

url = r'/content/TSLA.csv'
dataset_total = pd.read_csv(url)
```

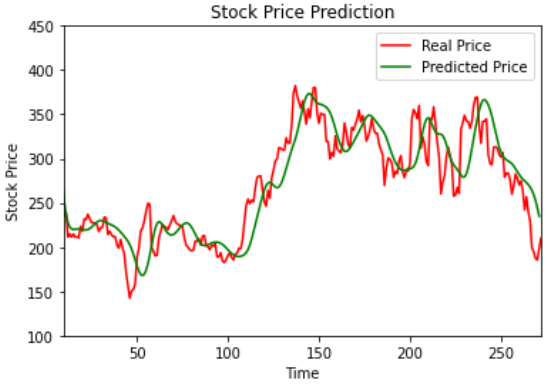
ix. Modifying the test set: We modify the test dataset. To do this we merge the training set and the test set on 0 axis, set 60 as the timestep again, use MinMaxScaler to reshape the data and then use inverse_transform to put the stock prices in a readable format.

```
dataset_total = pd.concat((dataset_train['Open'],
                           dataset_test['Open']), axis = 0)

inputs = dataset_total[len(dataset_total) -
                       len(dataset_test) - 60:].values

inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []

for i in range(60, 332):
  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))
predicted_stock_price = model.predict(X_test)

predicted_stock_price =
sc.inverse_transform(predicted_stock_price)
```

Result

We use Matplotlib to visualize the result of our predicted stock price and the actual stock price.

```
plt.matplotlib.pyplot.xlim(10,272)
plt.matplotlib.pyplot.ylim(100,450)
plt.plot(real_stock_price,
         color = 'red', label = 'Real Price')
plt.plot(predicted_stock_price, color =
         'green', label = 'Predicted Price')
plt.title('Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```



Terminologies used

1. Training set: subsection of the original data that is used to train the neural network model for predicting the output values.
2. Test set: part of the original data that is used to make predictions of the output value, which are then compared with the actual values to evaluate the performance of the model.
3. Validation set: portion of the original data that is used to tune the parameters of the neural network model.
4. Activation function: in a neural network, the activation function of a node defines the output of that node as a weighted sum of inputs. Here, the sigmoid and ReLU (Rectified Linear Unit) activation functions were tested to optimize the prediction model.
5. Batch size: number of samples that must be processed by the model before updating the weights of the parameters.
6. Epoch: a complete pass through the given dataset by the training algorithm.
7. Dropout: a technique where randomly selected neurons are ignored during training i.e., they are "dropped out" randomly. Thus, their contribution to the activation of downstream neurons is temporally removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass.
8. Loss function: a function, defined on a data point, prediction and label, that measures a penalty such as square loss.
9. Cost function: a sum of loss functions over the training set. An example is the Mean Squared Error (MSE).
10. Root Mean Square Error (RMSE): measure of the difference between values predicted by a model and the values actually observed. It is calculated by taking the summation of the squares of the differences between the predicted value and actual value, and dividing it by the number of samples.
    In general, smaller the RMSE value, greater the accuracy of the predictions made.

Outcomes

1) Learned about neural networks and LSTM.
2) Studied various python libraries and implemented them.
3) Learned to use platforms like Anaconda, Jupyter notebook and Google Colaboratory.
4) Learned about the stock market.
5) Developed an interest in the field of Data Analytics.

Acknowledgement

We would like to express our gratitude towards Dr. Venugopal N for this wonderful opportunity and guidance to work on this special topic project on the afore-mentioned project. I would also like to thank our HOD for giving us the opportunity and a platform to work on this project. At last, I would also like to thank my teammates for their support and continuous encouragement.

Bibliography

- https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e
- Derrick Mwiti's GitHub profile
- Stack Overflow
- Towards Data Science magazine
- Python.org
- Yahoo finance
- Tableau software
- Google Colaboratory
- Tensorflow.org
- https://www.youtube.com/channel/UCiT9RITQ9PW6BhXK0y2jaeg/featured
- Nazar, Nasrin Banu, and Radha Senthil Kumar. "An online approach for feature selection for classification in big data." Turkish Journal of Electrical Engineering & Computer Sciences 25.1 (2017): 163-171.
- Zhu, Maohua, et al. "Training Long Short-Term Memory With Sparsified Stochastic Gradient Descent."(2016)
- https://www.researchgate.net/publication/327967988_Predicting_Stock_Prices_Using_LSTM