Closing Price Prediction and Sentiment Analysis

ABSTRACT

Machine Learning models can be used for predicting closing price by making use of historical data without manual intervention. Sentiment analysis can also be used to determine whether to invest in Nvidia in 2020 based on available prediction resources.

Deepika Srinivasan

Artificial Intelligence- CS5700

ACKNOWLEDGEMENTS

I would like to thank our professor Dr. Md Farhadur Reza for guiding us and encouraging us with valuable feedback.
In addition, I also offer our sincere thanks to Yahoo for documenting data and metadata on stock prices for various companies that is used in this experiment.

INTRODUCTION

Best artificial intelligence stocks to buy today systematically use technologies such as machine learning, and neural networks as their main business feature. Predicting the closing price is one of the tedious tasks in Trading industry. This involves lot of manual work and understanding the previous day and even months of data.

Nvidia, the advanced semiconductor company whose technology plays a central role in many young, high-growth tech areas. Central to Nvidia's leadership is the graphics processing unit (GPU) of the company, which power autonomous vehicles, high-performance gaming, cloud computing, and many other areas that require profound learning. NVDA's first-mover edge in high-performance GPUs has eroded competition, but it remains a industry leader and one of the strongest markets for buying AIs.

Significance of Problem

As mentioned above, data needs to be scrutinized and compared with various factors to understand and predict the right closing price. In fact, human error might result in huge loss or deficit for the companies. Even though data is available in abundance, time and effort is needed to come up with right prices.

Objective

The main goal of this assignment is to predict closing prices for Nvidia, which stands at the forefront of artificial intelligence and machine learning, powering a future of self-driving vehicles and cloud gaming. Deep Learning models can be used for predicting closing price using historical data without manual intervention once model is trained with enough data. We will make use of dataset to train our Recurrent Neural Network model to find the closing price of future.

In addition, we will also make use of Sentiment analysis and predict whether to invest in Nvidia in 2020 based on available forecast prediction resources.

Related works

K. Pawar, R. Jalem and T. Vivek have analyzed Historical stock data with the standard Machine Learning Algorithms — Regression, Support Vector Machine, Random Forest, Feed Forward Neural Network and Backpropagation against LSTM RNN which focusses on time series data [1]. Furthermore, they have addressed how the customer's emotions will impact stocks along with pattern shifts.

An alternative methodology is proposed to classify the latent dynamics in the data using deep learning algorithms. The sliding window technique is used to estimate the price of companies listed in the NSE and to compare their results [2].

L. Yifan, Q. Zengchang, L. Pengyu, W. Tao are proposing a model to evaluate online stock forum sentiment and use the details to forecast stock prices in the Chinese market. Sentimental dictionary is built, and score is added to each element linked to a specific stock. By using the Recurrent Neural Networks (RNNs) these metrics are merged into market data for price volatility prediction [3].

Frameworks and Libraries

TensorFlow: End to end open source platform for machine learning

Keras: High-level neural networks library which is running on the top of TensorFlow

Scikit-Learn: Efficient for data mining and data analysis

Matplotlib: Plotting library for graphical representations

Numpy: General purpose array processing package

Pandas: Data manipulation and analysis package

Pandas_datareader: Extract data from various Internet sources into a pandas DataFrame

Keras: Deep Learning library for Theano and TensorFlow

Keras is a Python-written high-level neural network API that can run on top of either TensorFlow or Theano. It was designed to allow rapid experimentation with a focus. It is necessary for good research to be able to go from idea to outcome with the least possible delay.

Advantages of Keras

- Allows easy and quick prototyping (user-friendly, modular and extensible).
- Supports both convolutional and recurrent networks and combinations of both.
- Easy operation on the CPU and GPU.

Artificial Neural Network is an information processing technique. It functions like the way information is processed by the human brain. ANN includes many related processing units working together and produce meaningful outcomes from it.

It can be used for both for classification and regression of continuous numeric attribute.

Mathematical function that maps a given input to a desired output. It consists of the following components:

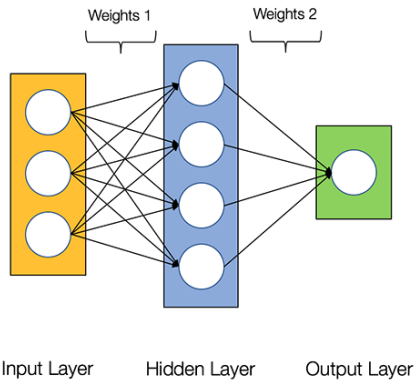
▪ Input layer, x - Raw information that can feed into the network.

▪ Hidden layers - To determine each hidden unit's activity. The input unit activities and the weights on the input connections to the hidden units. One or more secret layers can exist.

▪ Output layer, ŷ - The output unit behavior depends on the hidden unit activity and the weights between the hidden units and the output units.

▪ Weights and Biases between each layer

▪ Activation function(σ) for each hidden layer



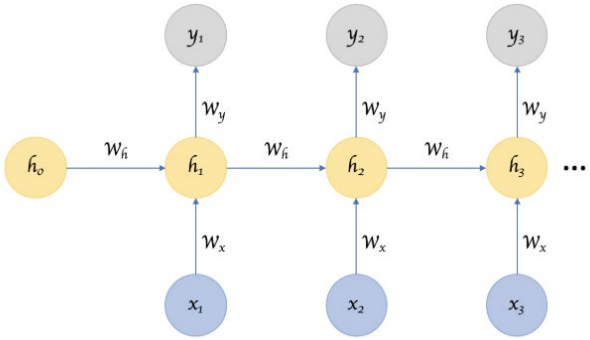
The diagram above shows the architecture of a 2-layer Neural Network

Recurrent Neural Network

A recurrent neural network can be thought of as multiple copies of the same network, every network passing a message to a successor. Recurrent Neural Network knows the past, and what it has learned from the past affects its decisions. Although RNNs learn similarly during preparation, they also recall things learned while generating output(s) from previous input(s).

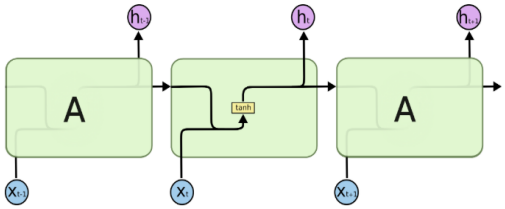
RNNs will take one or more input vectors and generate one or more output vectors, and output(s) are determined not only by weights added to inputs such as a standard NN, but also by a "secret" state vector representing

meaning dependent on input(s)/output(s) beforehand. And, based on previous inputs in the sequence, the same input may produce a different output.

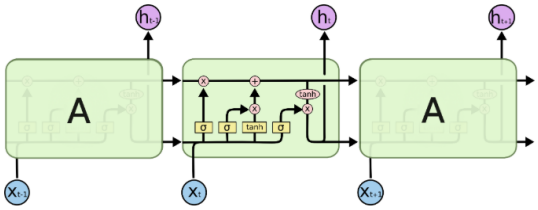


LSTM

LSTM is a specific form of recurrent neural community which works for many tasks much higher than the general version. LSTMs are explicitly designed to avoid the long-term dependency problem. It remembers data for long intervals of time. Most recurrent neural networks take the form of a series of neural network repeated modules. This repeated module would have a basic structure as shown below in regular RNNs, for instance a single tanh layer.

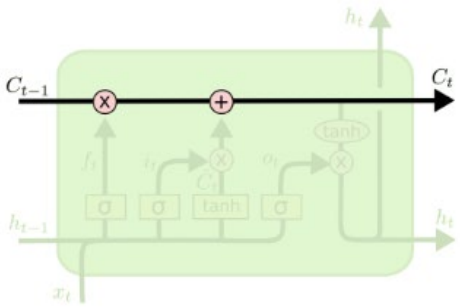


As a structure, LSTMs do have this chain but the repeated module has a different structure. Rather than having a single layer of neural network, there are four, which communicate in a unique way shown below

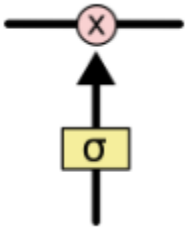


LSTM cell state

The essential component of LSTM is the state of the cell, the horizontal line running along the diagram's top which is like a conveyor belt. This continues right down the entire chain, with just a few small linear interactions. It is very possible for information just to flow unchanged through it.



The LSTM does have the power to delete or attach information that is tightly controlled by mechanisms called gates to the cell state. Optionally, gates are a means of getting knowledge in. These are constructed of a sigmoid neural net layer and an operation of point-wise multiplication.



The sigmoid layer emits zero and one numbers, specifying how much of each part can be allowed in. A value of zero means allows nothing in, while a value of one allows everything through. An LSTM has three of these gates to shield and control the state of the cells.

Dataset Overview

The dataset used in this report is taken from the Yahoo source. It consists of 6 columns out of which we are concerned about 'Close' which provides closing price for Nvidia for the requested timeframe. Number of training samples to be used can be determined based on the model. It consists of the following attributes. In this experiment, we have taken data from '2012-01-01' to '2019-12-30' which constitutes for about 2011 records and finally predict the closing price for '2019-12-31'.

Attribute
High
Low
Open
Close
Volume
Adj

Dataload

Python is used for downloading financial data from free online repositories, processing the downloaded data and then generating some simple technical metrics that will be used as the basis for our quantitative strategy. To accomplish this, we will use one of the most popular and commonly used Python packages to manipulate data, called 'pandas'.

Once we install the required libraries such as pandas-datareader, it is easy to import the library to access market data from a free online source such as Yahoo Finance. Because pandas have a simple remote access to the data from the Yahoo Finance API, we can call the function below to get the appropriate data.

```
df = web.DataReader('NVDA', data_source='yahoo', start=start_date, end=end_date)
```

For our prediction, we will first load data using Pandas where 'df' stands for data frame. The 'head ()' function will show the first few rows of the data frame so we can check that the data has been loaded properly and can take an initial look at how the data is structured.

Date	High	Low	Open	Close	Volume	Adj Close
2012-01-03	14.400000	14.010000	14.300000	14.040000	11701100.0	12.922127
2012-01-04	14.260000	13.920000	14.050000	14.200000	8684300.0	13.069386
2012-01-05	14.780000	14.070000	14.130000	14.710000	14088700.0	13.538781
2012-01-06	14.710000	14.370000	14.700000	14.540000	13331300.0	13.382315
2012-01-09	14.820000	14.440000	14.550000	14.540000	12706100.0	13.382315

```
2 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2011 entries, 2012-01-03 to 2019-12-30
Data columns (total 6 columns):
High         2011 non-null float64
Low          2011 non-null float64
Open         2011 non-null float64
Close        2011 non-null float64
Volume       2011 non-null float64
Adj Close    2011 non-null float64
dtypes: float64(6)
memory usage: 110.0 KB
```

Data Visualization

Visualization of data is the discipline of trying to understand data by placing it in a visual context in order to expose patterns, trends and correlations that might otherwise not detected. Python offers several great libraries of graphing that come packed with many different

features. Few of the popular plotting libraries are as follows:

Matplotlib: low level, provides lots of freedom

Pandas Visualization: easy to use interface, built on Matplotlib

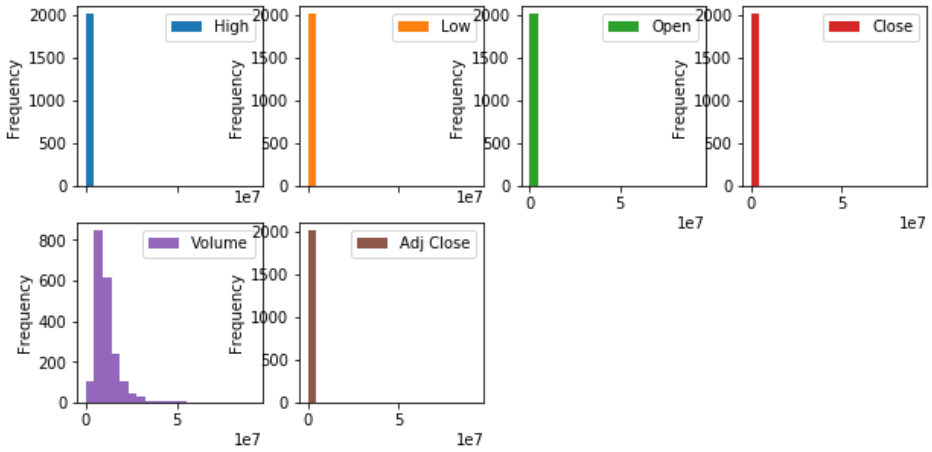
Seaborn: high-level interface, great default styles

Plotly: can create interactive plots

Data Analysis

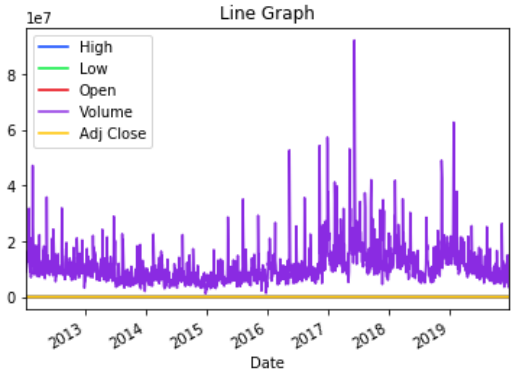
Histogram

In Pandas, we can use the plot.hist method to create a Histogram. There are no arguments required, but we can pass some such as bin size as an option.

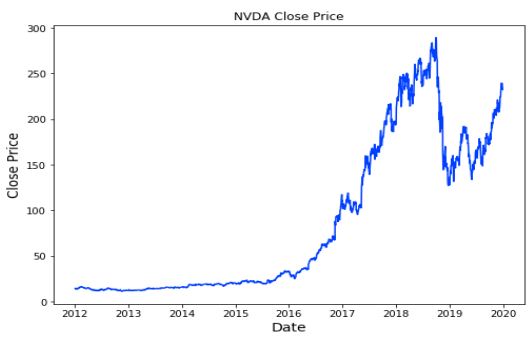


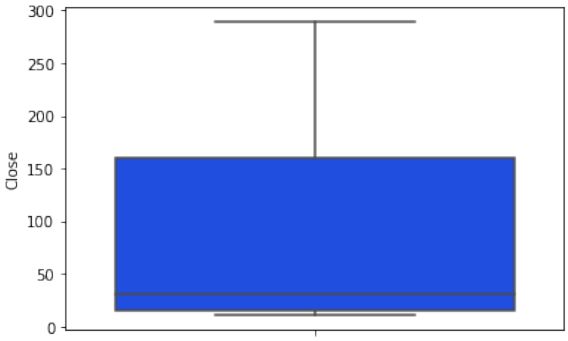
Line Chart

We can call < dataframe>.plot.line to create a line-chart in Pandas after dropping the 'close' column.



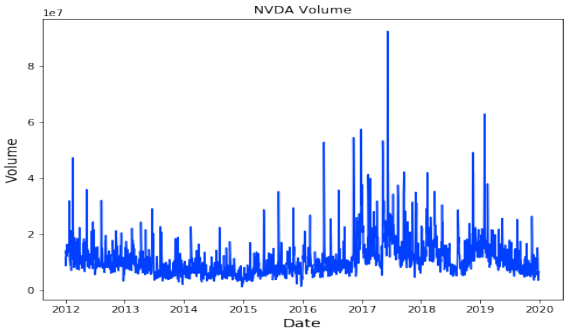
Visualize Closing price Trend





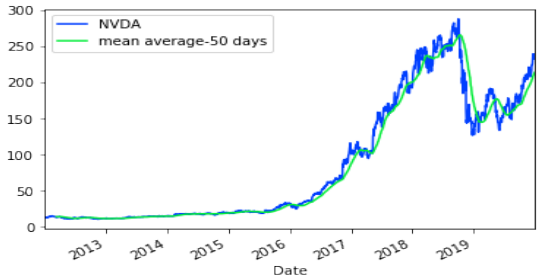
From the above figure we can understand that Closing price of NVDA was around 20 during 2012 and quickly rose to 290 in the mid 2018 and then plunged down after that.However we can also witness upward trend after 2019.

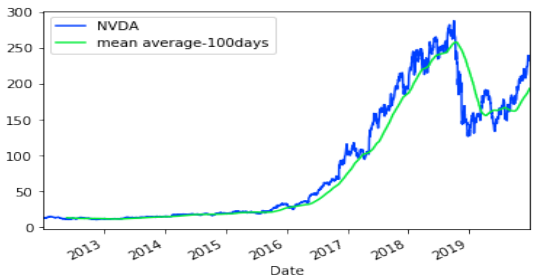
Visualizing NVDA Volume data trend



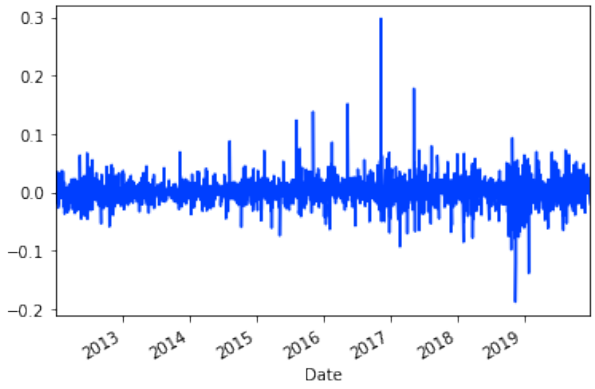
Adjacent Close Data

A simple moveable average of the initial time-series is determined by taking the sum of the last W prices for each year. Pandas has rolling(), a built-in Sequence function that returns a rolling object for a user-defined window. After receiving a rolling object, a variety of functions can be added to it, such as sum(), std() and mean() over the user-defined window.For NVDA 50 and 100 days window is compared as shown below





Finally, we have derived return value and obtained below graph which depicts that returns in 2017 were highest and then it decreased.



Comparison -Best AI Trading companies

Via virtual assistants, facial-recognition apps, navigation tools, and a variety of other devices, millions of customers communicate with AI directly or indirectly everyday [4].In this study we will compare 10 best AI based trading company with special focus on NVDA using the below code.

```
dfcompare = web.DataReader(['NVDA','AMZN','GOOG','FB','CRM','MSFT','BIDU','TWLO','INTC','TCEHY',],'yahoo',start=start_date, end=end_date)['Close']
```

The finding below comes as we attempt to aggregate the data for the first three days and then the last three days to examine how the pattern has shifted from 2012 to 2019 over the years.

```
1 dfcompare.head(3)
```

Symbols	NVDA	AMZN	GOOG	FB	CRM	MSFT	BIDU	TWLO	INTC	TCEHY
Date										
2012-01-03	14.04	179.029999	331.462585	NaN	25.299999	26.77	124.290001	0.0024	24.540001	4.104
2012-01-04	14.20	177.509995	332.892242	NaN	24.370001	27.40	121.989998	0.0015	25.110001	4.086
2012-01-05	14.71	177.610001	328.274536	NaN	24.692499	27.68	123.269997	0.0015	25.400000	4.062

```
1 dfcompare.tail(3)
```

Symbols	NVDA	AMZN	GOOG	FB	CRM	MSFT	BIDU	TWLO	INTC	TCEHY
Date										
2019-12-26	239.190002	1868.770020	1360.400024	207.789993	164.509995	158.669998	127.580002	102.610001	59.820000	48.810001
2019-12-27	236.869995	1869.800049	1351.890015	208.100006	164.979996	158.960007	126.449997	100.430000	60.080002	49.240002
2019-12-30	232.320007	1846.890015	1336.140015	204.410004	162.440002	157.589996	126.290001	97.790001	59.619999	48.619999

From the results we can see that Nvidia, Amazon , Google and Facebook have shown tremendous growth.

pct_change() determines by default the percentage difference from the immediate preceding lines. This is useful to measure the percentage of change in a series of elements over time.

Symbols Date	NVDA	AMZN	GOOG	FB	CRM	MSFT	BIDU	TWLO	INTC	TCEHY
2012-01-03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2012-01-04	0.011396	-0.008490	0.004313	NaN	-0.036759	0.023534	-0.018505	-0.375000	0.023227	-0.004386
2012-01-05	0.035916	0.000563	-0.013871	NaN	0.013233	0.010219	0.010493	0.000000	0.011549	-0.005874
2012-01-06	-0.011557	0.028152	-0.013642	NaN	0.023185	0.015535	-0.017685	0.000000	-0.005905	-0.017725
2012-01-09	0.000000	-0.022178	-0.042399	NaN	-0.003067	-0.013163	-0.008093	0.000000	0.008713	0.008521
...	...	...	...	...	...	...	...	...	...	...
2019-12-23	-0.002298	0.003638	-0.000556	-0.000582	-0.004923	0.000000	0.010511	0.001297	0.004750	0.006438
2019-12-24	-0.000838	-0.002114	-0.003914	-0.005141	-0.002993	-0.000191	-0.002639	0.001594	0.003039	-0.002476
2019-12-26	0.002389	0.044467	0.012534	0.013017	0.007718	0.008197	-0.007082	0.020487	0.006901	0.009723
2019-12-27	-0.009699	0.000551	-0.006256	0.001492	0.002857	0.001828	-0.008857	-0.021245	0.004346	0.008810
2019-12-30	-0.019209	-0.012253	-0.011650	-0.017732	-0.015396	-0.008619	-0.001265	-0.026287	-0.007657	-0.012591

2011 rows × 10 columns

Correlation Matrix

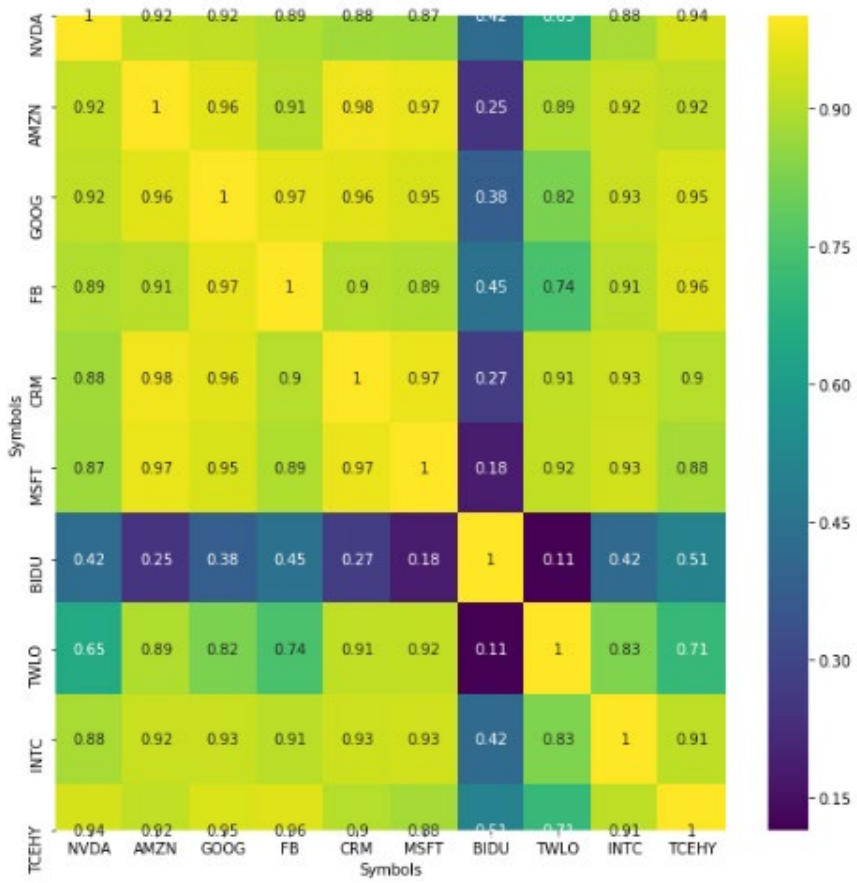
A correlation matrix is a table showing coefficients of correlation between variables. The relationship between two variables is shown by each cell in the table. A matrix of correlation is used to analyze results, as an input for a more detailed analysis and as a diagnostic for advanced analysis.

Dataframe.corr() is used to consider the association of all columns in the dataframe between each pair. Any 'Na' values are excluded automatically. This is ignored for any columns of a non-numeric code form within the dataframe.

Symbols Symbols	NVDA	AMZN	GOOG	FB	CRM	MSFT	BIDU	TWLO	INTC	TCEHY
NVDA	1.000000	0.353772	0.390498	0.282930	0.397822	0.433033	0.317614	0.023622	0.461080	0.317248
AMZN	0.353772	1.000000	0.555392	0.400437	0.432802	0.476455	0.339449	0.017692	0.330243	0.335261
GOOG	0.390498	0.555392	1.000000	0.414404	0.421621	0.531805	0.394795	0.008419	0.375610	0.372857
FB	0.282930	0.400437	0.414404	1.000000	0.375070	0.312987	0.295485	0.009174	0.262461	0.263313
CRM	0.397822	0.432802	0.421621	0.375070	1.000000	0.460191	0.356107	0.022414	0.350640	0.344746
MSFT	0.433033	0.476455	0.531805	0.312987	0.460191	1.000000	0.328410	0.026576	0.535634	0.359889
BIDU	0.317614	0.339449	0.394795	0.295485	0.356107	0.328410	1.000000	0.010206	0.303571	0.433013
TWLO	0.023622	0.017692	0.008419	0.009174	0.022414	0.026576	0.010206	1.000000	0.030988	0.018495
INTC	0.461080	0.330243	0.375610	0.262461	0.350640	0.535634	0.303571	0.030988	1.000000	0.321773
TCEHY	0.317248	0.335261	0.372857	0.263313	0.344746	0.359889	0.433013	0.018495	0.321773	1.000000

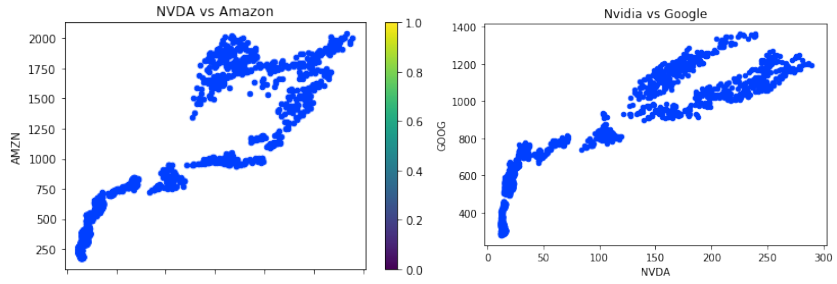
HeapMap

Heat Map with correlation values is a helpful way to test correlations between columns easily by visualizing the matrix as a heat map. The brighter the color, the larger the extent of the association. 'Seaborn' library was used for this purpose.

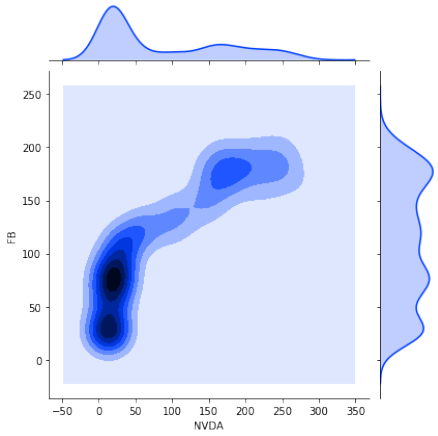


Scatterplot Usage

Two data frame columns describe the coordinates of each point, and filled circles are used to represent each point. Scatterplot is useful to see complex correlations between two AI companies



'seaborn.jointplot' is used to display two variables for bivariate and univariate graphs. This feature provides a simple framework for various canned plot forms to the JointGrid class.



Feature Selection

Based on data analysis on heat map and correlation matrix, we decided that 'Close' data provides better prediction for closing data for future based on historical data.

Data Pre-processing

A new data frame (Close_data) is generated with only the 'Close' column, since it is the only column required for studying the pattern. We need numpy array (Close_data_numpy) generated from data frame created in the first step.

```
3 close_data
4
```

Date	Close
2012-01-03	14.040000
2012-01-04	14.200000
2012-01-05	14.710000
2012-01-06	14.540000
2012-01-09	14.540000
...	...
2019-12-23	238.820007
2019-12-24	238.619995
2019-12-26	239.190002
2019-12-27	236.869995
2019-12-30	232.320007

2011 rows × 1 columns

```
3 Close_data_numpy
```

```
array([[ 14.03999996],
       [ 14.19999981],
       [ 14.71000004],
       ...,
       [239.19000244],
       [236.86999512],
       [232.32000732]])
```

Feature Scaling

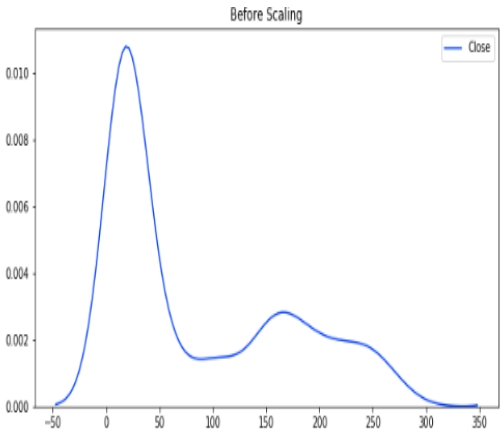
From the above figures, we can understand that 'Close' data is not scaled as it is different range so we will make use of MinMaxScaler function to achieve the goal .This rescales the features in such a way that they have the properties of a normal standard distribution with a zero mean and one standard deviation.

min-max scaler method scales the dataset so that all the input features lie between 0 and 1 inclusive

scaler = MinMaxScaler(feature_range=(0,1))

Close_data_numpy_scaled = scaler.fit_transform(Close_data_numpy)

Close_data_numpy_scaled




```
array([[0.00956903],
       [0.01014461],
       [0.01197928],
       ...,
       [0.81951944],
       [0.81117349],
       [0.79480545]])
```

Data Split- Train and Test

Next, we need to split up our dataset into Training and Test dataset. Since 80:20 is most widely used measure we also split the data using the same measure.

Training data count is 1609 while Test Data count is 402

Shape of Training data is (1609, 1)

Number of data in the Training set is 1609

Training data is from index 0 to 1608

Testing data starts from index 1609 to 2011

Batch size

Batch size helps to split the time series data in different windows. In this project, we have made use of '60' as batch size and split the training and testing data. As a result, we obtain definite number of batches as shown below

We will also have to convert this batch of data into array as we did in pre-processing step. In addition, we may need to reshape the data in to 3 dimensional structure(number of samples, number of time steps, and number of features) in order to fit LSTM requirements.

Number of batches: 1549 obtained with batch size: 60

batch_x_training_data = np.reshape(batch_x_training_data, (batch_x_training_data.shape[0], batch_x_training_data.shape[1], 1))

batch_x_training_data.shape

Output= (1549, 60, 1)

Training Neural Network Model

The output ŷ of a simple 2-layer Neural Network is:

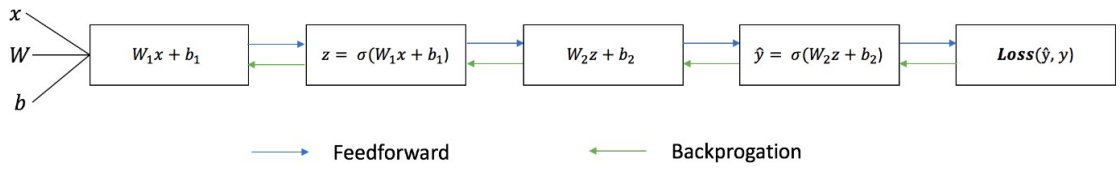
$$\hat{y} = \sigma(W_2\sigma(W_1x + b_1) + b_2)$$

We can notice that the weights W and biases b are the only variables affecting the output in the above equation. The correct weight and bias values, of course, determine the strength of the predictions. The process of fine-tuning the input data weights and biases is known as the Neural Network training.

Each training process iteration consists of the following steps:

Calculation of the predicted output, known as feedforward

Updating weights and biases, known as backpropagation The sequential graph below illustrates the process.



Feedforward

As we saw in the sequential graph above, feedforward is a simple calculation and the performance of the Neural Network is for a typical2-layer neural network:

$$\hat{y} = \sigma(W_2\sigma(W_1x + b_1) + b_2)$$

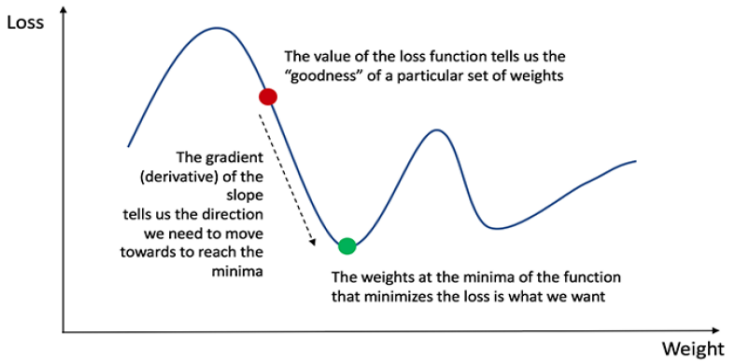
Nonetheless, we still need a way to evaluate our predictions which can be obtained using the Loss function.

Loss Function

The function used to evaluate a candidate solution (i.e. a set of weights) is called the objective function in the context of an optimization algorithm. We may either try to maximize or minimize the objective function, which means we are looking for a candidate solution with the highest or lowest score, respectively. Typically, we try to minimize the error with neural networks. We need to find the optimal set of weights and biases which will minimize the loss function.

Backpropagation

Now that we have calculated our prediction error (loss), we need to find a way to propagate the error and update our weights and biases. We need to know the derivative of the loss function in terms of weights and biases in order to know the appropriate amount to change the weights and biases by.



Sequential Model

Keras ' core data structure is a model, a way for layers to be organized. The simplest type of model is a linear stack of layers, the Sequential model. We can use Keras functional API for more complex architectures, which enables arbitrary layer graphs to be built.

```
#Build the LSTM model
model = Sequential()
model.add(LSTM(100, return_sequences=True, input_shape= (batch_x_training_data.shape[1], 1)))
model.add(LSTM(100, return_sequences= False))
model.add(Dense(25))
model.add(Dense(1))
```

The Sequential model is a linear stack of layers which is explained as follows

- Initially, we have first layer with input shape as the number of steps that we decided (in this case it would be 60) and the number of feature is 1 which is the 'close' data. We have also indicated that return sequence as True because we will be adding more layers later. Also, number of neurons in this layer is 100.
- Secondly, we will add one more LSTM layer with 100 neurons and also update return sequence as False since we do not have any LSTM layers after this layer.
- Thirdly, Dense layer is added with 25 neurons which act as a regular densely connected Neural Network.
- Finally, we have the last Dense layer with 1 neuron which provides the final outcome of our model.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 60, 100)           40800
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_1 (Dense)              (None, 25)                2525
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 26
=================================================================
Total params: 123,751
Trainable params: 123,751
Non-trainable params: 0
_________________________________________________________________
```

```
2  model.layers
```

```
[<keras.layers.recurrent.LSTM at 0x1adfbafdac8>,
 <keras.layers.recurrent.LSTM at 0x1adfbafdf60>,
 <keras.layers.core.Dense at 0x1adfbb86358>,
 <keras.layers.core.Dense at 0x1adfd0f1470>]
```

Model Compilation

We need to configure the learning process, which is done using the compile method, before training a model. Below three arguments are given to it:

• An optimizer: Existing optimizer's string identifier or an Optimizer class instance.

• A loss function: Goal of the model is to minimize loss function. It may be an existing loss function's string identifier or it may be an objective function.

• A list of metrics: A metric may be an existing metric string identifier or a custom metric function.

Optimizer

Optimizers helps to minimize (or maximize) the Objective function. For example, we call the neural network's Weights(W) and Bias(b) values its internal learning parameters that are used in the measurement of output values and are trained and modified in the direction of optimal solution, i.e. minimizing the loss through the training process of the network and also playing a major role in the Neural Network Model training process. Optimizers controls the learning rate. The learning speed determines how easily the correct weights are determined for the model. A lower learning level can result in more accurate weights (up to a certain point), but it will take longer to calculate the weights.

- **Adam- Adaptive Moment Estimation**

Adam is a method for calculating adaptive learning rates for each parameter. As Adam converges quickly, when compared to other optimizers it works well. Also learning speed of the model is high. It also corrects every problem faced by other optimization techniques such as fading learning rate, slow convergence or high variance in parameter updates which leads to fluctuating functional losses

In my project, the question was to select the best optimizer for our Neural Network Model to converge quickly and to properly learn and adjust the internal parameters to minimize the loss function. When experimenting with other optimizers, Adam works well and exceeds other adaptive techniques.

- **Stochastic Gradient Descent**

Stochastic gradient descent (often abbreviated SGD) is an iterative approach for optimizing an objective function with enough properties of smoothness. It can be called a stochastic approximation of gradient downward optimization since it replaces the true gradient. We can do this in Keras to permit SGD + Nesterov, it works well for shallow networks. Nesterov momentum has significantly less over-shooting compared to the normal momentum

- **Adagrad**

Adagrad allows big changes for uncommon parameters and minor changes for extensively used parameters. It is well suited for working with fragmented data for that purpose.

- **Adadelta**

Adadelta is an AdaGrad extension that appears to eliminate the increasing Rate learning problem. In addition, we don't have to set a learning pace by choice.

Loss function

Machines learn through loss functions. It is a way of determining how well the data is modelled by different algorithms. Gradually, the loss function learns to reduce the error in estimation with the aid of some optimization function. Generally, loss functions can be divided into two main categories depending on the type of learning activity we deal with — regression and classification loss function.

- **Mean Squared Error loss**

Mean square error is calculated as the square difference between forecasts and real measurements on average. It is just about the average magnitude of error regardless of their direction. However, forecasts that are far from real values are strongly penalized as opposed to less deviated expectations, owing to squaring.

- **Mean Squared Logarithmic Error Loss**

Regression problems in which the target value has a number of values and when forecasting a large value we can first measure the normal logarithm of any of the expected values rather than abusing a variable as harshly as a mean squared error, then measure the mean squared error. This is termed the logarithmic error loss which has the effect of easing the punishing effect of huge disparities in broad values expected.

- **Mean absolute error loss**

The distribution of the target variable can be mainly Gaussian on certain regression issues, but may have outliers, e.g. high or low values far apart from the mean value. In this case, the Mean Absolute Error, or MAE, error is an

acceptable error function since it is more reliable for outliers. This is measured as the sum of the total difference between expected and real values.

Model Training

We will use the function fit() on our model to train with the following five parameters: training data (batch_x_training_data), target data (batch_x_training_data) and number of epochs.
The number of epochs is the number of times the model will cycle through the data. The more epochs we run, the stronger the model, to some degree. The model should stop improving after that point during each epoch.

In this project, since we are dealing with regression problem of predicting the stock price using batch size of 60, there is no need to train on multiple epochs.

history=model.fit(batch_x_training_data, batch_y_training_data, batch_size=1, epochs=1)

```
Epoch 1/1
1549/1549 [==============================] - 179s 116ms/step - loss: 0.0016
```

Model weights

As the model gets trained for 1543 sets of 60 sample observation each, weights are also getting updated so that we will be able to get prediction for Dec 31st 2019 based on last 60 days data. We can find how the weights have been updated during model training and the shape of weight using below code.

model.get_weights()

for weight in model.get_weights():

print (weight.shape)

```
(1, 400)
(100, 400)
(400,)
(100, 400)
(100, 400)
(400,)
(100, 25)
(25,)
(25, 1)
(1,)
```

Model Evaluation

We have trained the model so far using 80% data, and we need to pre-process the remaining 20% data and then start testing the model.

```
Number of data in batch_x_testing_data : 402 and batch_y_testing_data : 402
```

Similar to Training data, we will also need to convert the Testing data as numpy array and reshape to fit LSTM model

```
#Final shape of the training and testing dataset used in our model
print("*********************************************************************")
print("Training data shape: ", batch_x_training_data.shape)
print("Test data shape    : ", batch_x_testing_data.shape )
print("*********************************************************************")
```

```
*********************************************************************
Training data shape:  (1549, 60, 1)
Test data shape    :  (402, 60, 1)
*********************************************************************
```

Predictions

We will make use of predict() function to obtain the model predictions for testing data based on the model we created using training data

predict_close = model.predict(batch_x_testing_data)

It should be noted that we need to reverse the values back to unscaled data as initially we have used MinMaxScaler to normalize the data.

predict_close = scaler.inverse_transform(predict_close)

RMSE

Root Mean Square Error (RMSE) is a typical method for calculating a model's error when forecasting quantitative results. It is formally defined as:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

RMSE has a two function in data science:

• Acting as a heuristic for training models

• Testing trained models for usefulness / accuracy

```
root_mean_square_value=np.sqrt(np.mean(((predict_close- batch_y_testing_data)**2)))
root_mean_square_value
```

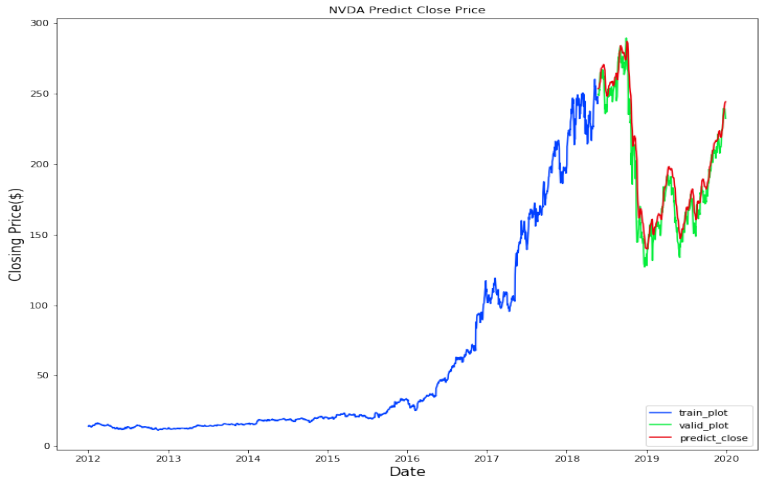
```
11.5923518804296
```

In contrast, Training data RMSE is found to be 92.93

```
#Obtain model predictions for testing data based on the model we created using training data
predict_close_train = model.predict(batch_x_training_data)
#Reverse the values back to unscaled data
predict_close_train = scaler.inverse_transform(predict_close_train)
#RMSE calculates how well the model has predicted the values
root_mean_square_value_train=np.sqrt(np.mean(((predict_close_train- batch_y_training_data)**2)))
root_mean_square_value_train
```

```
92.93121522313963
```

Based on the evaluation, we will plot the results to compare the real values in the test data, and the estimation of the model to explain the model's performance.



From the above graph, we will see that the estimation of the model is very similar to the real values. To get a better view, we must always look directly at the values.

```
valid_plot.head(5)
```

Date	Close	predict_close
2018-05-25	249.279999	253.417389
2018-05-29	248.589996	253.984177
2018-05-30	252.990005	254.503876
2018-05-31	252.190002	255.621246
2018-06-01	257.619995	256.650452

```
valid_plot.tail(5)
```

Date	Close	predict_close
2019-12-23	238.820007	237.471481
2019-12-24	238.619995	240.063431
2019-12-26	239.190002	242.029114
2019-12-27	236.869995	243.517197
2019-12-30	232.320007	244.203186

We calculate the closing price for test date '2019-12-31' with the model prediction and the real closing price

```
NVDA's Predicted Closing Price for 2019-12-31 is :  243.74408
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

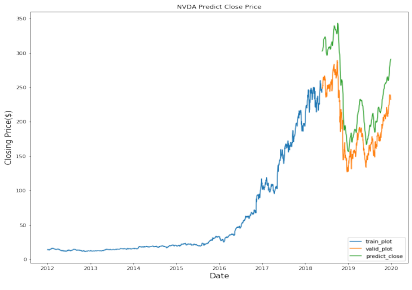
We have tested the model for a new batch of data that would use the last 60 days to estimate the trading price for '2020-01-16' in addition to the test results.

```
NVDA's Predicted Closing Price for 2020-01-16 is :  252.51556
**********************************************************************
NVDA's Actual Closing Price
Date
2020-01-16    248.929993
Name: Close, dtype: float64
```

Comparison of Model

- **Significance of Loss**

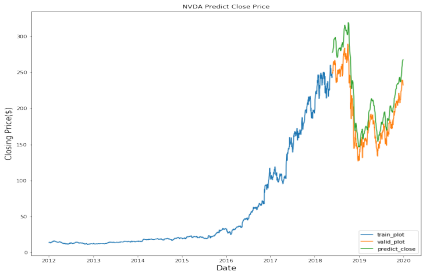
Mean Squared Logarithmic Error




```
NVDA's Predicted Closing Price for 2019-12-31 is :  290.5988
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

RMSE – 45.73

Mean Absolute Error



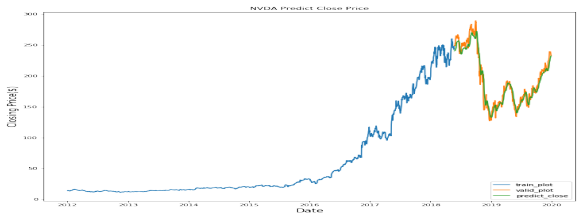

```
NVDA's Predicted Closing Price for 2019-12-31 is :  266.68677
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

RMSE -26.33

Observation: We can see that from above comparison, Mean Square error loss seems to be a better fit for our model

- **Significance of Optimizers**

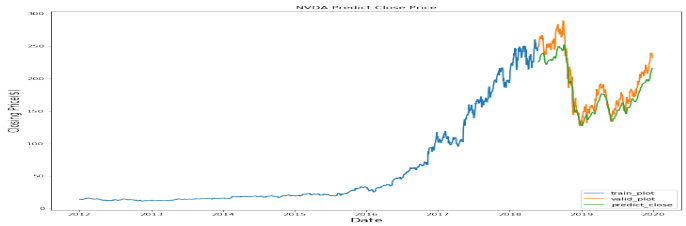
Adagrad




```
NVDA's Predicted Closing Price for 2019-12-31 is :  232.16678
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

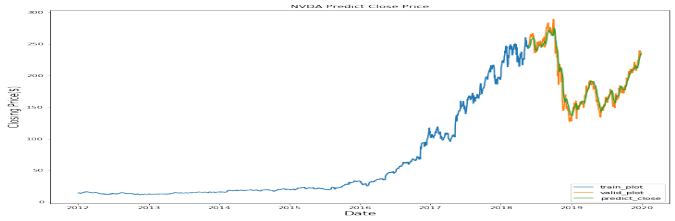
RMSE-9.77 and Loss =0.0012

Adadelta



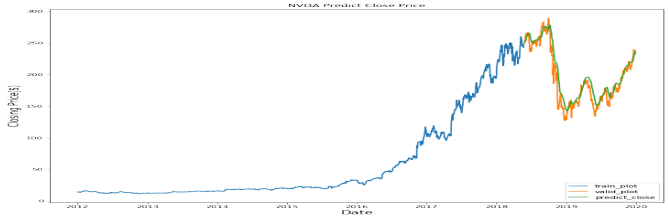
RMSE : 17.63 and Loss: 0.0364

Adamax



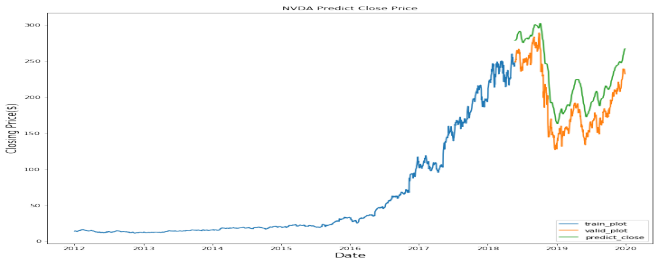
RMSE: 9.24 and Loss: 0.0256

SGD



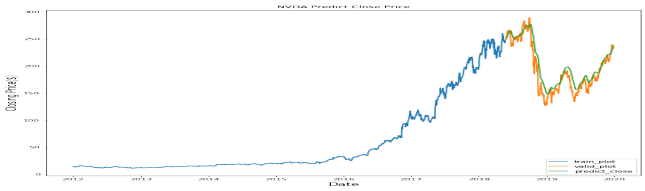
RMSE: 13.52 and Loss: 0.0538

SGD with Momentum



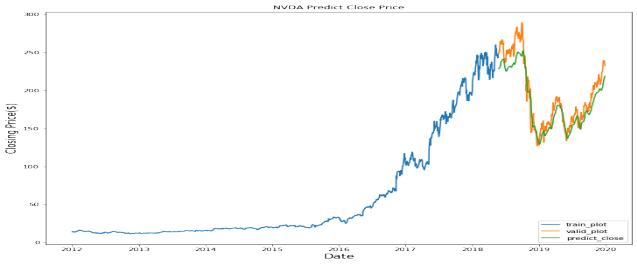
RMSE: 34.86 and Loss:0.0732

SGD and Nesterov



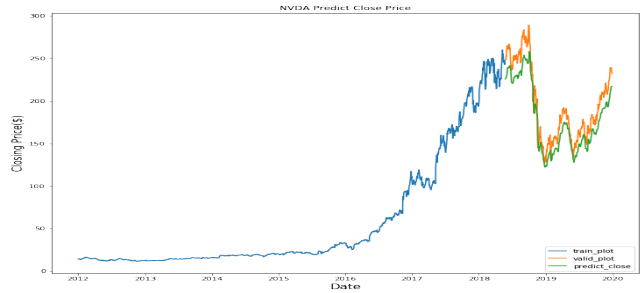
RMSE: 15.63 and Loss: 0.0482

SGD with Nesterov and Momentum



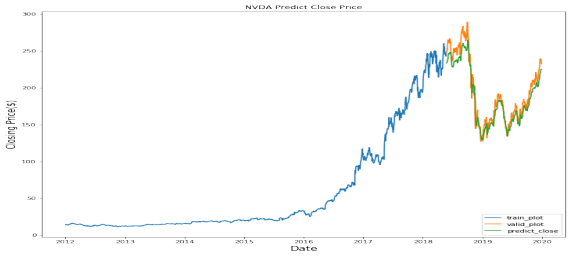
RMSE: 15.70 and Loss: 0.0464

Rmsprop



RMSE: 17.93 and Loss: 0.0295

Nadam



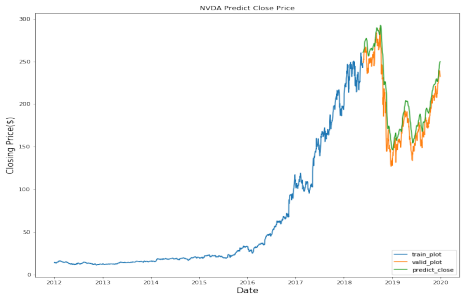
RMSE: 11.54 and Loss 0.0277

Observation: In terms of RMSE value, Adamax is found to be better than Adam.

- **Significance of Batch size**

  We will compare the model based on the number of days considered for every batch

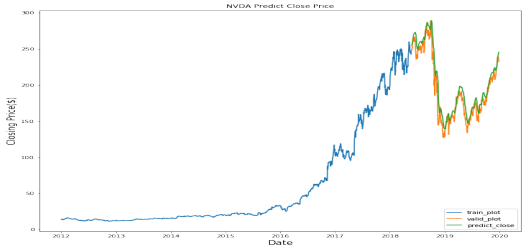
30 days




```
NVDA's Predicted Closing Price for 2019-12-31 is :  249.97661
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

RMSE: 17.25 and Loss: 0.0019

90 days




```
NVDA's Predicted Closing Price for 2019-12-31 is :  245.68343
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

RMSE: 12.47 and Loss: 0.0022

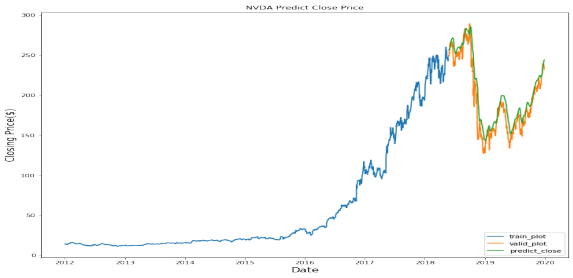
Observation: From the above figures and RMSE value we can see that 60 days in a batch is the best option for our model.

- **Significance of Neurons**

Number of Neurons 50

```
#Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape= (batch_x_training_data.shape[1], 1)))
model.add(LSTM(50, return_sequences= False))
model.add(Dense(25))
model.add(Dense(1))
```

```
Total params: 31,901
Trainable params: 31,901
Non-trainable params: 0
```

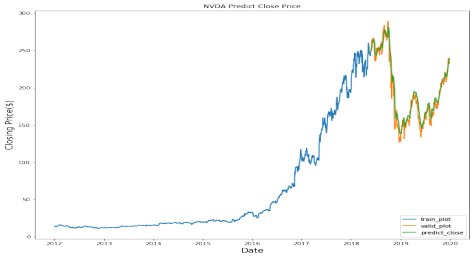



```
NVDA's Predicted Closing Price for 2019-12-31 is :  244.89087
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

RMSE:14.50 and Loss: 0.0014

Number of Neurons 150

```
#Build the LSTM model
model = Sequential()
model.add(LSTM(150, return_sequences=True, input_shape= (batch_x_training_data.shape[1], 1)))
model.add(LSTM(150, return_sequences= False))
model.add(Dense(25))
model.add(Dense(1))
```




```
NVDA's Predicted Closing Price for 2019-12-31 is :  238.98344
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

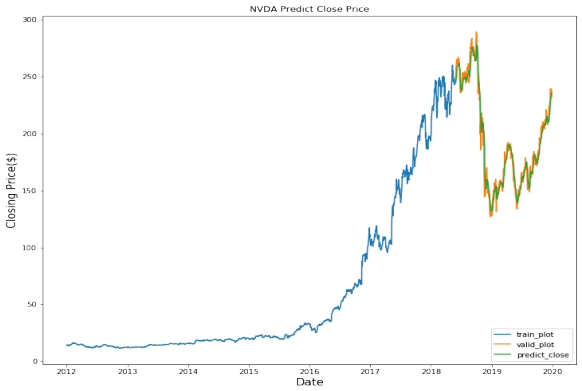
RMSE: 9.52 and Loss: 0.0019

Observation: As the complexity of the model increases, when we increase the number of neurons, we can see that the performance of our model was higher as the number of neurons increased to 150. This is apparent in the RMSE value of 9.52 relative to 11.59 when the number of neurons was 100.

- **Significance of LSTM Layer count**

LSTM – 1 layer

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100)               40800
_________________________________________________________________
dense_1 (Dense)              (None, 25)                2525
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 26
=================================================================
Total params: 43,351
Trainable params: 43,351
Non-trainable params: 0
_________________________________________________________________
```




```
NVDA's Predicted Closing Price for 2019-12-31 is :  234.97708
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

RMSE: 8.12 and Loss: 0.0012

Observation:

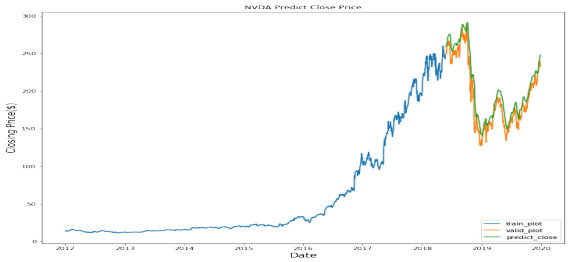
By reducing one of the LSTM layer we have reduced the number of parameters from 123,571 to 43,351 RMSE from 11.59 to 8.12 compared to 2 layers of LSTM.

In addition, we can also see that test date closing price prediction by the model is almost the same as actual value.

- **Significance of one LSTM layer and Number of Neurons**

50 Neurons

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 50)                10400
_________________________________________________________________
dense_9 (Dense)              (None, 25)                1275
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 26
=================================================================
Total params: 11,701
Trainable params: 11,701
Non-trainable params: 0
```

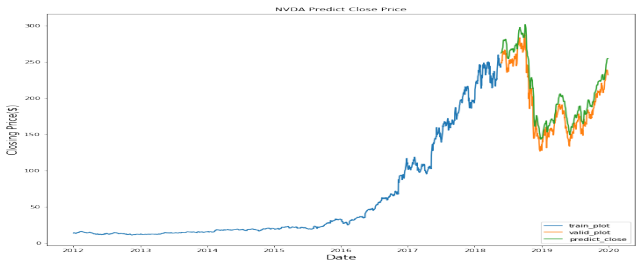



```
NVDA's Predicted Closing Price for 2019-12-31 is :  248.15805
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

RMSE: 14.31 and Loss: 0.0017

150 Neurons

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_6 (LSTM)                (None, 150)               91200
_________________________________________________________________
dense_11 (Dense)             (None, 25)                3775
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 26
=================================================================
Total params: 95,001
Trainable params: 95,001
Non-trainable params: 0
```




```
NVDA's Predicted Closing Price for 2019-12-31 is :  253.01187
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

RMSE: 16.76 and Loss: 9.0538e-04

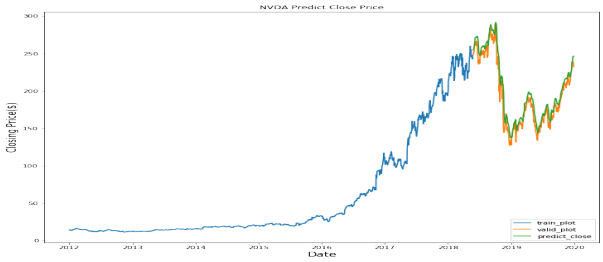
Observation:

Even though RMSE value is higher in both the above case when compared to our base model with Number of neurons as 100, we can find that number of parameters to train is quite low(11,701) when the number of neuron is 50.

- **Significance of one LSTM layer and Dense Layer Neurons count**

Number of Neurons-50

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 100)               40800
_________________________________________________________________
dense_5 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 51
=================================================================
Total params: 45,901
Trainable params: 45,901
Non-trainable params: 0
```

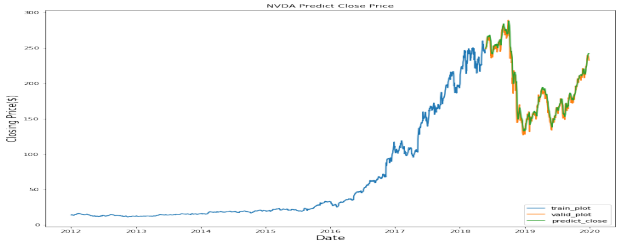



```
NVDA's Predicted Closing Price for 2019-12-31 is :  245.28459
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

RMSE: 10.85 and Loss: 0.0012

Number of Neurons-100

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 100)               40800
_________________________________________________________________
dense_7 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 101
=================================================================
Total params: 51,001
Trainable params: 51,001
Non-trainable params: 0
```




```
NVDA's Predicted Closing Price for 2019-12-31 is :  239.57063
**********************************************************************
NVDA's Actual Closing Price
Date
2019-12-31    235.300003
Name: Close, dtype: float64
```

RMSE: 7.41 and Loss: 0.0013

Observation:

From the above graphs, we can see that Dense layer with 100 neurons performs better due to lower RMSE value and due to comparable test date closing value.

Nvidia-Sentiment Analysis

Sentiment Analysis or Opinion Mining refers to the use of NLP, text analysis and computational linguistics to assess the writer / subject / topic's subjective knowledge or emotional state. It is widely found in articles that save companies a lot of time by reviewing responses manually. It is the most common text analysis method evaluating an incoming message and saying whether the underlying sentiment is positive, negative or neutral.

The python 'newpaper' library is used to collect the articles-related information. That includes the author name, the article's main photos, the release dates, the article's video current, keywords explaining the topic and the article's description.

Article Information

In this report we will study an article published in investorplace.com on Nvidia stock prediction. We can obtain author and date information using newspaper library as shown below.

```
3 print(article.publish_date)
```

```
2020-01-08 13:16:37-05:00
```

```
2 article.authors
```

```
['Louis Navellier', 'The Investorplace Research Staff', 'Investorplace Staff']
```

We can obtain the keywords from the article using keywords method.

```
1 article.keywords
```

```
['market',
 '500',
 'wont',
 'stocks',
 'nvidia',
 '2020',
 'nvda',
 'correction',
 'stock',
 'sp',
 'beat',
 'index',
 'match',
 '2019']
```

Article.text will help to show the entire article while article.summary will provide summarized version of the text.

```
1 text=article.summary
2 print(text)
```

```
Nvidia (NASDAQ: NVDA ) had an excellent year in 2019 with Nvidia stock gaining 74%, a considerable bounce-back from a 31% de
cline the year before.
Nvidia Stock Won't Correct in 2020InvestorPlace's Laura Hoy recently called Nvidia one of the 10 best stocks to buy in 2020.
In late November, I suggested that Nvidia stock could repeat its strong 2019 performance in 2020.
If its net income rises, the multiple of Nvidia stock will almost definitely increase as well."For now, momentum is on the s
ide of Nvidia stock.
Despite the expected slowdown in the S&P 500, I expect it to deliver a moderate, if not spectacular, performance in 2020.
```

Tokenization

Tokenization is the mechanism by which vast volumes of text are separated into smaller sections called tokens. Natural language processing is used to develop applications such as text recognition, smart chatbot, emotional interpretation, language translation, etc. To attain the above mentioned goal it becomes crucial to understand the trend in the text.

Natural Language toolkit has incredibly significant tokenization feature which also includes sub-modules

- word_tokenize()
- sent_tokenize()

We use the sent_tokenize() function to break the text into individual sentences. So, from the output below we can see that the text is broken into a list with each sentence as an item in the list.

```
1 from nltk import tokenize
2 sentence_list=tokenize.sent_tokenize(text)
3 print(sentence_list)
```

```
['Nvidia (NASDAQ: NVDA ) had an excellent year in 2019 with Nvidia stock gaining 74%, a considerable bounce-back from a 31%
decline the year before.', 'Nvidia Stock Won’t Correct in 2020InvestorPlace’s Laura Hoy recently called Nvidia one of the 10
best stocks to buy in 2020.', 'In late November, I suggested that Nvidia stock could repeat its strong 2019 performance in 2
020.', 'If its net income rises, the multiple of Nvidia stock will almost definitely increase as well.”For now, momentum is
on the side of Nvidia stock.', 'Despite the expected slowdown in the S&P 500, I expect it to deliver a moderate, if not spec
tacular, performance in 2020.']
```

obj.sentiment.polarity

Return value of this function is between -1 and 1.If the result is a negative number then the feeling towards the topic was adverse while favourable topic will score around positive number. On the other hand, if the score is 0 it means that sentiment shown by the article is neutral.

For example, let’s consider one of the sentence as shown below which has a negative sentiment polarity of -0.133

“Despite the expected slowdown in the S&P 500, I expect it to deliver a moderate, if not spectacular, performance in 2020.”

On the whole, all the sentences in the summary are analyzed and the article's overall market analysis is seen which will give us an indication of whether to purchase Nvidia stocks in 2020.

```
Overall sentiment Analysis is : 0.9833333333333334
```

The result given here was 0.9833, which shows that there was an extremely optimistic perception of the feeling about that text because it is not just the positive number but also really close to 1.0!

Conclusion:

Thus, Nvidia closing price is predicted using LSTM model along with Sentiment analysis on an article which discusses about Nvidia's shares in 2020.In addition, this paper also compares different optimizers, loss functions, Role of dimensions(number of neurons),LSTM layers based on their performance with respect to RMSE and test date closing price prediction.

References

[1] K. Pawar, R. Jalem and T. Vivek, "Stock Market Price Prediction Using LSTM RNN," Springer Singapore, 2019.

[2] S. Sreelekshmy, R. Vinayakumar, R. Gopalakrishnan, E. A Menon, K. Krishna, K. P Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," in 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp 1643-1647.

[3] L. Yifan, Q. Zengchang, L. Pengyu, W. Tao, "Stock Volatility Prediction Using Recurrent Neural Networks with Sentiment Analysis," May 6, 2017, [Online], Available : https://arxiv.org/pdf/1705.02447v1.pdf, [Accessed: April 13, 2020].

[4] J. Divine, "Artificial Intelligence Stocks: The 10 Best AI Companies," [Online], Available: https://money.usnews.com/investing/stock-market-news/slideshows/artificial-intelligence-stocks-the-10-best-ai-companies, [Accessed: April 13, 2020].