Stock prediction & Machine Learning

Project's report

Course of Deep Learning

Authors:
Antonio Antona
Barillon Yanis
Vandenbussche David

Teacher: *Anastase - Alexandre Charantonis*
Department: Applied mathematics
Date: Novermber 15,2022

Contents

List of Figures

1 Introduction

The attempt is to forecast or predict the upcoming value of one stock, activity known as Stock Market Prediction. It is an area that has driven the focus of many individuals including not only companies, but also traders, market participants, data analysts, and even computer engineers working in the domain of Machine Learning (ML) and Artificial Intelligence (AI).
Investing funds in the market is subjected to various market risks, as the value of the shares of the company is highly dependent upon the profits and performance of the organization in the marketplace and can thus vary due to various factors such as government policies, microeconomic indicators, demand and supply etc. These variations in the market are studied to develop software and programs using various techniques such as ML, Deep Learning, Neural Networks, AI. Such systems and software can enable the investor to properly anticipate the situation of the company, on the basis of past and present data, the current condition in the market, etc. and give them a direction to make decisions so that they don't lose their valuable money and earn maximum profits. Machine learning can be defined as the data which is obtained by knowledge extraction. Machines dont have to be programmed explicitly instead they are trained to make decisions that are driven by data. Instead of writing a code for every specific problem, data is provided to the generic algorithms and logic is developed on the basis of that data. When a machine improves its performance based on its past experiences it can be said that machine has truly learnt. The technique for most accurate prediction is by learning from past instances, and to make a program to do this is best possible with machine learning techniques.

1.1 Methods

The numerous methods applied for achieving share price prediction are broadly divided into four categories:

- Traditional Machine Learning Methods - Includes traditional methods such as linear regression analysis and logistic regression analysis.
- Deep Learning and Neural Networks - Many of these techniques make use of RNNs and LSTMs which are a special type of RNN.
- Time Series Analysis Methods - This method depends on forecasts and projection of discretetime data.
- Graph-Based Approaches - Often concerned with comparing the stock market as a network of interconnected nodes where a change in one component will impact the prices of other components.

One of the approaches for predicting stock prices is the big data approach that aims to derive insights from a large amount of data that is publicly available and this data is analyzed on platforms such as Hadoop. The base concept of the deep learning approach is to make calculations based on neural networks. Long Short-Term Memory (LSTM) is a special type of Recurrent Neural Network (RNN) that is used to overcome the problem of long-term dependencies.
Another way to forecast equity prices is by analyzing the sentiments on social media data or news stories that help in determining the general trend that a particular company's or industries' shares may take based on a collective opinion. The value of a stock is often seen as a time series model and therefore time series analysis is also one popular model for forecasting stock prices.

1.2 Common problems

Here we are going to show some of the most commons problems when we try a Stock price prediction:

- **Selection Bias:**
  Many projects start with the arbitrary selection of a stock with which an algorithm is to be applied to, this stock is often a tech stock such as Apple or Amazon, the simple reason being these companies are well known and ingrained in the everyday lives of consumers. This is problematic as stock selection is not an arbitrary process, it is part of the investment decision making process that requires a model in itself.
- **Portfolio Construction:**
  Controlling risk is as important to a robust investment strategy as generating returns. If stock selection is the first part of the investment process, then portfolio construction is the vital next step.

- **Incorrect Application of Pre-processing:**
  Standard rinse, wash and repeat data pre-processing techniques like standardization cannot be directly applied to stock prices. Distribution of stock prices change from year to year, meaning that the mean and standard deviations will also change. This property of financial time series is called non-stationarity and it remains an open problem in financial prediction.

1.3 Conclusion

The data in this paper consist of the daily opening prices of two stocks in the New York Stock Exchange NYSE from our dataset, we choose a period from 01/01/2005 to 01/01/2015 .
To build our model we are going to use the LSTM method, our model uses 80% of data for training and the other 20% of data for testing. For training we use mean squared error to optimize our model.
As the stock market is bound tightly to a country's economic growth and brings in huge investments by the investors and issues equities in the public interest, forecasting the movement of the stock prices and the market becomes essential in order to prevent huge losses and make relevant decisions but we don't have to forget that predicting the market is challenging because the future has a lot of uncertainty for the lot of external parameters we have.

2 Architecture:

We will try the LSTM architecture. Long Short-Term Memory is an advanced version of recurrent neural network (RNN) architecture that was designed to model chronological sequences and their long-range dependencies more precisely than conventional RNNs.
LSTMs deal with both Long Term Memory (LTM) and Short Term Memory (STM) and for making the calculations simple and effective it uses the concept of gates.

- **Forget Gate:**LTM goes to forget gate and it forgets information that is not useful.
- **Learn Gate:**Event (current input) and STM are combined together so that necessary information that we have recently learned from STM can be applied to the current input.
- **Remember Gate:**LTM information that we haven't forget and STM and Event are combined together in Remember gate which works as updated LTM.
- **Use Gate:**This gate also uses LTM, STM, and Event to predict the output of the current event which works as an updated STM.

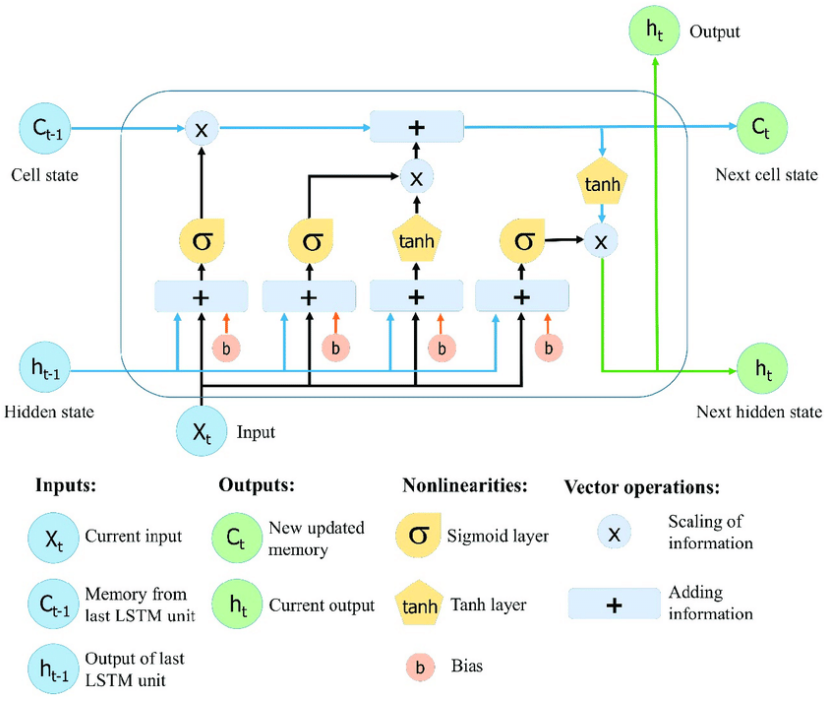


Figure 1: LSTM Architecture

2.1 Usage of LSTM

Training LSTMs removes the problem of Vanishing Gradient (weights become too small that under-fits the model), but it still faces the issue of Exploding Gradient (weights become too large that over-fits the model). Training of LSTMs can be easily done using Python frameworks like Tensorflow, Pytorch, Theano, etc. and the catch is the same as RNN, we would need GPU for training deeper LSTM Networks.
Since LSTMs take care of the long term dependencies its widely used in tasks like Language Generation, Voice Recognition, Image OCR Models, etc. Also, this technique is getting noticed in Object Detection also (mainly scene text detection).

3 Anticipated issues:

Data leakage is an important issue in our study because we are using time series, and if the train test validation data splitting is not well structured it will wrongly influence the results. Therefore to prevent data leakage we are going to separate the three train test validation sets by a time period of three months. We are going only to scale the data between 0 and 1 because normalizing is not very appropriate when dealing with prices.
We can define data leakage as: "When data set contains relevant data, but similar data is not obtainable when the models are used for predictions, data leakage (or leaking) occurs. This results in great success on the training dataset (and possibly even the validation accuracy), but lack of performance in production."
Data leakage, or merely leaking, is a term used during machine learning to describe the situation in which the data used to teach a machine-learning algorithm contains unexpected extra information about the subject you're estimating. Leakage occurs when information about the target label or number is introduced during learning that would not be lawfully accessible during actual use. The most fundamental example of data leakage would be if the true label of a dataset was included as a characteristic in the model. If this object is classified as an apple, the algorithm would learn to predict that it is an apple.

3.1 Why does data leakage happen?

Data leakage can occur for a variety of reasons, often in subtle and difficult-to-detect ways. When data leakage occurs, it usually leads to overly optimistic outcomes during the model building phase, followed by the unpleasant surprise of poor results after the prediction model is implemented and tested on new data. In other words, the leakage might lead your system to train a suboptimal model that performs significantly worse in practice than a model generated in a leak-free environment. Leakage can have a variety of real effects, ranging from the financial expense of making a terrible financial and technological expenditure something that doesn't work to system failures that hurt consumers' perceptions of your system's reliability or affect the employer products. As a result, data leaking is among the most significant and pervasive challenges in machine learning and statistical, and one that deep learning practitioners must be mindful of.

4 Evaluating Prediction Performance for Stock Price Prediction:

Before putting the algorithms into practice, let's clarify the metric to measure the performance of our models. **Stock price prediction** being a fundamental regression problem, we can use RMSE (Root Mean Squared Error) or MAPE (Mean Absolute Percentage Error) to measure how close or far off our price predictions are from the real world.
We are going to use the $RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$ which is most useful when large errors are particularly undesirable.It measures the average magnitude of the errors. RMSE value with zero indicates that the model has a perfect fit. The lower the RMSE, the better the model and its predictions.
Looking closely at the formula of RMSE, we can see how we will be able to consider the difference (or error) between the actual (y_i) i-th measurement and predicted $(\hat{y}_i)$ i-th measurement price values for all N number of data points and get an absolute measure of error.
What is a good RMSE value? The short answer: It depends. The lower the RMSE, the better a given model is able to "fit" a dataset. However, the range of the dataset you're working with is important in determining whether or not a given RMSE value is "low" or not.
One way to gain a better understanding of whether a certain RMSE value is "good" is to normalize it using the following formula: $NormalizedRMSE = RMSE/(maxvalue–minvalue)$ This produces a value between 0 and 1, where values closer to 0 represent better fitting models.

The **mean absolute percentage error (MAPE)**, also known as **mean absolute percentage deviation (MAPD)** , is a measure of *prediction accuracy* of a forecasting method in statistics. It usually expresses the accuracy as a ratio defined by the formula:

$$\text{MAPE} = \frac{1}{n} 100 \sum_{i=1}^{n} \left| \frac{(y_i - \hat{y}_i)}{\hat{y}_i} \right| \tag{1}$$

On the other hand, MAPE looks at the error concerning the true value – it will measure relatively how far off the predicted values are from the truth instead of considering the actual difference. This is a good measure to keep the error ranges in check if we deal with too large or small values. For instance, RMSE for values in the range of 10e6 might blow out of proportion, whereas MAPE will keep error in a fixed range.

4.1 Nash-Sutcliffe efficiencies

$$NSE = 1 - \frac{\sum_{t=1}^{T} \left(Q_o^t - Q_m^t\right)^2}{\sum_{t=1}^{T} \left(Q_o^t - \overline{Q}_o\right)^2}$$

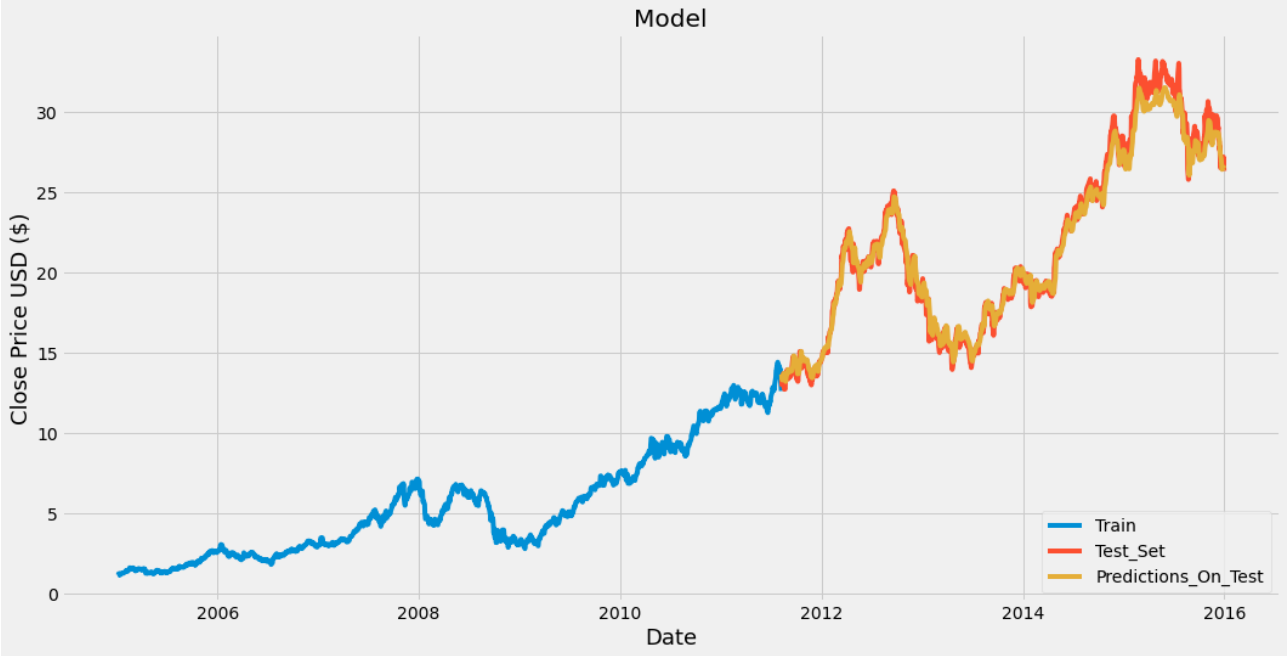
We will also try to implement the *Nash index* which is specifically adapted for time series, **The Nash-Sutcliffe efficiency (NSE)** is a normalized statistic that determines the relative magnitude of the residual variance (”noise”) compared to the measured data variance (”information”) (Nash and Sutcliffe, 1970).
NSE indicates how well the plot of observed versus simulated data fits the 1:1 line. Nash-Sutcliffe efficiencies range from -Inf to 1. Essentially, the closer to 1, the more accurate the model is. -) NSE = 1, corresponds to a perfect match of modelled to the observed data. -) NSE = 0, indicates that the model predictions are as accurate as the mean of the observed data, -) -Inf ¡ NSE ¡ 0, indicates that the observed mean is better predictor than the model.

5 Results phase of the project:

As we choose a LSTM architecture to avoid overfitting, we now need to find the hyper parameters that best fit the model. To begin with LSTM is already a complex architecture so we will want to avoid adding more complexity to the model by adding too much layers. Plus the time of computation seems to explode with the optimisation of the number of layers.
As the time of computation was already huge we want to focus on the optimisation of the most important hyperparameters which are the units in each layer, the activation function and the learning rate for the optimizer. We will later focus on what time period would be optimal as it seems really important to us. We use the Bayesian Optimisation supermodel in order to determine the most relevant hyperparameters. Our number of trials for the determination of hyperparameters is 10 as we have 10*10*3 combinations for our hyperparameters' model.

Our result for the hyperparameter optimisation gives us this model : The optimal number of neurons in the first LSTM layer is 35.
The optimal number of units for the second LSTM layer is 125.
The optimal activation function for the 3rd and 4th densely-connected layers is tanh.

So we have a first LSTM layer with 35 neurons then another LSTM layer with 125 neurons, followed by a Dense layer of 31 neurons and another one of 15 neurons. The output layer is a Dense with 1 neuron and a linear activation as we want a regression. The mean squared error on the training set is 7 e-05 meanwhile the mean squared error on the testing set is 480.



As we see have we a big problem with overfitting even though the algorithm seems to do what we want it to do precisely as we can see in the graph.
For now we've only focus on the mean squared error loss function so we need also too use another measure to ensure that our model is robust.
We also need to separate the train test validation in order to avoid data leakage.

6 How to find best hyperparameters:

Before we get into the tuning of the most relevant hyperparameters for LSTM, it is worth noting that there are ways to let your system find the hyperparameters for you by using optimizations tools. These methods are useful to bypass more manual processes in identifying good hyperparameters and tuning them. In Python, some such tools are:

1. Keras Tuner: Keras Tuner is an easy-to-use, distributable hyperparameter optimization framework that solves the pain points of performing a hyperparameter search.

2. Bayesian Optimization : This technique is particularly suited for optimization of high cost functions, situations where the balance between exploration and exploitation is important.

3. Grid search : implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used.

   The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

It should be kept in mind that many such hyperparameters are volatile, in the sense that different values (or even same values and different runs) may yield different results. So make sure you always compare models and performance by tweaking these hyperparameters to get the optimum results.

7 Relevant Hyperparameters to tune:

An LSTM (Long short-term memory) model is an artificial recurrent neural network (RNN) architecture which has feedback connections, making it able to not only process single data points, but also entire sequences of data. Below we will show all such hypermeters for an LSTM model necessary to improve the performance and what values are used as best practice.

- *NUMBER OF NODES AND HIDDEN LAYERS*
  The layers between the input and output layers are called hidden layers. This fundamental concept is what makes deep learning networks being termed as a "black box", often being criticized for not being transparent and their predictions not being traceable by humans. There is no final number on how many nodes (hidden neurons) or hidden layers one should use, so depending on the individual problem (believe it or not) a trial and error approach will give the best results.

- *NUMBER OF UNITS IN A DENSE LAYER* - Method: model.add(Dense(10, . . .))
  A dense layer is the most frequently used layer which is basically a layer where each neuron receives input from all neurons in the previous layer — thus, "densely connected". Dense layers improve overall accuracy and 5–10 units or nodes per layer is a good base. So the output shape of the final dense layer will be affected by the number of neuron / units specified.

- *DROPOUT* - Method: model.add(LSTM(. . . , dropout=0.5))
  Every LSTM layer should be accompanied by a dropout layer. Such a layer helps avoid overfitting in training by bypassing randomly selected neurons, thereby reducing the sensitivity to specific weights of the individual neurons. While dropout layers can be used with input layers, they shouldn't be used with output layers as that may mess up the output from the model and the calculation of error. While adding more complexity may risk overfitting (by increasing nodes in dense layers or adding more number of dense layers and have poor validation accuracy), this can be addressed by adding dropout.

  A good starting point is 20% but the dropout value should be kept small (up to 50%). The 20% value is widely accepted as the best compromise between preventing model overfitting and retaining model accuracy.

- *WEIGHT INITIALIZATION*
  Ideally, it is better to employ different weight initialization schemes according to what activation function is used. However, more commonly a uniform distribution is used while choose initial weight values. It is not possible to set all weights to 0.0 as the asymmetry in the error gradient is brought out by the optimization algorithm; to begin searching effectively. Different set of weights results in different starting points of the optimization process, potentially leading to different final sets with different performance characteristics. Weights should finally be initialized randomly to small numbers (an expectation of the stochastic optimization algorithm, otherwise known as stochastic gradient descent) to harness randomness in the search process.

- *DECAY RATE*
  The weight decay can be added in the weight update rule that makes the weights decay to zero exponentially, if no other weight update is scheduled. After each update, the weights are multiplied by a factor slightly less than 1, thereby preventing them from growing to huge. This specifies regularization in the network.
  The default value of 0.97 should be enough to start off.

- *ACTIVATION FUNCTION*
  Activation functions are what defines the output of a node as either being ON or OFF. These functions are used to introduce non-linearity to models, allowing deep learning models to learn non-linear prediction boundaries. Technically, activation functions can be included in the dense layers but splitting them into them into different layers makes it possible to retrieve the reduced output of the density layer.

- *LEARNING RATE* This hyperparameter defines how quickly the network updates its parameters. Setting a higher learning rate accelerates the learning but the model may not converge (a state during training where the loss settles to within an error range around the final value), or even diverge. Conversely, a lower rate will slow down the learning drastically as steps towards the minimum of loss function will be tiny, but will allow the model to converge smoothly.

- *MOMENTUM*
  The momentum hyperparameter has been researched into to integrate with RNN and LSTM. Momentum is a unique hyperparameter which allows the accumulation of the gradients of the past steps to determine the direction to go with, instead of using the gradient of only the current step to guide the search. Typically, the value is between 0.5 to 0.9.
- *NUMBER OF EPOCHS*
  This hyperparameters sets how many complete iterations of the dataset is to be run. While theoretically, this number can be set to an integer value between one and infinity, this should be increased until the validation accuracy starts to decrease even though training accuracy increases (and hence risking overfitting).
- *BATCH SIZE*
  This hyperparameter defines the number of samples to work on before the internal parameters of the model are updated. Large sizes make large gradient steps compared to smaller ones for the same number of samples "seen". Widely accepted, a good default value for batch size is 32. For experimentation, you can try multiples of 32, such as 64, 128 and 256.

References

[1] Payal Soni et al 2022 J. Phys.: Conf. Ser. 2161 012065.

[2] Singh, Nirbhey Khalfay, Neeha Soni, Vidhi Vora, Deepali. (2017). Stock Prediction using Machine Learning a Review Paper. International Journal of Computer Applications. 163. 36-43. 10.5120/ijca2017913453.