Advanced Data Analysis Project

Google Stock Dataset

Week 4

LUT University

Alejandro Lopez Vargas
Sina Saeedi
Pau Ventura Rodríguez

Visit our GitHub to gain access to the script.

Introduction

In this project, we aim to analyze the historical data of Google's (GOOGL) stock price and create a predictive model for it. The dataset includes the following variables: *Date*, *Open*, *High*, *Low*, *Close*, *Volume*, and *Name*. The *'Name'* variable is constant and represents the company's name, which is Google in this case, making it unnecessary for our analysis.

Here are brief explanations about each of the variables:

1. **Date**: This variable represents the specific dates corresponding to each data point in the dataset. It serves as the timeline for the stock's historical performance.

2. **Open:** The opening price of the Google stock on a particular date. This is the price at which the stock begins trading at the beginning of the trading day.

3. **High:** The highest price at which Google stock traded during a given day. It provides insights into the highest level the stock reached within a trading session.

4. **Low:** The lowest price at which Google stock traded during a given day. It indicates the lowest level the stock reached within a trading session.

5. **Close:** The closing price of Google stock on a specific date. It is the final price at which the stock is traded for the day, representing the market sentiment at the close of the trading session.

6. **Volume:** The total number of shares of Google stock traded on a given day. Volume is a crucial indicator of market activity, providing insights into the level of interest and participation in the stock.

Data Preprocessing:

1. **Data Formatting:** Ensure that the date format is the correct one and build an appropriate dataframe indexed by date. Ensure as well that data has the same frequency between observations (in our case we need to have all business days). We have converted the Date column on the dataframe to format Datetime in order to give it an ordinal value. The stock market is only open on business days, which means we do not have data for all days of the year. Furthermore, some holidays did also not register data like Christmas, which we will create in order to have an equally spaced dataframe. Pandas allows us to create these extra samples by using the command: *df.asfreq('b')*, which will fill their values with NaN.

2. **Data Cleaning:** Dropping unnecessary columns. In our case the column “Name” only has one unique value, which does not give us any information on the target and will then be dropped. Check for missing values in the dataset and handle them appropriately. We have missing values as we created them when we added the extra days to keep the frequency. Unfortunately, we have 3 samples that have consecutive missing values. As all those samples consist of only clusters of 2 consecutive samples, we will use Frontfilling on the first instance and Backfilling on the last instance. To fill up NaN values, we can use Backfilling (setting up the values with the next datapoint’s values), Frontfilling (setting up the values with the previous datapoint’s values),

Meanfilling (assigning the mean value of that feature, which in this case will not yield good results due to the increasing tendency). In this case, we will use Backfilling. In addition, check for outliers in the data and consider whether to remove or transform them.

3. **Feature Engineering:** Create lag features for Open, High, Low, Close, and Volume. However, if we want to predict the Closing price, we will not have the High, Low, or Volume variables. However, what we will have is the values for those variables on the previous timestamp, so we can use on each sample the lagged version of Low, High, and Volume instead of the original one. This means including previous time steps as features. The number of lags is a hyperparameter that can be tuned.

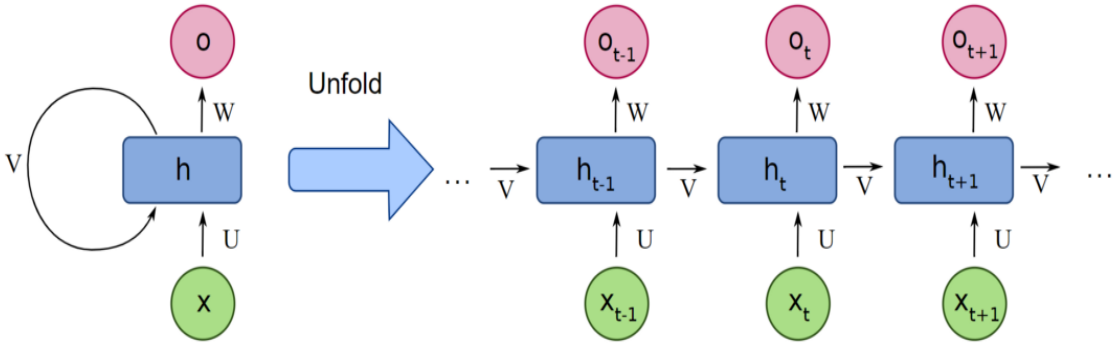
4. **Train, Test, and Validation Split:** Divide the dataset into training, testing, and validation sets, ensuring that the test and validation data capture the trend and seasonality patterns present in the training data. Employ a split ratio of (80-10-10) for the dataset allocation. The training split will be used to train our model. The validation will be used to know when to stop training the model and avoiding overfitting. Finally, the test split will be used to measure the performance of the model on unseen data. We can also perform cross validation to ensure that our data partitions are not biased and our performance results are as rigorous as possible. The type of cross validation used in that case would be Time Series Cross-Validation, which divides the training set into two folds at each iteration on condition that the validation set is always ahead of the training set

5. **Normalization:** It's common to normalize the data, especially when using neural networks, to improve convergence and training speed. For our analysis, we will use the Z-score normalization method on train data and with the mean and standard deviation that we found from the train data, we will normalize the test and validation data.

Model Architecture:

The following graphs show the architectures for each neural network. However, we might include extra layers in the beginning or at the end of the network if we find it necessary when we do the implementation.

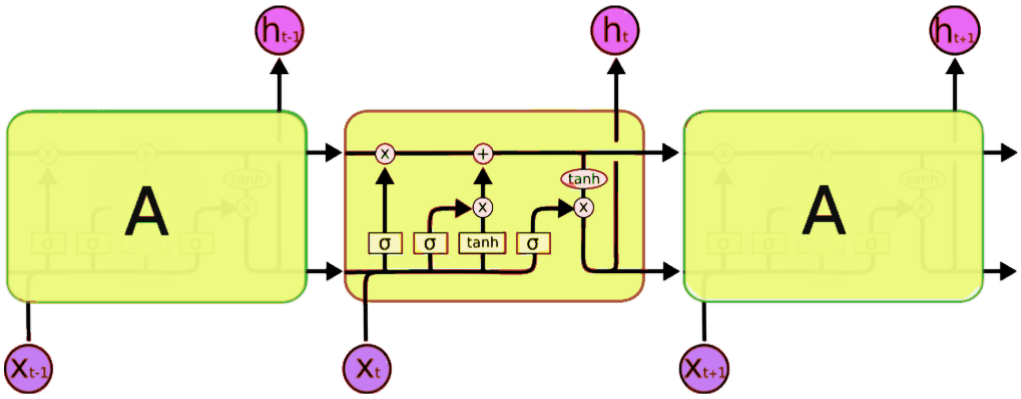
1. Simple RNN:
    - Recurrent Neural Networks (RNNs) are a type of artificial neural network designed to recognize patterns in sequences of data, such as time series. This is in contrast to regular neural networks, which process inputs independently of each other. The information cycles through a loop of neurons, allowing information to persist.
    - Employ a single-layer RNN with a suitable activation function. In our case, we will not use an activation function for the outputs as we have stock market prices.
    - Set the number of units based on the complexity of the data. We will tune this step while modeling the hyperparameters.
    - Use the previous Close price and other previous variables as input features.
    - This model has some big flaws: it is not able to keep the past information for too long and the gradient vanishes too easily, so LSTM might yield better results.



RNN architecture typically includes a single layer with a self-loop. This loop represents the temporal aspect of the network, where at each time step, the layer not only receives an input from the previous layer but also receives its

own output from the previous time step as input. This recurrent connection effectively gives the network a form of memory, allowing it to retain information between processing steps.

2. LSTM:
    - Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) that are designed to avoid the long-term dependency problem. They can remember information for long periods of time, which is particularly useful when dealing with sequences of data where context from earlier points is important for understanding later points.
    - Implement a stacked LSTM architecture with multiple layers and a suitable number of units per layer which will be tuned using grid search.
    - Employ a dropout layer between LSTM layers to prevent overfitting.
    - Utilize the previous Close price and other previous variables as input features.



The architecture of an LSTM has two main components:

Cell: Each unit of the LSTM network is known as a cell. Each cell has three inputs: the current input data, the previous cell state, and the previous hidden state.

Gates: LSTMs utilize a gating mechanism to manage the memory process. These gates regulate the flow of information into and out of the LSTM cells.

They store memory components in an analog format, and convert them to a probabilistic score by performing point-wise multiplication using the sigmoid activation function, which results in a range of 0-1.

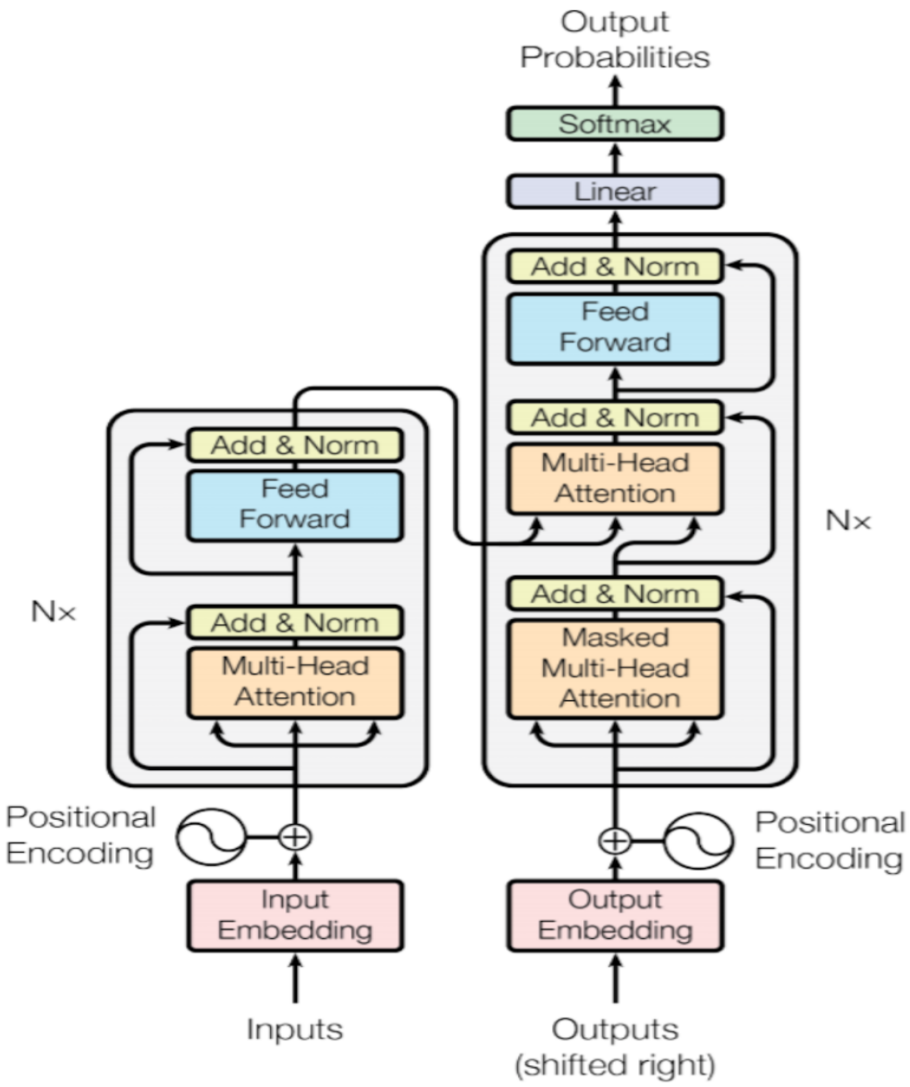
There are three types of gates within each LSTM cell:

Input Gate: This gate allows optional information necessary from the current cell state. It decides which information is relevant for the current input and allows it in.

Output Gate: This gate updates and finalizes the next hidden state. Since the hidden state contains critical information about previous cell inputs, it decides for the last time which information it should carry for providing the output.

Forget Gate: This gate is capable of eliminating unnecessary information. It assigns a 0 to tokens that are not important or relevant, effectively causing them to be forgotten 5.

3. Transformer:
    - Transformer is a type of neural network architecture designed for processing sequential data and is particularly effective for Natural Language Processing (NLP) tasks but it has also shown to be useful in some time series datasets.
    - Construct a Transformer encoder-decoder model, where the encoder processes the time-series data and the decoder predicts the future Close prices.
    - Utilize self-attention mechanisms within the encoder and decoder for efficient feature extraction and relationships.
    - Employ the previous Close price and other previous variables as input features to the encoder.



The main components of the Transformer are:

Input Embeddings: The input sentences are converted into numerical representations called embeddings.

Positional Encoding: Positional encoding is added to the input embeddings to maintain the order of the samples in the sequence.

Encoder: The encoder consists of several identical layers, each of which has two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network.

Decoder: Similar to the encoder, the decoder also has several identical layers. However, in addition to the two sub-layers present in the encoder, the decoder has a third sub-layer that performs multi-head attention over the output of the encoder stack.

Output Embeddings and Softmax: The decoder's output is transformed into the final output sequence through an output embedding and a softmax operation.

Evaluating Strategy:

For evaluation, several steps can be done which are explained briefly:

1. Performance Metrics

In the following equations, Y is the real value and \hat{Y} is the predicted value while "n" is the number of total samples.

- Mean Squared Error (MSE): measures the average squared difference between the predicted values and the actual values giving us the overall accuracy of the model.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- Mean Absolute Error (MAE): the average absolute difference between predicted and actual values. It is less sensitive to outliers.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

- Mean Absolute Percentage Error (MAPE): Calculates the percentage difference between predicted and actual values.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|A_i - F_i|}{A_i}$$

A_i is the actual value

F_i is the forecast value

n is total number of observations

2. Visual Inspection

- Time Series Plot: Plot the actual vs. predicted values over time to visually inspect the model's performance.
- Residual Plot: Plot the residuals (the differences between predicted and actual values) to check for patterns or systematic errors and look for white noise on residuals.

3. Hyperparameter Tuning

- Assess the impact of changes in hyperparameters on model performance.
- Conduct sensitivity analysis to identify the most influential hyperparameters.

4. Comparing the Models (RNN, LSTM, and Transformer) with Each Other

- Evaluating the performance metrics of all models in comparison with each other to determine the most suitable model for predicting our stock time series.

Strategy for Optimizing Model (Ablation/Sensitivity Study):

1. **Define Goals:**

· Clearly define the goals of the optimization study. Identify specific aspects of the model or hyperparameters to be studied. One optimization goal could be achieving a model with a lower RMSE by changing the hyperparameters. Another could be reducing the amount of time that the training phase will take.

2. **Parameter Selection:**

· Choose specific parameters or components of the model to study. This will include the number of layers, learning rates, activation functions, batch size, number of epochs, cell size, number of attention heads, ...

3. **Design Experiments:**

· Set up a series of experiments, systematically varying the chosen parameters. Keep other aspects constant to isolate the impact of the changes. For that purpose, it is also possible to use MATLAB's Experiment Manager tool. This can help tune the hyperparameters and also do a sensitivity analysis for some specific hyperparameters. Otherwise we will use Random Grid Search, which involves using random combinations of the hyperparameters to avoid going through all the hyperparameter search space, and keeping the best combination.

4. **Iterative Refinement:**

· Iterate the model based on the findings. Refine the model architecture or hyperparameters and repeat the experiments if necessary.

Strategy for work division

Since most of the materials and models are new to all of us, especially the Transformer NN, we will strive to work on each model together and collaborate. However, as some divisions might enhance the overall performance of our project, each person will be responsible for creating the basic code for one specific model (RNN, LSTM, or Transformer NN).