CSE 674 Project 3: Stock Market Forecasting and Analysis based on Hierarchical-LSTM-AutoEncoder

Neeraj Ajit Abhyankar
Department of Computer Science and Engineering
State University of New York
University at Buffalo
Email – nabhyank@buffalo.edu

Abstract

Machine Learning and Deep Learning has touched almost all the fields ranging from not only science but also medicine, e-commerce, finance, telecommunication, defense and security. Recently, mathematical formulations and variations of various concepts like Bayes, Bernoulli's, Euclidean, Gaussian, Markov's, SVD's have been found to have prominent yields and improvements in the existing models of Machine Learning and Deep Learning. One of the most recent and most trending use of such techniques is the Variational Autoencoders (VAE's) which are used to produce semantical representations which are meaningful for modeling natural data. However, the VAE's face the difficulty of modeling sequences having a long-term memory structure; to address this problem a hierarchical-decoder can be used to generate embeddings for each of the subsequences independently. The above idea of using a hierarchical-decoder to model the long-term memory structures can be used to model not only sequential data but also time series data. In this project we will try to use these hierarchical decoders to model financial data such as the stock market which is in itself a time series and analyze the the same. We will try to use the hierarchical decoder model to predict the values of stock market data using the novel structure of these HVAE's with unique property of modeling continuous data.

1 Introduction :

To get to know about what we will be working with in this project we need to understand various Machine Learning and Deep Learning concepts with their base knowledge and structures, also we will look at various concepts from Stock Market Analysis domain since this project is based on Stock prices and their fluctuations. We will try to understand as many dynamics as we can of the stock pricing and their variations with respect to market trends. Moreover, it will be interesting to understand and provide an intuitive study about the Hierarchical Variational Autoencoder (HVAE) as they are very popular due to their Long-Term sequence modeling benefits. Furthermore, the HVAE's provide us with a novel solution for the problem of posterior collapse by including LSTM layers. As discussed above, lets dive a little deeper into these concepts and explore the domain of Deep Learning and Stock Market.

1.1 What is a Autoencoder :

An Autoencoder network is basically a pair of two connected networks i.e. an encoder and decoder. The encoder network takes the inputs converting them into smaller denser representations, whereas the decoder network can be used to covert these back to original inputs. The below diagram shows the layers in the encoder network converting the representations of the input provided to it.

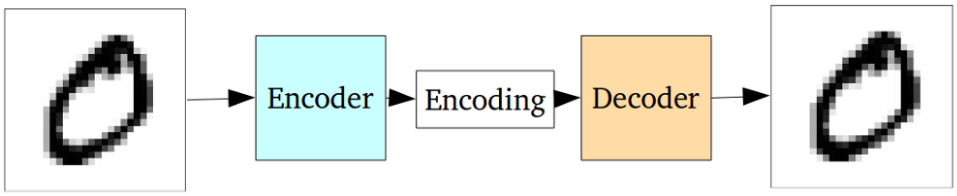


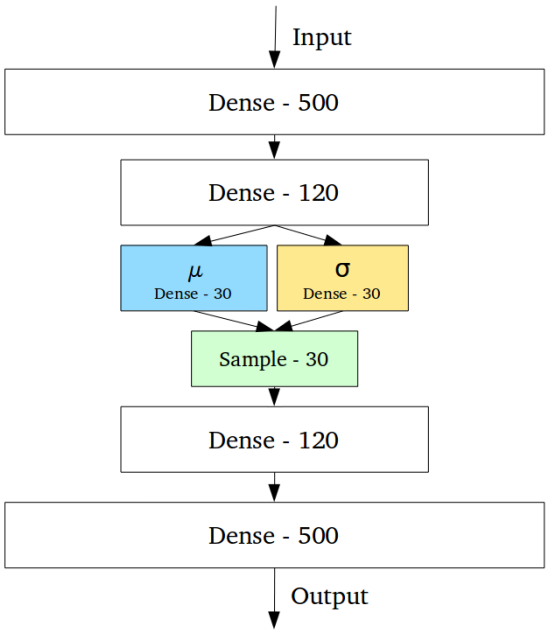
Fig-1.1 Autoencoder Architecture for input reconstruction by encoding and decoding of given sample data.

The entire network is usually trained as a whole and the loss function is usually either the mean-squared error or cross-entropy between the output and the input, known as the reconstruction loss, which penalizes the network for creating outputs different from the input.

As the encoding has far less units than the input, the encoder chooses to discard some unnecessary information. The encoder learns to preserve as much of the relevant information as possible in the limited encoding, and intelligently discard irrelevant parts. The decoder then takes the generated encoding and properly reconstruct it into a full image and together they form an Autoencoder.

1.2 Problems with Autoencoders and use of Variational Autoencoders:

Standard Autoencoders work by learning to generate latent representations which are compact and then the decoder does the reconstruction of these latent representations. The problem associated with Autoencoders is in the encoder part for generation, the latent space they convert their inputs into vectors may not be continuous, or allow easy interpolation. However, a solution to this problem can be achieved by using VAE's as they possess a unique property which separates them from the vanilla Autoencoders which makes them useful for generative modeling as their latent spaces are continuous in nature by design which in turn allows easy random sampling and interpolation.



The variation is achieved by doing something like a low variance gradient estimator based on re-parameterization trick i.e. making its encoder not output an encoding vector of size n, rather, outputting two vectors of size n where

a) $\boldsymbol{\mu}$ is a vector of means and

b) $\boldsymbol{\sigma}$ is a vector of standard deviations.

Fig-1.2 Variational Autoencoder Architecture for input reconstruction using re-parameterization trick.

1.3 Hierarchical Variational Autoencoders for Long-Term Memory sequences :

The above explanation tells us that the VAE's are a good option as generative models for continuous data and can reconstruct the inputs given to them better than those produced by the vanilla Autoencoders. However, these VAE's have difficulty in modeling sequences with long-term memory structure. To address this particular issue, the Google Brain Team came up with a novel approach of using a LSTM based Bidirectional Encoder and a Hierarchical Decoder to model where firstly, the outputs are embeddings for subsequences of input and each subsequence is generated independently. This helps in solving the posterior collapse problem faced by the VAE's while modeling long-term continuous structures. The below image shows the architecture of the original LSTM based Hierarchical VAE by the Google Brain Team for generation of Music-VAE paper. The decoder has a hierarchical structure for better sampling and reconstruction of long sequences, this architecture will be discussed in detail further in this report.

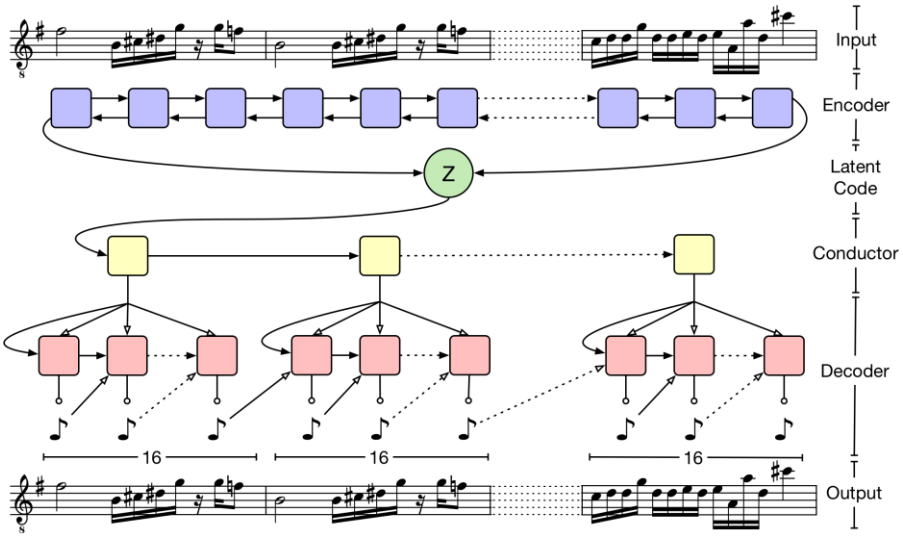


Fig-1.3 Hierarchical Variational Autoencoder Architecture with LSTM Based Bi-directional Encoder and Hierarchial Decoder for Music Generation.

1.4 LSTM Based Bidirectional Encoder and Hierarchical Decoder :

The main issue the traditional LSTM's face for long sequences such as music are majorly due to the posterior collapse problem. The posterior collapse problem can be said to be the vanishing of the latent space as output sequence is generated. To solve this problem the Google Brain Team came up with a novel approach of using a Hierarchical Decoder where the latent vectors which are sampled are passed through multiple levels of decoder rather than a single flat decoder.

1.5 LSTM Based Bidirectional Encoder and Hierarchical Decoder :

1.5.1 Bidirectional Encoder :

For the encoder $\mathbf{q_\lambda = (z|x)}$, a two-layer bidirectional LSTM network is used where the processing of input sequence $\mathbf{x = \{x_1, x_2, x_3, \ldots\ldots, x_T\}}$ to obtain the final state of the vectors in the second bidirectional LSTM layer which are then concatenated to produce h_T and fed into fully connected layers to produce the latent distribution parameters μ and σ given by below equations.

$$\mu = W_{h\mu} h_T + b_\mu$$
$$\sigma = \log\left(\exp(W_{h\sigma} h_T + b_\sigma) + 1\right)$$

where $W_{h\mu}$ $W_{h\sigma}$ and b_{μ} b_{σ} are weight matrices and bias vectors, respectively. Here the LSTM state size is of 2048 for all layers and 512 latent dimensions. The use of a bidirectional recurrent encoder ideally gives the parametrization of the latent distribution longer-term context about the input sequence.

1.5.2 Hierarchical Decoder :

It was found by the Google Brain Team in the preliminary experiments that using a simple RNN as the decoder resulted in poor sampling and reconstruction for long sequences and this caused the vanishing influence of the latent state as the output sequence is generated. To rectify this issue, they proposed a novel hierarchical RNN for the decoder. Assuming that the input sequence and target output sequence *x* can be segmented into *U* non-overlapping subsequences $\mathbf{y_u}$ with endpoints $\mathbf{i_u}$ so that using the below equations we can represent latent vectors,

$$y_u = \{x_{i_u}, x_{i_u+1}, x_{i_u+2}, \ldots, x_{i_{u+1}-1}\}$$
$$\rightarrow \mathbf{x} = \{y_1, y_2, \ldots, y_U\}$$

where they defined special case of $\mathbf{i_{U+1} = T}$. So that the latent vector z could be passed through a fully-connected layer followed by a ***tanh*** activation to get the initial state of a "conductor" RNN. The conductor RNN produces ***U*** embedding vectors $\mathbf{c = \{c_1, c_2, \ldots. c_U\}}$, one for each subsequence using a two-layer unidirectional LSTM for the conductor with a hidden state size of 1024 and 512 output dimensions.
When the conductor produces the sequence of embedding vectors c, each one is individually passed through a shared fully-connected layer followed by a ***tanh*** activation to produce initial states for a final bottom-layer decoder RNN. The decoder RNN then autoregressively produces a sequence of distributions over output tokens for each subsequence $\mathbf{y_u}$ via a *softmax* output layer. At each step of the bottom-level decoder, the current conductor embedding $\mathbf{c_u}$ is concatenated with the previous output token to be used as the input by using a 2-layer LSTM with 1024 units per layer for the decoder RNN.

2 Music-VAE Approach :

2.1 Some Basic Definitions from Music-VAE Project w.r.t. MUSIC :

2.1.1 Bars, Loops and Melodies :

a) Bar : 1) According to the Music Theory a bar is a way of organizing the written **music in** small sections. 2) Each **bar** is a small amount of time which can be felt with the metronome ticks for measurement. 3) Each **bar** usually has the same number of beats in it as shown in below figure having beats 1-**2**-3-4-1-**2**-3-4 divided into **bars** with four beats **music in** each **bar**.

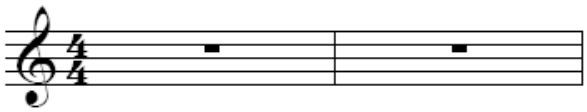


Fig 2.1.1 2-Bar pallet sheet for music.

b) Loop : 1) Consider any drum beat of 2-bar music, if we join the start and the end of the same beat we can form a loop i.e. continuous repeating of the same beat for a rewuired time snippet in Music. 2) Thus a loop is technically a short recording of multiple drum materials which has been edited to loop seamlessly, a drum loop repeats until an exact duration is satisfied.

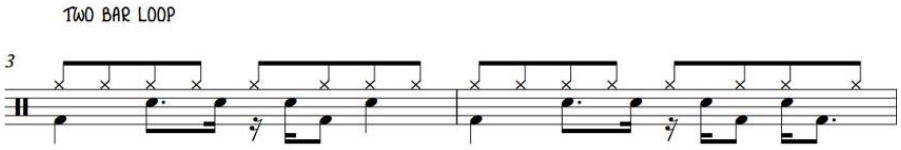


Fig 2.1.1 2-Bar drum beat loop.

c) Melody : 1) A melody can be as simple as a piece of music for one instrument or it can be a bit more complex with a few instruments playing simultaneously to make music. Consider the below examples for music such as in i) we have a simple one line of music piece and from ii) we can see that a melody is generated just like in i) but by using multiple instruments.

i) Single Instrument Music Melody :

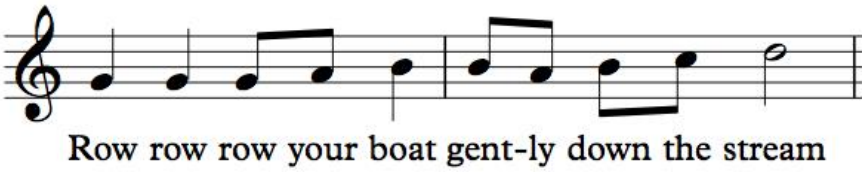


Fig 2.1.1 Simple 2-bar music piece with single instrument.

ii) Multiple Instrument Music Melody :

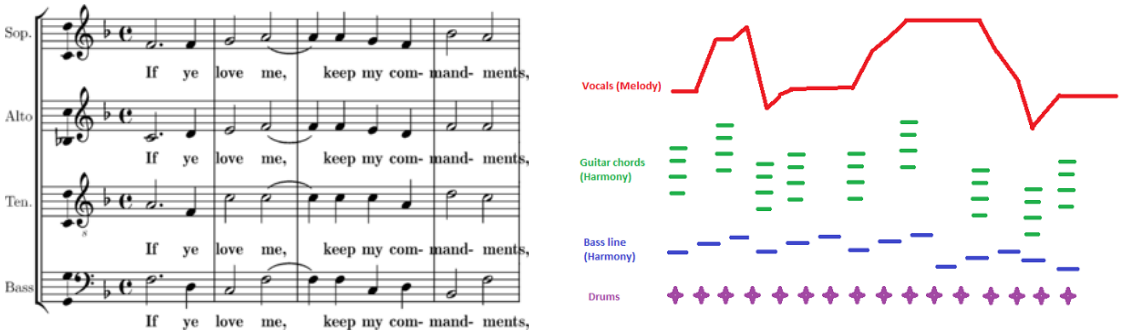


Fig 2.1.1 Complex Loop of 2-bar Music pieces with multiple instruments.

2.2 Music-VAE approach for interpolation of music melodies :

The Google Brain Team's approach to enable interpolations between different musical loops was to separately model 2-bar sequences for melodies and drum beats. The melody loops were represented as a sequence of 32 categorical variables taking one of 130 discrete states per sixteenth note: 128 note-on pitches, a hold state, and a rest state. Drums were represented as 32 categorical variables taking one of 512 discrete states per sixteenth note, representing each of the possible combinations between 9 types of hits: kick, snare, closed hi-hat, open hi-hat, low tom, mid tom, high tom, crash cymbal, and ride cymbal. For each of these they carried out training of a variational autoencoder with a $N_{256}(0, I)$ prior having a bidirectional LSTM encoder and 3-layer LSTM decoder, they achieved a near-perfect reconstruction with low KL divergence. Also it was found that using scheduled sampling during training significantly improved reconstruction accuracy and sample quality.

After successfully applying the VAE's to short loops they moved to the next stage i.e. lengthier melodic sequences that demonstrate aspects of long-term structure. It was found that the simple LSTM decoder they used to model loops did not scale well to 32-bar sequences, likely due to the limited context of the LSTM and the difficulty in parsing note-level detail directly from a latent code *z* representing 512 steps. The solution for this was a novel recurrent decoder with hierarchical structure where the first level had an LSTM was initialized with *z* and sequentially output 16 embeddings. These embeddings were then used to initialize a lower-level LSTM that autoregressively produced 512 sixteenth notes (32 bars). After every 32 outputs (2 bars), the state of the LSTM is reset using the next embedding from the first-level model. Furthermore, these embeddings are concatenated with the output at each step to provide the next input.
For the final step the Google Brain Team attempted to model the relationship between instruments in a composition. Where they made a small modification to the hierarchical decoder by splitting the second level into 3 separate LSTMs, one each to decode lead, bass, and drum parts with 130-category distribution for the bass and lead as for melody above and concatenate the three instruments at each step as the input to our encoder.

2.3 Music-VAE Data Description :

1) The Music-VAE datasets were built by first searching the web for publicly-available MIDI files and acquiring around 1.5 million unique files. Then the files removed those were identified as having a non-4/4-time signature and used the encoded tempo to determine bar boundaries, quantizing to 16 notes per bar.
2) For the 2-bar drum patterns a 2-bar sliding window was used to extract all unique drum sequences with at most a single bar of consecutive rests, resulting in 3.8 million examples.
3) For 2-bar melodies a 2-bar sliding window to extract all unique monophonic sequences with at most a single bar of consecutive rests, resulting in 28.0 million unique examples.
4) For the trio data, we used a 16-bar sliding window to extract all unique sequences containing an instrument with a program number in the piano, chromatic percussion, organ, or guitar interval, [0, 31], one in the bass interval, [32, 39], and one that is a drum with at most a single bar of consecutive rests in any instrument.
5) For multiple instruments in any of the three categories a cross product was applied to consider all possible combinations with 9.4 million examples.

2.4 Music-VAE Explanation in Detail for Code and Implementation :

2.4.1 Encoder used for Music-VAE :

2.4.1.1 Input Details to the Bi-Directional LSTM Encoder :

So, the Google Brain Teams approach to generate and interpolate samples of music consisted 16 bar music samples which were to be fed to the encoder. These music samples were represented as a 3-D matrix i.e. the encoder had to be passed with three values of inputs such as
[batch_size, max_sequence_length, input_depth]
where,
batch_size : consists of the number of samples which are used for training in this case 512 samples.
max_sequence length : is the maximum sequence length of a sample in this case 16 x 16 = 256
input_depth : the dimensions of each note that is generated at each time step. Also, depth of a note in the MIDI data is 4 i.e. it consists of **[pitch, velocity, start-time, end-time]** as parameters passed to it in encoded form.
Thus, the final input fed to the next phase of the encoder i.e. a Bidirectional LSTM encoder is in the form of [512 x 256 x 4].

2.4.1.2 Bi-Directional LSTM Encoder Details :

The Bi-Directional LSTM encoder is a Recurrent Neural Network (RNN) with two layers. As given by the Google Brain Teams paper it has 2048 hidden nodes where they are fully connected in the

LSTM cell, called a state size of 2048. The input that is fed to the encoder is a tensor which is converted to a time major before feeding to the first layer of the RNN as LSTM's need to preserve the essence of time for Long-Term sequences, the tensor is of dimensions [256 x 512 x 4].

Now, as the output from the first layer of the RNN we get two hidden states in Forward and Backward directions which is then passed to the second layer, the hidden states can be represented as (H_t- fwd) and (H_t- bkwd). Also for a 16 bar melody, $2^{16} = 256$ LSTM's one for each note.
The configuration code snippet can be found on magenta's website in the configs.py file,

https://github.com/tensorflow/magenta/blob/master/magenta/models/music_vae/configs.py/

```
)

CONFIG_MAP['hierdec-mel_16bar'] = Config(
    model=MusicVAE(
        lstm_models.BidirectionalLstmEncoder(),
        lstm_models.HierarchicalLstmDecoder(
            lstm_models.CategoricalLstmDecoder(),
            level_lengths=[16, 16],
            disable_autoregression=True)),
    hparams=merge_hparams(
        lstm_models.get_default_hparams(),
        HParams(
            batch_size=512,
            max_seq_len=256,
            z_size=512,
            enc_rnn_size=[2048, 2048],
            dec_rnn_size=[1024, 1024],
            free_bits=256,
            max_beta=0.2,
        )),
    note_sequence_augmenter=None,
    data_converter=mel_16bar_converter,
    train_examples_path=None,
    eval_examples_path=None,
)
```

The final states from the LSTM layers i.e. H_t- fwd and H_t- bkwd with dimensions [1 x 512 x 4], are then concatenated since the latent features are present and describe the input in both directions. And for music interpolation it is required to have Bi-Directional latent representations. Now, these are passed to two fully connected network layers with a SoftPlus activation function which yield us representations of the posterior distribution of each sequence with the required Multivariate Gaussian Normal distributions **μ** (Mean) and **σ** (Deviation) of 512 dimensions, given by the below equation which is also explained before.

$$\mu = W_{h\mu} h_T + b_\mu$$
$$\sigma = \log\left(\exp(W_{h\sigma} h_T + b_\sigma) + 1\right)$$

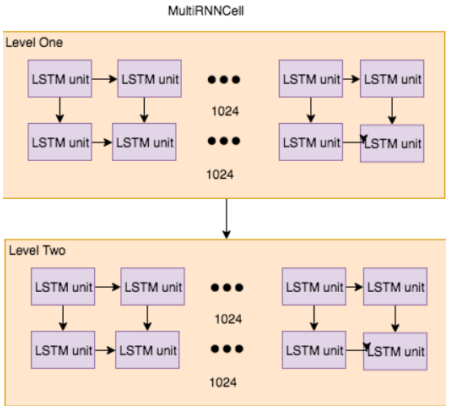
2.4.2 Decoder used for Music-VAE :

2.4.2.1 Hierarchical LSTM Decoder :

In the Hierarchical-LSTM-Decoder phase the paper specifically mentions the use of a conductor

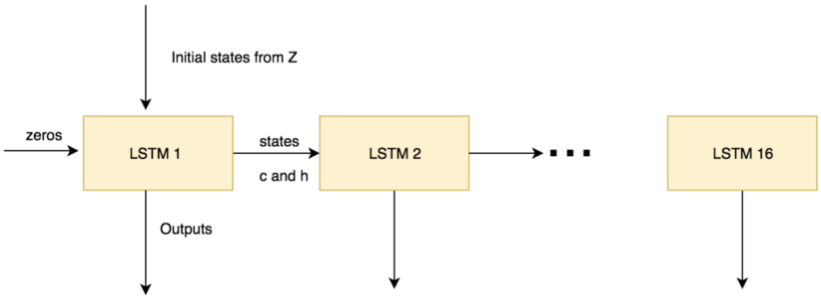
layer made of an LSTM layer i.e. an additional layer for learning Long-Term sequences from the encoded latent space. Varying on the type of music used the number of LSTM blocks in the layer is specified by hyper parameters. A ***tanh*** activation function is used for the fully connected networks used for passing the latent variables obtained from the encoder which is used to initialize the first level of decoder LSTM inputs.

As specified for the encoder layer there is also a length associated with the LSTM-layer in the decoder part. The length of the LSTM layer is set using a hyperparameter which will differ for model to model based on the use / application. Now, a single LSTM block in one layer has stacked LSTM each with 1024 units.



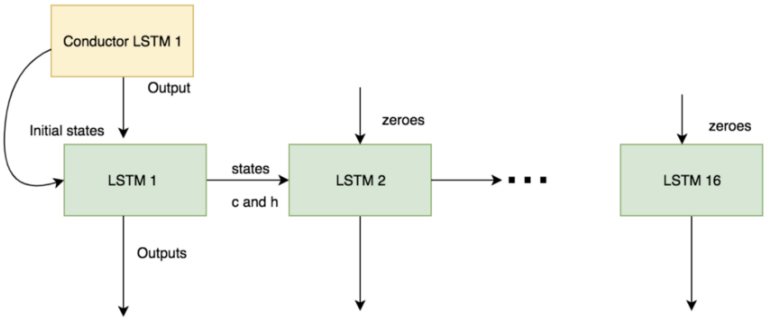
From the above figure we can see that for the two blocks the level_length is equal to [16, 16] i.e. 16 LSTM's in the first block and 16 LSTM's in the second block. Total blocks in layer 2 is 16 x16 = 256. Number of cells inside one LSTM block depends on the **dec_rnn_size**. In the figure shown below there are two stacked 1024 cells inside one block. In other words, a **multirnncell** is composed sequentially of 2 LSTM block cells of size 1024 each with the second layer connected to the output LSTM layer.

At the first level of unidirectional LSTM layer the additional layer i.e. the conductor is located. Only the first block cell in the conductor is initialized with the embeddings from latent space setting the hidden and cell state initializing the input to zeros based on batch size of the samples which is also set as a hyperparameter. Once the input is given the output of first cell provides the next cell state, hidden state and the output and these values are pass to the next LSTM and so on so forth till the LSTM 16 as shown in the below figure.



From the above figure we can see that the conductor layer has 16 LSTM's with the outputs of each states passed on to the next LSTM. Moreover, the next stage after the conductor stage is the decoder stage where each of the conductor's output becomes the input to the decoder which is again an unidirectional LSTM layer just as in the conductor layer. A hyperparameter specifies that the layer

length to number of blocks to each conductor is created. The states of first block cell becomes the initial state of the next block within the conductor only. There is no communication happening between the LSTM blocks of different conductors.



2.4.2.2 Hierarchical LSTM Decoder Details :

The latent space variable Z is passed to dense layer with activation function as ***tanh***. The output shape is the flattened state size of the first LSTM block in the first Conductor. If the block has 1024 + 1024 LSTM units as mentioned previously the state size is c = 1024, h = 1024 for each of the two stacked LSTM's with flattened state size [1024, 1024, 1024, 1024]. Also, the shape of the output of the dense layer is 4096 i.e. sum of the state size having a shape of 512 x 4096.

After the dense layer the sequence is split into four flattened state sizes with shape 512 x 1024 packed as tuples c and h, where c is the cell state and h is the hidden state of the LSTM in which they are packed. These packed tuples are then given to the first conductor for depth based decoding and finally passing it to the decoder stage. The states from decoder 1 is passed as initial states to decoder 2 to provide the outputs, also when conductor 1 has completed the decoding, the same procedure starts for conductor 2, 3 until the end of level_length = 16.

Each conductor decodes recursively by depth first and the conductors are independent of the base decoder outputs. This forces the base decoder to utilize the latent vectors and thereby helping to learn longer sequences. The final output stage is where the output from all the decoders are then merged to give one single output with the reconstruction loss calculated by comparing the final output with the actual output. Also, the concatenated decoder output is the interpolated music sequence which will be the output of the final decoder stage.

The configuration code snippet can be found on magenta's website in the configs.py file,

https://github.com/tensorflow/magenta/blob/master/magenta/models/music_vae/configs.py/

```
CONFIG_MAP['hier-trio_16bar'] = Config(
    model=MusicVAE(
        lstm_models.HierarchicalLstmEncoder(
            lstm_models.BidirectionalLstmEncoder, [16, 16]),
        lstm_models.HierarchicalLstmDecoder(
            lstm_models.SplitMultiOutLstmDecoder(
                core_decoders=[
                    lstm_models.CategoricalLstmDecoder(),
                    lstm_models.CategoricalLstmDecoder(),
                    lstm_models.CategoricalLstmDecoder()],
                output_depths=[
                    90,  # melody
                    90,  # bass
                    512,  # drums
                ]),
            level_lengths=[16, 16],
            disable_autoregression=True)),
    hparams=merge_hparams(
        lstm_models.get_default_hparams(),
        HParams(
            batch_size=256,
            max_seq_len=256,
            z_size=512,
            enc_rnn_size=[1024],
            dec_rnn_size=[1024, 1024],
            free_bits=256,
            max_beta=0.2,
        )),
    note_sequence_augmenter=None,
    data_converter=trio_16bar_converter,
    train_examples_path=None,
    eval_examples_path=None,
)
```

2.5 Music-VAE Results :

2.5.1 Results for reconstruction of inputs :

	Teacher-Forcing		Sampling	
Model	Flat	Hierarchical	Flat	Hierarchical
16-bar Melody	0.952	**0.956**	0.685	**0.867**
16-bar Drum	0.937	**0.955**	0.794	**0.908**
Trio (Melody)	0.866	**0.868**	0.660	**0.760**
Trio (Bass)	0.906	**0.912**	0.651	**0.782**
Trio (Drums)	0.943	**0.946**	0.641	**0.895**

Table 2. Reconstruction accuracies for the Lakh MIDI Dataset calculated both with teacher-forcing (i.e., next-step prediction) and full sampling. All values are reported on a held-out test set. A softmax temperature of 1.0 was used in all cases, meaning we sampled directly from the logits.

Fig. 2.4.1 Reconstruction Accuracies for Music-VAE Dataset.

2.5.2 Results for interpolation of inputs :

Figure 14. Example interpolation in the 2-bar melody MusicVAE latent space. Vertical axis is pitch (from A_3 to C_8) and horizontal axis is time. We sampled 6 interpolated sequences between two test-set sequences on the left and right ends. Each 2-bar sample is shown with a different background color. Audio of an extended, 13-step interpolation between these sequences is available in the online supplement.[5]

Fig. 2.4.2 Interpolation of 2-bar melodies.

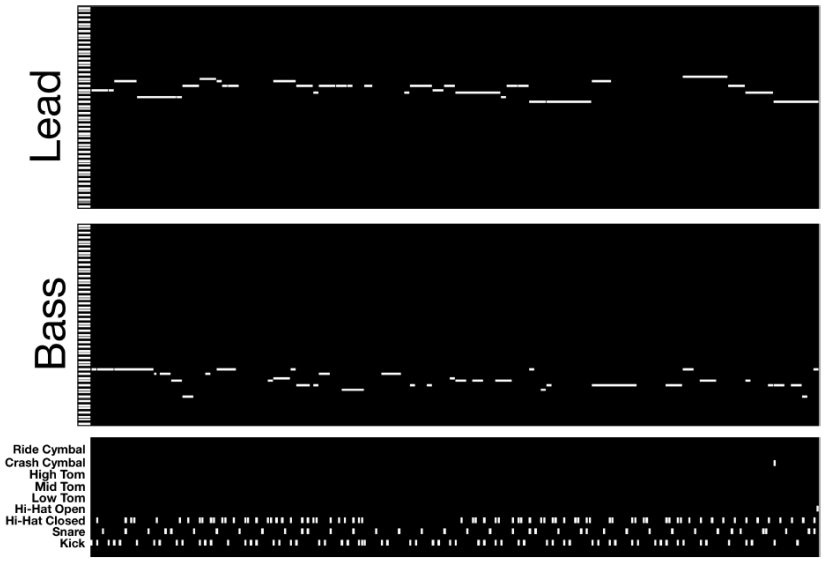


Figure 15. Selected example 16-bar trio sample generated by MusicVAE. Audio for this and other samples is available in the online supplement.[5]

Fig. 2.4.2 Interpolation of 2-bar melodies with Lead, Bass and Drum beats.

2.6 Conclusion for Music-VAE :

Thus, from the all the above discussions and explanations about the Google Brain Team's novel approach to solve long-term sequence modeling we can conclude that they successfully devised an approach to model long-term sequences with significant and mesmerizing results. The paper was indeed very interesting to read, very difficult to understand and the coding techniques used were highly sophisticated and hard to decode.

It was very enlightening to know about such researches being conducted in the fields of Machine Learning and Deep Learning. Also, the Hierarchical-LSTM-Autoencoder proves itself to be a better model for analyzing long-term sequence datasets. Thus, this provides us with a platform to further understand different uses of such models and architectures to perform various experiments of our own. There is also available the link to Google Colab Notebook of this project for enthusiast to explore what the model can do if tuned for various hyperparameters. Below is the link to the same :

https://colab.research.google.com/notebooks/magenta/music_vae/music_vae.ipynb/

3 Stock Market Analysis Using Various Techniques :

3.1 What is a Stock Market :

A **stock market** is the aggregation of buyers and sellers (a loose network of economic transactions, not a physical facility or discrete entity) of stocks (also called shares), which represent ownership claims on businesses; these may include *securities* listed on a public stock exchange, as well as stock that is only traded privately.

3.2 Analysis of Stock Market :

Stock analysis is the evaluation of a particular trading instrument, an investment sector, or the market as a whole. Stock analysts attempt to determine the future activity of an instrument, sector, or market based on the trends in fluctuation of prices of such instruments/stocks/shares.

3.3 Stock Market and Machine Learning :

As the use of Machine Learning and Deep Learning techniques is growing exponentially and reaching into various fields of not only science but also medicine, arts, music, literature, speech, finance, security etc. the exposure to analysis of such field has been increased. Moreover, Data Scientists and Computer Scientists enthusiastic to learn some other fields have started showing interest in such fields and try to apply their knowledge to develop different techniques to analyze such areas. One such application is the Machine Leaning and Deep Learning based Stock Market Analysis for predicting how the market will perform. But predicting how the stock market will perform is one of the most difficult things to do as there are so many factors involved in the prediction of the behavior of the market. Furthermore, not only physical factors and physiological factors but also rational and irrational factors contribute to the behavior of the stock market fluctuation. All these aspects combine to make share prices volatile and very difficult to predict with a high degree of accuracy.
Many Data Scientists and Computer Scientists have taken it as a challenge to answer the question of *"Can we use machine learning as a game changer in this domain?"* To answer this question the analysis of using features like the latest announcements about an organization, their quarterly revenue results, trading strategies etc. have been analyzed and basic concepts of trading have been set as fundamentals for the same. Moreover, the data analysts and computer scientists believe that machine learning techniques have the potential to unearth patterns and insights that haven't been seen or observed before and these can be used to make unerringly accurate predictions.

3.4 Some Details about Stock Market Analysis :

The study of Stock Market can be broadly classified into two main parts : a) Fundamental Analysis and b) Technical Analysis.

a) Fundamental Analysis – This involves analyzing the company's future profitability on the basis of its current business environment and financial performance.

b) Technical Analysis – This on the other hand, includes reading the charts and using statistical figures to identify the trends in the stock market.

The main focus of this study will be technical analysis, as we will be trying to analyze the stock market fluctuations using statistical data available to us.

Consider the above figure in which we have the details of a stock market data for one single company, in our case "APPLE". As we all know Apple is a Software and Technology giant with their products not only ranging from cellular phones but also other consumable electronics and many other fields. We can briefly introduce Apple as Apple Inc. as an American multinational technology

company headquartered in Cupertino, California, that designs, develops, and sells consumer electronics, computer software, and online services.

Before we start with our analysis it is important that we get to know certain technicalities as to what is included in the statistical data and what are we looking at from a broader perspective. The standard stock market data set consists of multiple variables in the dataset such as date, open, high, low, last, close, adjusted close, and volume etc.

```
In [5]: stock_data.head()
```

Out[5]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2013-01-02	79.12	79.29	77.38	78.43	55.68	140129500
1	2013-01-03	78.27	78.52	77.29	77.44	54.98	88241300
2	2013-01-04	76.71	76.95	75.12	75.29	53.45	148583400
3	2013-01-07	74.57	75.61	73.60	74.84	53.14	121039100
4	2013-01-08	75.60	75.98	74.46	75.04	53.28	114676800

Fig. 3.4 Stock Market Statistical Data with details of price variations.

i) The columns *Open* and *Close* represent the starting and final price at which the stock is traded on a particular day.

ii) *High*, *Low* represent the maximum, minimum of the share for the day.

iii) *Adj Close* is the price of a stock's closing price to accurately reflect that stock's value after accounting for any corporate actions. It is considered to be the true price of that stock and is often used when examining historical returns or performing a detailed analysis of historical returns.

iv) *Volume* is the number of shares or contracts traded in a security or an entire market during a given period of time. For every buyer, there is a seller, and each transaction contributes to the count of total volume. That is, when buyers and sellers agree to make a transaction at a certain price, it is considered one transaction. If only five transactions occur in a day, the volume for the day is five.

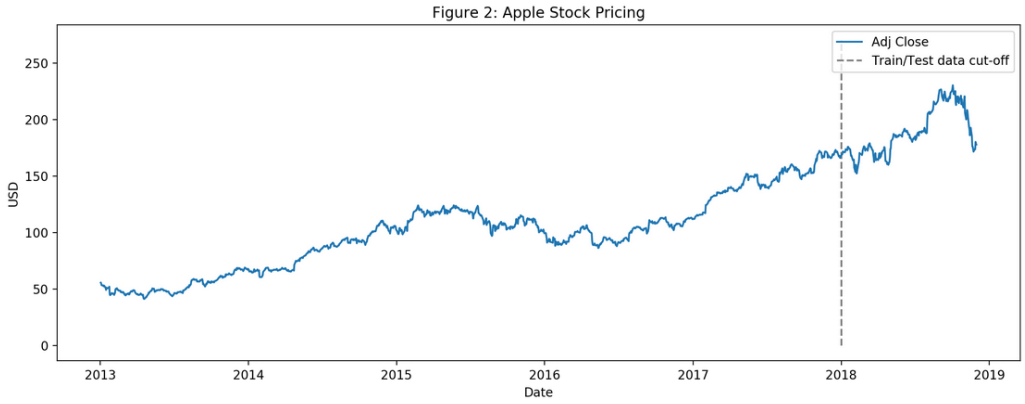


Fig. 3.4 Apple Stock Market Price Chart Based on Adjusted Closing price fluctuation from Date 1st January 2013 to 1st January 2018

The profit or loss calculation is usually determined by the closing price of a stock for the day, hence we will consider the closing price as the target variable. The above plot shows the target variable to understand how it's shaping up in our data.

3.4 Some of the approaches used for Stock Market Analysis :

3.4.1 Moving Average :

The most easiest and firstly used approach to analyze the Stock Market Datasets was 'Average' which very common in finance and commerce sector. Say for example we can use the example of calculating the average marks to determine overall performance of student in exams, or analyzing the weather based on the average temperature range in the past few days to get an idea about current day's temperature. Keeping this in mind the use of calculating 'Average' proved to be a good starting point to analyze the Stock Market dataset based on the past average of prices for making predictions of the future stock pricing.
The most important factor in the stock market analysis is the closing price of any stock for the same day. For the experiment to be performed for predicting prices the closing price for each day was considered to be the average of a set of previously observed values.
But since this is a Time-Series data we cannot simply use the average of the prices so instead it is beneficial to use the moving average technique. The 'Moving Average' technique uses the latest set of values for each prediction from the given dataset i.e. for each subsequent step, the predicted values are taken into consideration while removing the oldest observed value from the set.

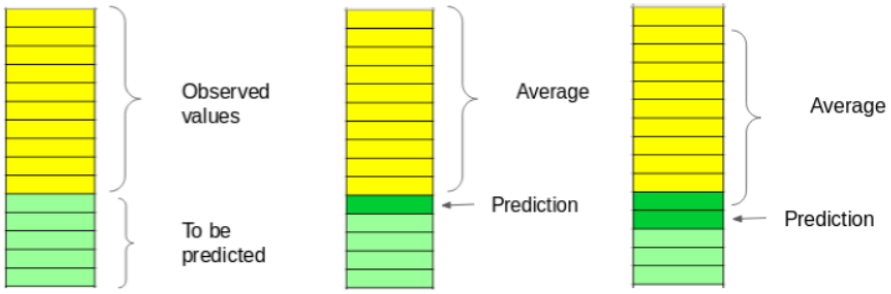


Fig. 3.4.1 'Moving-Average' for Time-Series Prediction

As we can see from the above figure we can see in the first bar the observed values and the values to be predicted values are considered, and that's the hypothesis. In the second bar we can see that for the first step the predicted value is shown with respect to the average. In the third bar we can see that the recent predicted value is taken into consideration and the arrow jumps to the newest value as it removes the oldest observed value from the set and bases its prediction on the newest value.
For the 'Moving-Average' we will need the dates and the closing price of the stock for which we will create a dataframe containing only the Date and Close price columns, then split it into train and validation sets to verify our predictions.

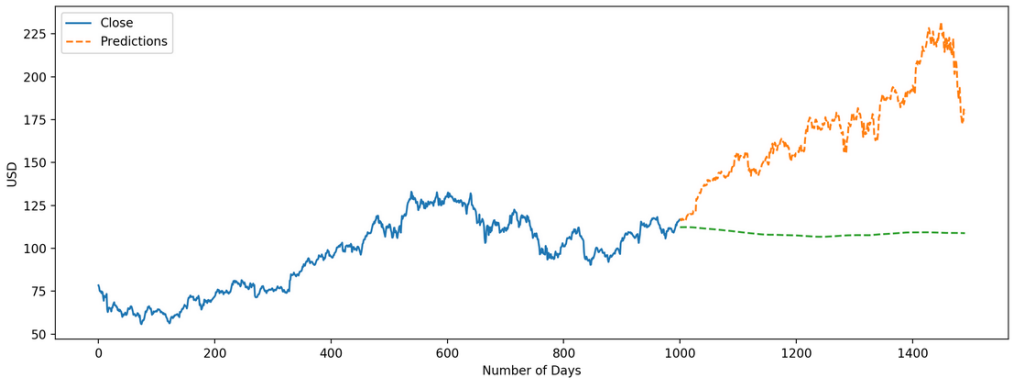


Fig. 3.4.1 'Moving-Average' Plot based on the Predicted Values for Stock Predictions

3.4.2 Linear Regression :

Linear Regression is the most basic algorithm that is used to test any data, similarly there have been attempts at using this algorithm on time series data to analyze the depth of features which should be used for a more better result. The linear regression model is a very basic of machine learning algorithms and it returns an equation that determines the relationship between the independent variables and the dependent variable showcasing their linearity with each other.

The equation for linear regression can be written as below :

$$Y = b_0 + b_1x_1 + b_2x_2 + ... + b_nx_n$$

Where Y = Dependent Variable(DV)

X_1, x_2, x_n – Indepdent Variable(IV)

b_0 – intercept

b_1, b_2 – coefficients

n – No. of observations

Consider the below plot from the Linear Regression performed on the Stock Market Dataset for further explanation of performance and efficiency.

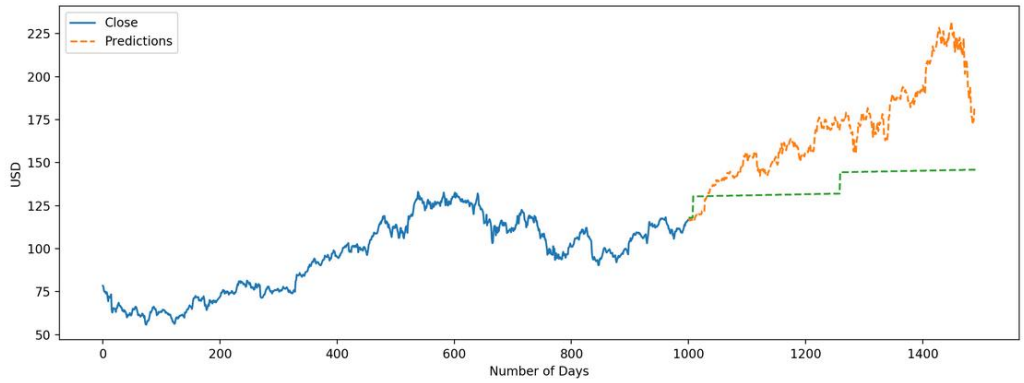


Fig. 3.4.2 Linear Regression Plot based on the Predicted Values for Stock Predictions

3.4.3 ARIMA :

Recently ARIMA (Autoregressive Integrated Moving Average Model) has been very popular statistical method for time series forecasting using machine learning. This acronym is descriptive, capturing the key aspects of the model itself. Briefly, the key aspect are as below :
i) **AR**: *Autoregression* – Uses the dependent relationship between an observation and some number of lagged observations.
ii) **I**: *Integrated* – Uses differencing of raw observations in order to make the time series stationary.
iii) **MA**: *Moving Average* – Uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

An improvement to the basic algorithms such as Linear Regression the ARIMA models take into account the past values to predict the future values. There are three important parameters in ARIMA:

- p – past values used for forecasting the next value
- q – past forecast errors used to predict the future values
- d – order of differencing

But as linear regression the ARIMA also has one disadvantage i.e. the parameter tuning which has to be done for ARIMA models consumes a lot of time. But a solution to this is to use the auto tune ARIMA models which automatically selects the best combination of (p,q,d) that provides the least error and using these values, the model might capture the increasing trend in the series. Although the predictions using this technique are far better than that of the previously implemented machine learning models, it was observed that these predictions were still not close to the real values where the model captured a trend in the series, but tends to ignore the seasonality part. An improvisation to ARIMA models was PROPHET models which will be discussed next.

3.4.4 PROPHET :

Another popularly used approach for time series forecasting was PROPHET models, this model was used widely and largely over other techniques since they require a lot of data preprocessing before fitting the model. PROPHET was originally designed by Facebook and made open to public use as a time series forecasting library that requires no data preprocessing and is extremely simple to implement. Since it was published as a library and was meant to be used with ease the input for Prophet was given using a dataframe with two columns: date and target (ds and y).

It was observed that the PROPHET model tries to capture the seasonality in the past data and tends to work well when the dataset is large. But like most time series forecasting techniques, PROPHET tries to capture the trend and seasonality from past data and thus the model usually performs well on time series datasets, but fails to live up to its reputation in particularly stock market datasets and their analysis. This is due to the fact that; stock prices do not have a particular trend or seasonality and is highly dependent on what is currently going on in the market and thus the prices rise and fall.

Hence forecasting techniques like ARIMA, S-ARIMA and PROPHET show a poor performance for this particular problem of Stock Market Analysis. Next we will discuss another advanced machine learning technique designed specifically for Long-Term Sequences i.e. LSTM – Long-Short Term Memory.

3.4.5 K-Nearest Neighbours Classification :

Going From ARIMA to PROPHET we will be coming back to another interesting algorithm in the field of Machine Learning which is used quite often to analyze the relationship between two values

based on its neighbouring values. The kNN based its analysis on the existing independent variables, kNN finds the similarity between new data points and old data points.
To understand how the kNN works we need to look at an example, consider below figure :

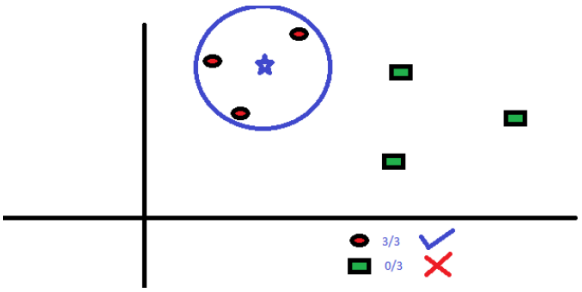


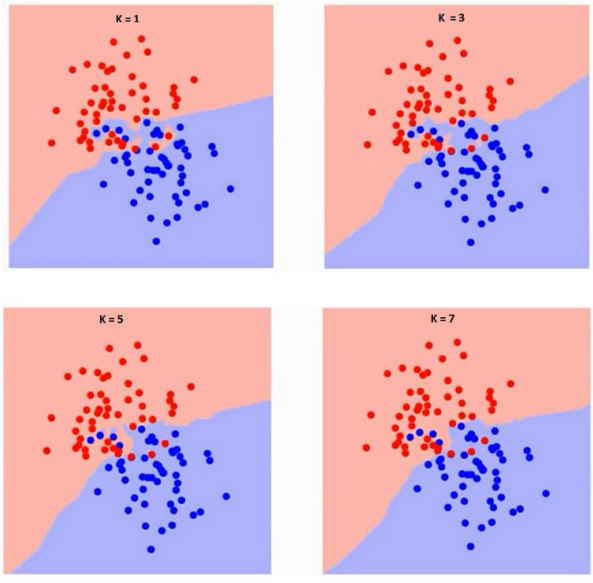
Fig. 3.4.5 kNN Algorithm example for explanation of concept.

Let's take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS).

You intend to find out the class of the blue star (BS). BS can either be RC or GS and nothing else

The "K" is KNN algorithm is the nearest neighbors we wish to take vote from. Let's say K = 3. Hence, we will now make a circle with BS as center just as big as to enclose only three datapoints on the plane. The three closest points to BS is all RC. Hence, with good confidence level we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm.

First let us try to understand what exactly does K influence in the algorithm. If we see the last example, given that all the 6 training observation remain constant, with a given K value we can make boundaries of each class. These boundaries will segregate RC from GS. The same way, let's try to see the effect of value "K" on the class boundaries. Following are the different boundaries separating the two classes with different values of K.



As we can observe from the figure the boundary becomes smoother with increasing value of K. With K increasing to infinity it finally becomes all blue and / or all red depending on the total majority present. The training error rate and the validation error rate are two parameters we need to access on different K-value.

To get the optimal value of K, we can segregate the training and validation from the initial dataset.

Fig. 3.4.5 kNN Clustering and effect of K on the classification process.

Now, consider the below plot for further explanation of the experiment which was performed on Stock Market Data using kNN algorithm.

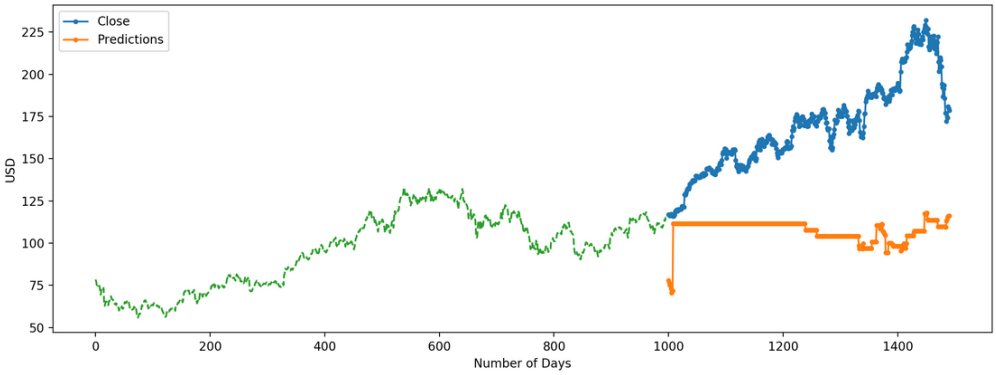
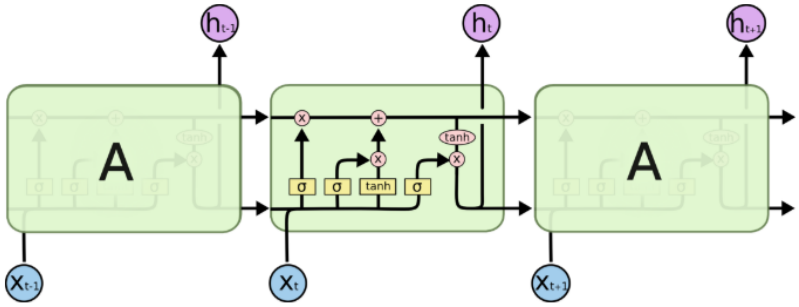


Fig. 3.4.5 kNN Clustering Classification Algorithm for Stock Market Prediction.

3.4.6 LSTM's :

Since the introduction of LSTMs they are being widely used for sequence and time series prediction problems and have proven to be extremely effective in predicting near to accurate values for such types of long-term sequences analysis performed. The reason for their popularity is because of an LSTM's unique ability to gather and store all the past information that is important, and forget the information that is irrelevant and not important. Consider the below image for an LSTM architecture where the basic building blocks of an LSTM cell are shown :



The repeating module in an LSTM contains four interacting layers.

Fig. 3.4.4 Long-Short Term Memory Architecture

An LSTM has three gates:

i) The input gate: The input gate adds information to the cell state.
ii) The forget gate: It removes the information that is no longer required by the model.
iii) The output gate: Output Gate at LSTM selects the information to be shown as output.

The activation function of the LSTM gates is often the *tanh* logistic function used for classification. There are connections in and out of the LSTM gates, a few of which are recurrent. The weights of these connections, which need to be learned during training, determine how the gates operate.

After performing the time series analysis many Computer Scientists observed that due to the flexibility of the LSTM model to be tuned for various parameters such as changing the number of LSTM layers, adding dropout value or increasing the number of epochs can produce higher quality results than the previously used approaches to solve the Stock Market Analysis problem for predictions of Prices. Although, the LSTM's perform very well on time series it is important to know that the stock prices are affected by various factors around the company to which the stock belongs and other factors like inflation, merger/demerger of the companies, accidents, incidents etc. with existence of some intangible factors as well which can often be impossible and anomaly to predict beforehand which can affect the whole price of a company's stock positively or negatively.

4 Stock Market Analysis Based on AutoEncoders :

4.1 General Problem with Time-Series Data :

The Time-Series data has always been complex due to the essence of time and the sensitive information that needs to be analyzed. Due to the complexity of the stock market dynamics, stock price data is often filled with noise that might distract the machine learning algorithm from learning the trend and structure. Hence, it is in our interest to remove some of the noise, while preserving the trends and structure in the data.

For this purpose we can use two methods i) Fourier Transform and ii) Wavelet Transforms
in our experiment we have used both Fourier Transforms and Wavelet Transforms to analyze the dynamics of the data structure and the observed that the Wavelet Transforms can be a better choice to preserve the time factor of the data, instead of producing a merely frequency based output.

4.2 Wavelet Transforms :

The wavelet transform is very closely related to the Fourier Transform, just that the function used to transform is different and the way this transformation occurs is also slightly varied.

$$X_\omega(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t)\psi\left(\frac{t-b}{a}\right) dt$$

Fig. 4.2 Equation for Wavelet Transforms for de-noising data

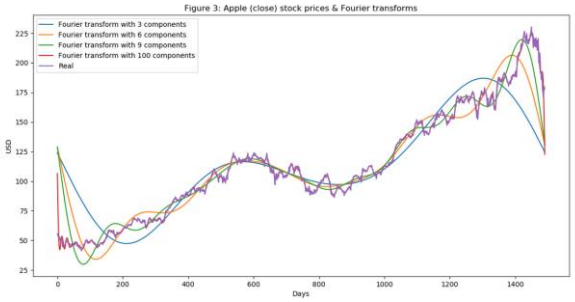


Fig. 4.2 Equation for Fourier Transforms for de-noising data

The process is as follows:

i) The data is transformed using Wavelet transform.
ii) Coefficients that more than a full standard deviation away are removed (out of all the coefficients)
iii) Inverse transform the new coefficients to get the de-noised data.

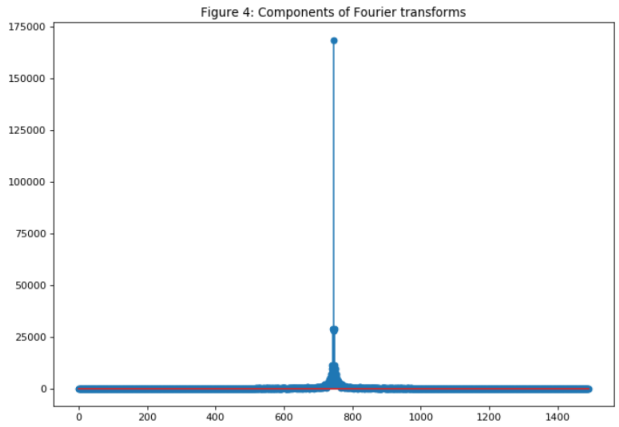
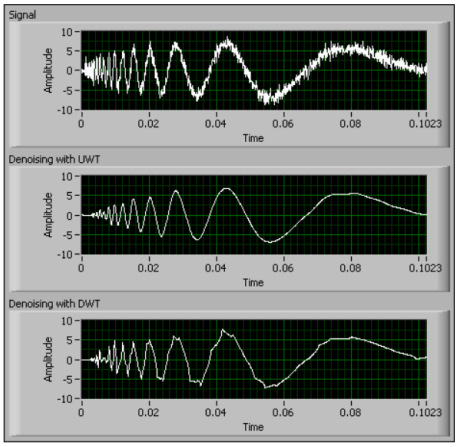


Fig. 4.2 Components of Fourier Transforms for de-noising data

4.3 Wavelet Feature Extraction :

For any analysis a brief domain knowledge and expertise is required, also a huge amount of research for the details which are necessary to implement such complex models. And since due to some limitations we lack that in-depth knowledge it can be achieved by using automated feature extraction using stacked autoencoders or other machine learning algorithms like restricted Boltzmann machines. Since we have that knowledge of autoencoders we can choose to use stacked autoencoders due to the interpretability of the encoding as compared to the probabilities from the restricted Boltzmann machines.



The random noise that was present in the initial signal as shown in the figure is absent in the de-noised versions.
This will help us very much when we process and de-noise the stock market data.
This is exactly what we are looking to do with our stock price data.
We can use the available library **pywt** which is excellent for wavelet transforms which lessens the load tremendously for implementing such tedious formulae.

Fig. 4.3 Wavelet Transforms for de-noising data

4.4 Stacked AutoEncoders :

We know this very well that an autoencoder is very good for compressing data and reproducing it back again. What we are interested in is the compression part, as it means the information required to reproduce the data is in some way encoded in the compressed form. This suggests that these compressed data can be in some way the features of the data that we are trying to extract features out from. The following is the network structure of a stacked autoencoder:

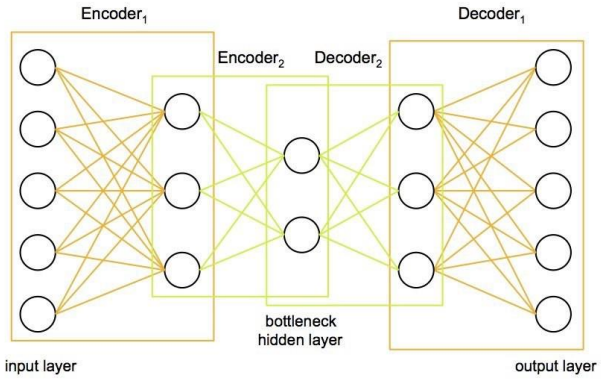


Fig. 4.4 Stacked Autoencoders with Decoders for Data Compression and Recontruction

The input data is compressed into however many neurons desired and the network is forced to rebuild the initial data using the autoencoder. This forces the model to extract key elements of the data, which we can interpret as features. One key thing to note is that this model actually falls under unsupervised learning as there are no input-output pairs, but both input and output is the same.
The training of the autoencoder with the denoised stock price data from 2013 till 2018. After training for 1000 epochs, the RMSE decreases to around 0.9. Then, we will use that model to encode the rest of stock price data into features.

4.5 LSTM's and Autoencoders :

The LSTM model needs no introduction as it has become very widespread and popular in predicting time series. It gets its exceptional predictive ability from the existence of the cell state that allows it to understand and learn longer-term trends in the data, also a detailed explanation of LSTM's is given above with working and architecture.

Optimizer *:*

The type of optimizer used can greatly affect how fast the algorithm converges to the minimum value. Also, it is important that there is some notion of randomness to avoid getting stuck in a local minimum and not reach the global minimum. There are a few great algorithms, but I have chosen to use Adam optimizer. The Adam optimizer combines the perks of two other optimizers: ADAgrad and RMSprop.
The ADAgrad optimizer essentially uses a different learning rate for every parameter and every time step. The reasoning behind ADAgrad is that the parameters that are infrequent must have larger learning rates while parameters that are frequent must have smaller learning rates. In other words, the stochastic gradient descent update for ADAgrad becomes

$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i}$$

Where

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i})$$

The learning rate is calculated based on the past gradients that have been computed for each parameter. Hence,

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t$$

Where G is the matrix of sums of squares of the past gradients. The issue with this optimization is that the learning rates start vanishing very quickly as the iterations increase. RMSprop considers fixing the diminishing learning rate by only using a certain number of previous gradients. The updates become

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t$$

Where

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

Now that we understand how those two optimizers work, we can look into how Adam works. Adaptive Moment Estimation, or Adam, is another method that computes the adaptive learning rates for each parameter by considering the exponentially decaying average of past squared gradients and the exponentially decaying average of past gradients. This can be represented as

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2 \qquad \text{and} \qquad m_t = \beta m_{t-1} + (1-\beta) g_t$$

The v and m can be considered as the estimates of the first and second moment of the gradients respectively, hence getting the name Adaptive Moment Estimation. When this was first used, researchers observed that there was an inherent bias towards 0 and they countered this by using the following estimates:

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \qquad \text{and} \qquad \hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

This leads us to the final gradient update rule

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t$$

This is the optimizer that I used, and the benefits are summarized into the following:

1. The learning rate is different for every parameter and every iteration.
2. The learning does not diminish as with the ADAgrad.
3. The gradient update uses the moments of the distribution of weights, allowing for a more statistically sound descent.

Regularization :

Another important aspect of training the model is making sure the weights do not get too large and start focusing on one data point, hence overfitting. So we should always include a penalty for large weights (the definition of large would be depending on the type of regularizer used). I have chosen to use Tikhonov regularization, which can be thought of as the following minimization problem:

$$\underset{f \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} V(f(x_i), y_i) + \gamma ||f||_K^2$$

The fact that the function space is in a Reproducing Kernel Hilbert Space (RKHS) ensures that the notion of a norm exists. This allows us to encode the notion of the norm into our regularizer.

4.6 LSTM's Autoencoders Model Implementation :

LSTM based Stacked AutoEncoders were found by people after decades of research in the Deep Learning Field. Maybe there are hidden correlations that people cannot comprehend due to the enormous amount of data points, events, assets, charts, etc. The stacked AutoEncoders is a type of neural networks we can use the power of computers and probably find new types of features that affect stock movements.
The stacked AutoEncoders provide us with better encodings at the bottle-neck output and the stacked decoders offer a better reconstruction at the output of the AutoEncoder. Also the LSTM Layers provide with a better learning than the normal AutoEncoders since they tend to learn Long-Term sequences with more precision and can predict the outputs based on all the learnt data.
Below diagram shows Graphs of the Original values Vs Predicted Values using LSTM-Stacked AutoEncoders.

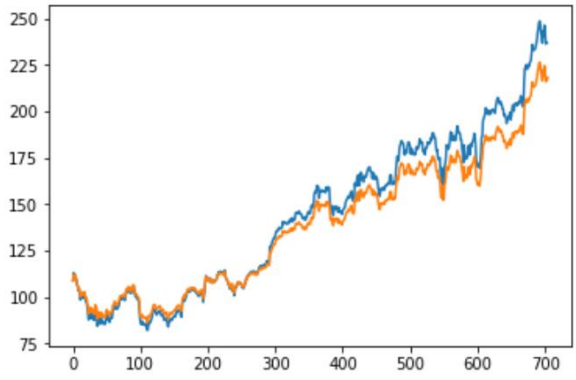


Fig. 4.5 Stacked Autoencoders LSTM for Stock Market Analysis

5 Results :

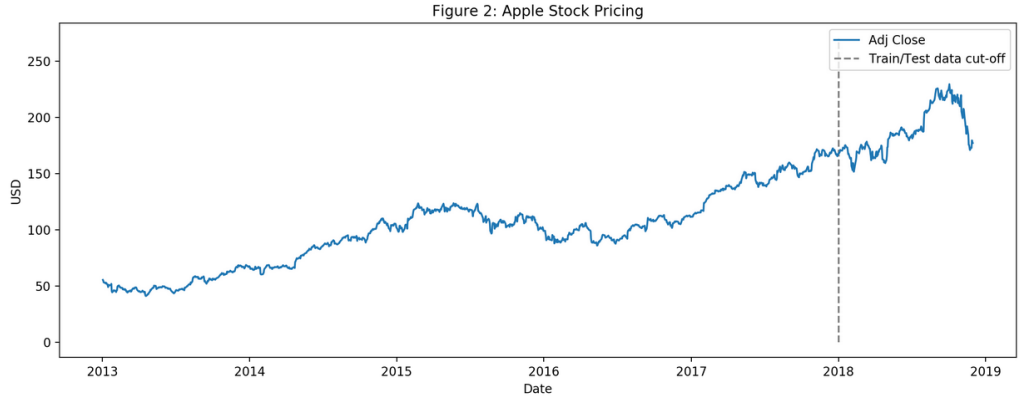
5.1 Stock Data For Apple Inc. :

5.1.1 Stock Data Head For Apple Inc. :

	Date	Open	High	Low	Close	Adj Close	Volume
0	2013-01-02	79.12	79.29	77.38	78.43	55.47	140129500
1	2013-01-03	78.27	78.52	77.29	77.44	54.77	88241300
2	2013-01-04	76.71	76.95	75.12	75.29	53.25	148583400
3	2013-01-07	74.57	75.61	73.60	74.84	52.93	121039100
4	2013-01-08	75.60	75.98	74.46	75.04	53.07	114676800

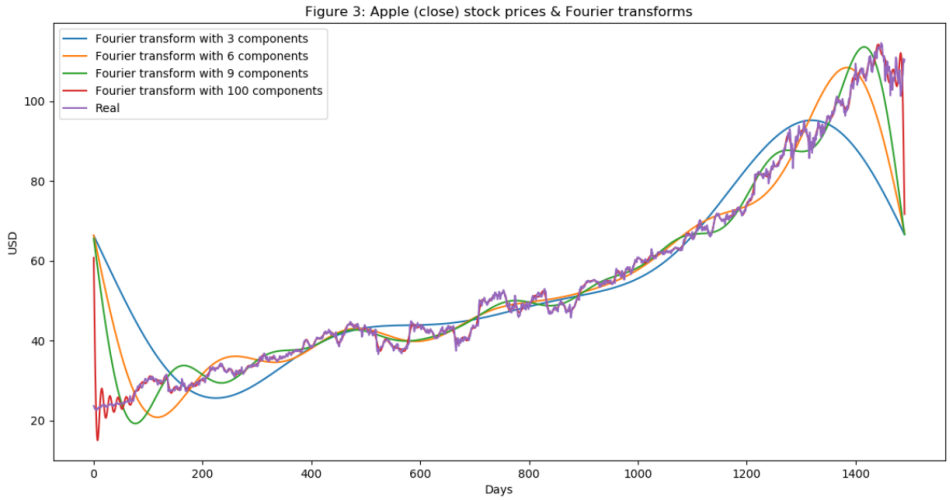
5.1.2 Stock Data Plot Trace For Apple Inc. :

Figure 2: Apple Stock Pricing

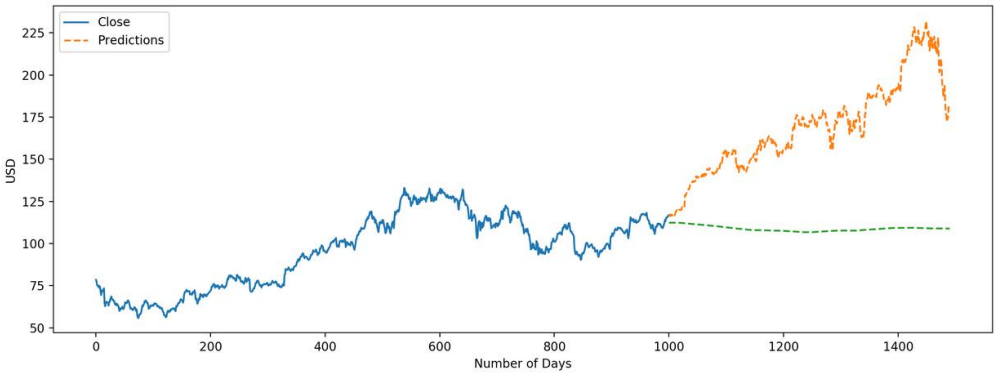


5.1.3 Stock Data Fourier Transforms Plot For Apple Inc. :

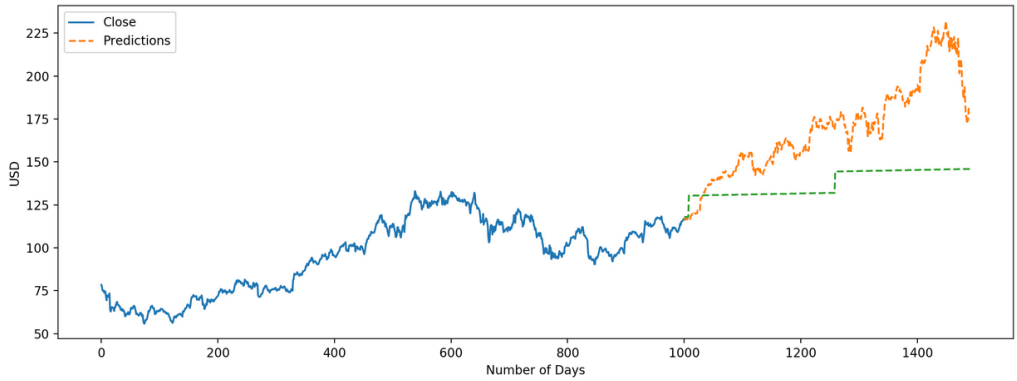
Figure 3: Apple (close) stock prices & Fourier transforms



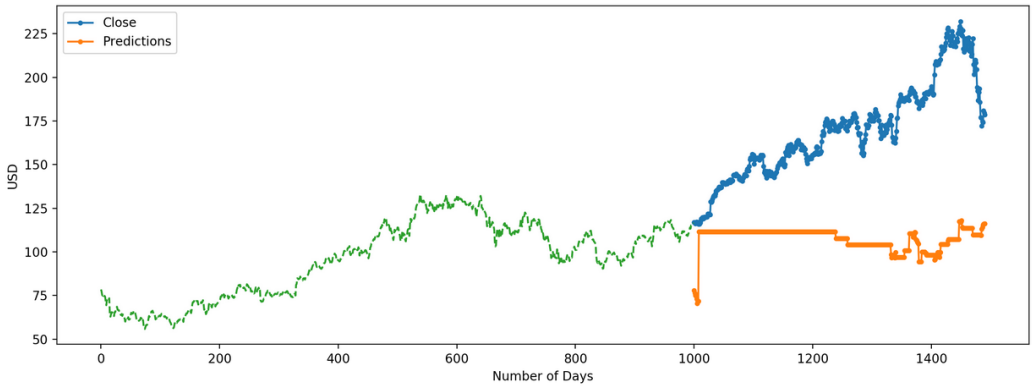
5.1.4 Stock Data Moving Average Plot For Apple Inc. :



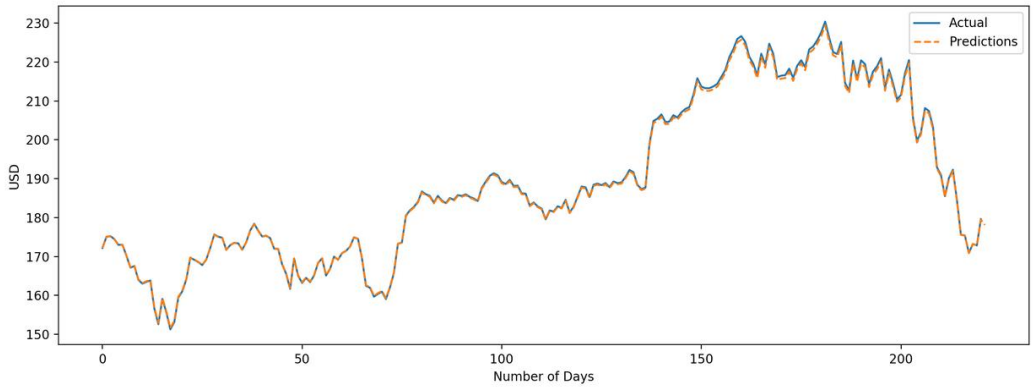
5.1.5 Stock Data Linear Regression Plot For Apple Inc. :



5.1.6 Stock Data kNN Classifier Plot For Apple Inc. :



5.1.7 Stock Data LSTM-Stacked Autoencoders Plot For Apple Inc. :



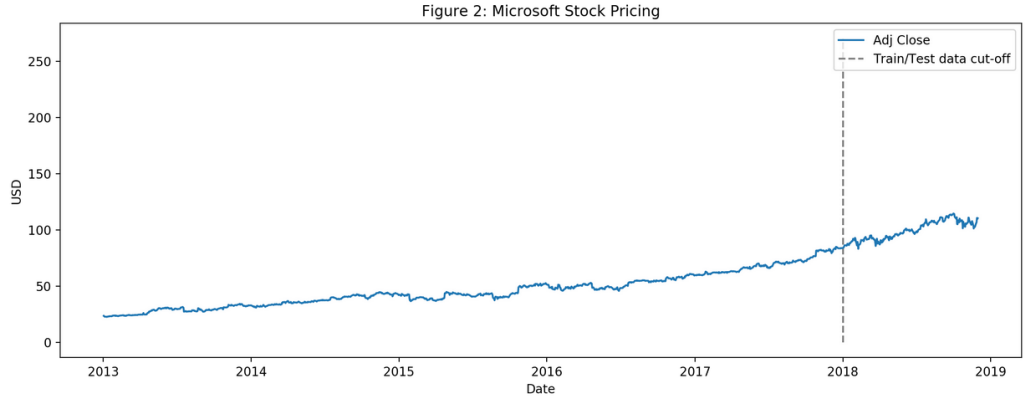
5.2 Stock Data For Microsoft Inc. :

5.2.1 Stock Data Head For Microsoft Inc. :

	Date	Open	High	Low	Close	Adj Close	Volume
0	2013-01-02	27.25	27.73	27.15	27.62	23.67	52899300
1	2013-01-03	27.63	27.65	27.16	27.25	23.35	48294400
2	2013-01-04	27.27	27.34	26.73	26.74	22.92	52521100
3	2013-01-07	26.77	26.88	26.64	26.69	22.87	37110400
4	2013-01-08	26.75	26.79	26.46	26.55	22.75	44703100

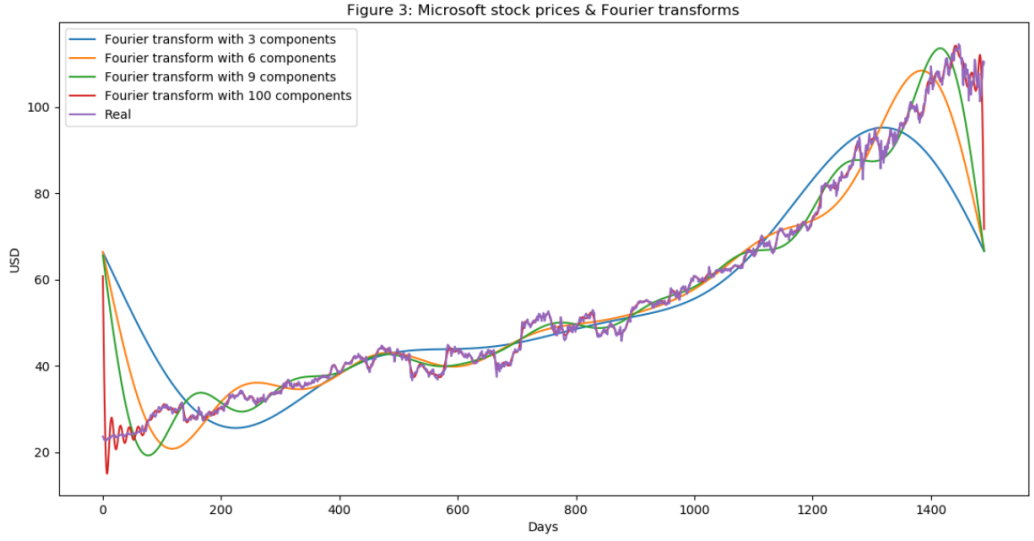
5.2.2 Stock Data Plot Trace For Microsoft Inc. :

Figure 2: Microsoft Stock Pricing

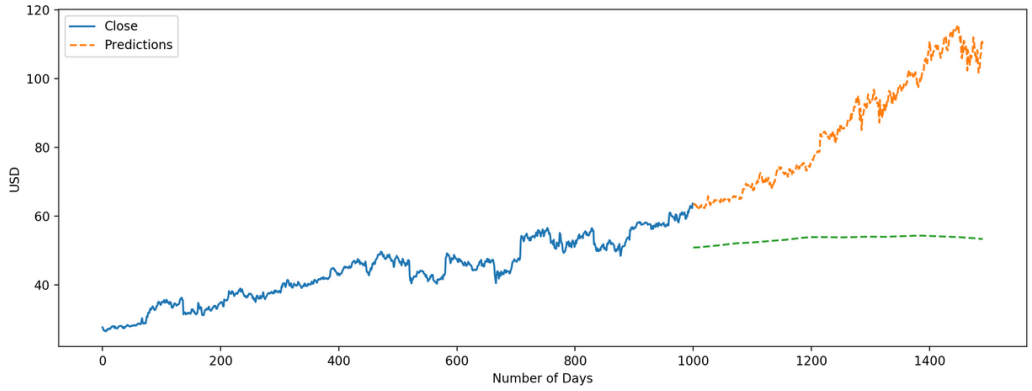


5.2.3 Stock Data Fourier Transforms Plot For Microsoft Inc. :

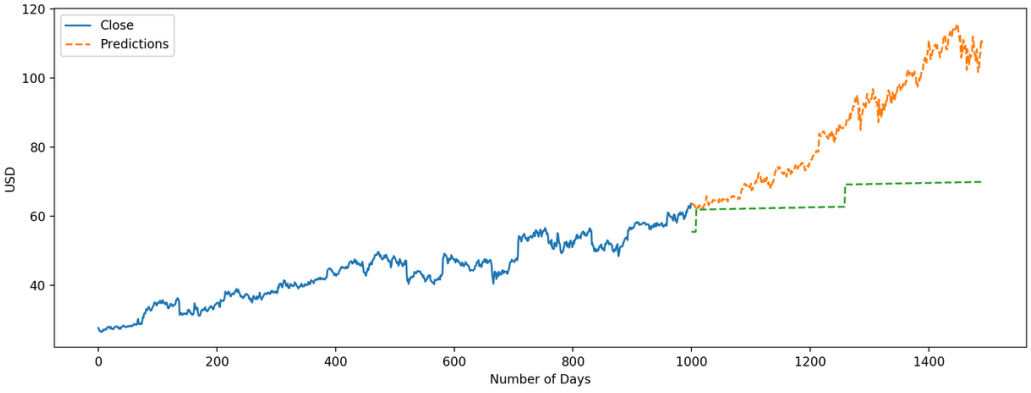
Figure 3: Microsoft stock prices & Fourier transforms



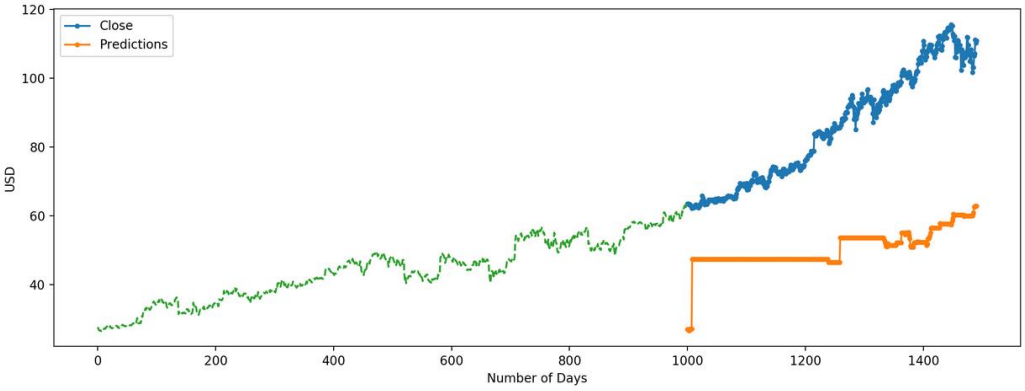
5.2.4 Stock Data Moving Average Plot For Microsoft Inc. :



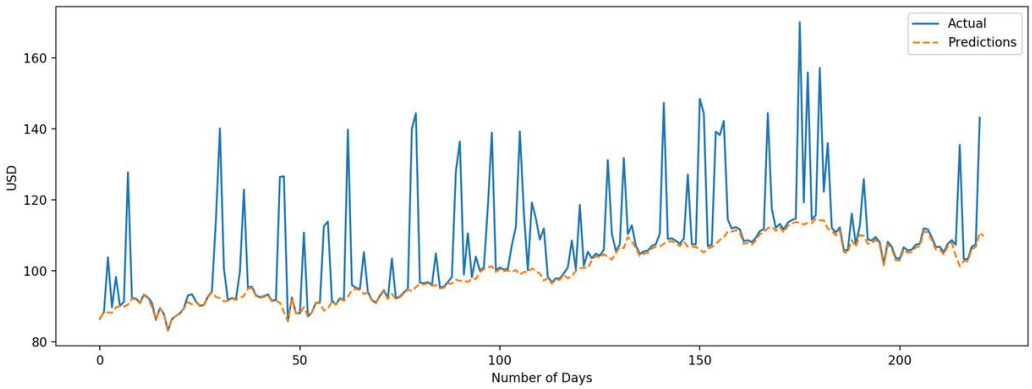
5.2.5 Stock Data Linear Regression Plot For Microsoft Inc. :



5.2.6 Stock Data kNN Classifier Plot For Microsoft Inc. :



5.2.7 Stock Data LSTM-Stacked Autoencoders Plot For Microsoft Inc. :



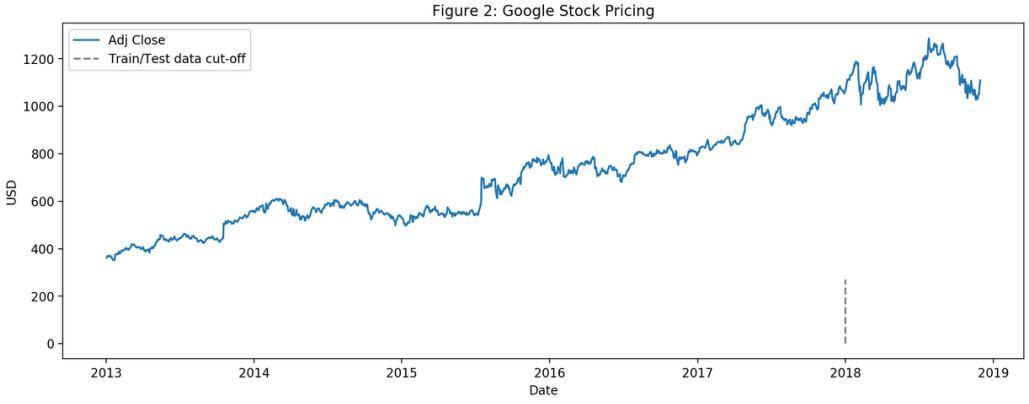
5.3 Stock Data For Google Inc. :

5.3.1 Stock Data Head For Google Inc. :

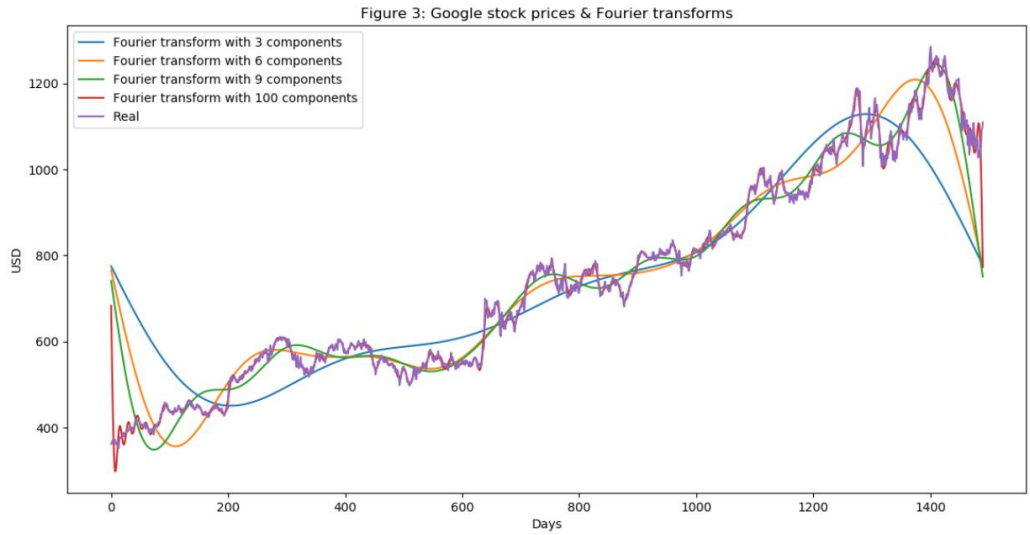
	Date	Open	High	Low	Close	Adj Close	Volume
0	2013-01-02	360.07	363.86	358.63	361.99	361.99	5077500
1	2013-01-03	362.83	366.33	360.72	362.20	362.20	4631700
2	2013-01-04	365.04	371.11	364.20	369.35	369.35	5521400
3	2013-01-07	368.09	370.06	365.66	367.74	367.74	3308000
4	2013-01-08	368.14	368.52	362.58	367.02	367.02	3348800

5.3.2 Stock Data Plot Trace For Google Inc. :

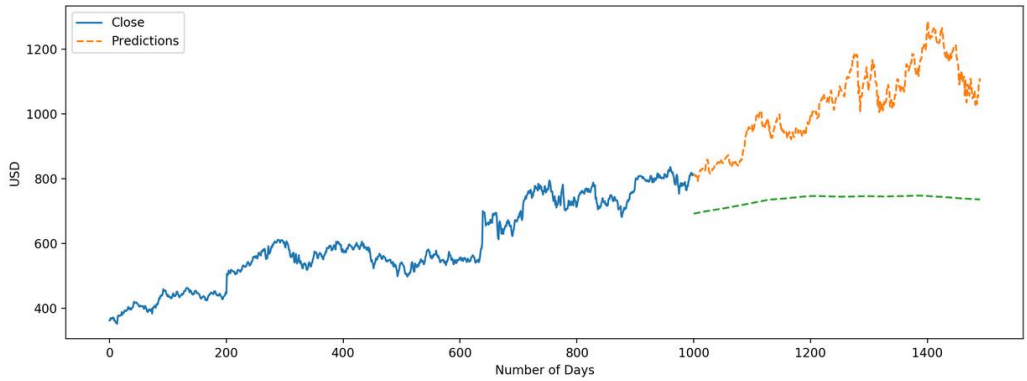
Figure 2: Google Stock Pricing



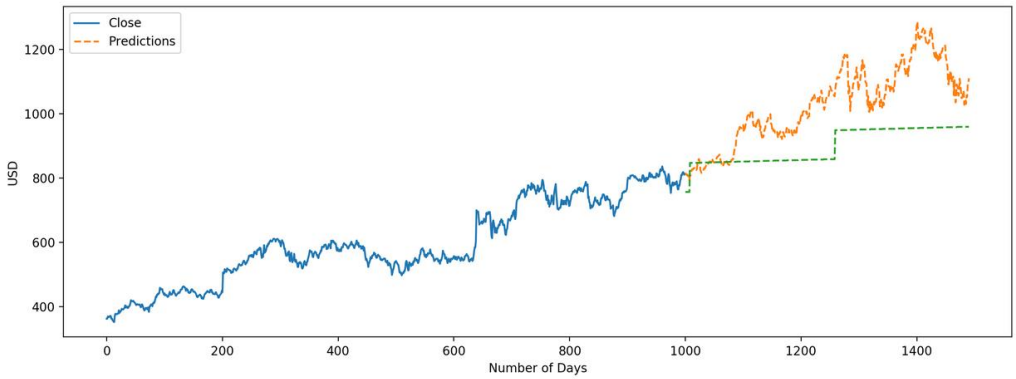
5.3.3 Stock Data Fourier Transforms Plot For Google Inc. :



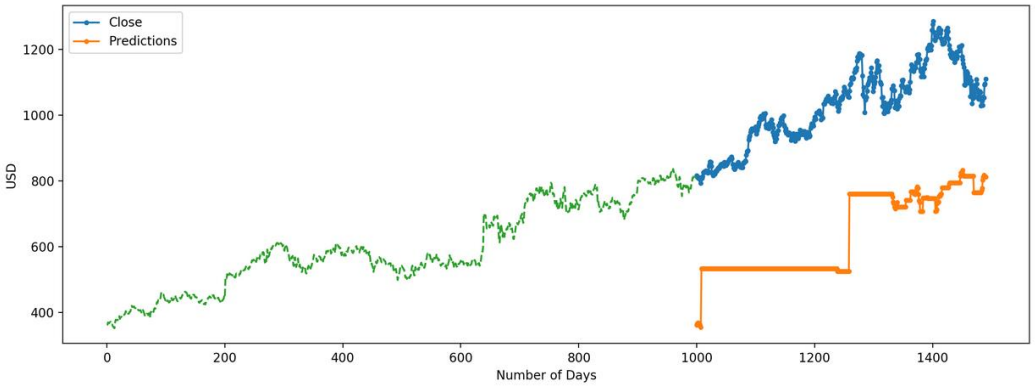
5.3.4 Stock Data Moving Average Plot For Google Inc. :



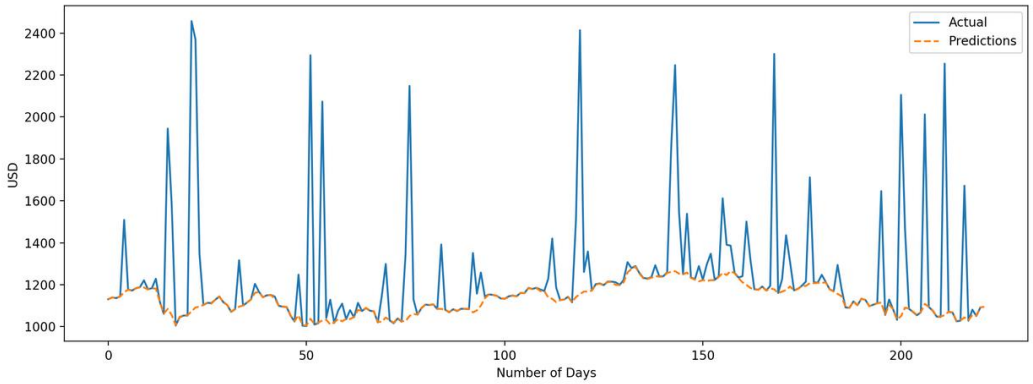
5.3.5 Stock Data Linear Regression Plot For Google Inc. :



5.3.6 Stock Data kNN Classifier Plot For Google Inc. :



5.3.7 Stock Data LSTM-Stacked Autoencoders Plot For Google Inc. :



5.4 Stock Data RMSE Error Observed Values Table :

RMSE Value	Apple Inc.	Microsoft Corporation	Google Inc.
Moving Average	66.47	35.71	311.33
Linear Regression	38.36	23.69	152.096
kNN Classifier	68.09	36.96	395.15

6 General Analysis of the Results for Stock Prices :

6.1 Stock Data Moving Average Analysis :

As we can see from the plots for Moving Average models there is substantial difference between the original values and the predicted values of the results and they are not very promising, the green dotted line shows the shift between the original values of closing price and the predicted values of the closing price. As inferred from the plot the predicted values are of the same range as the observed values in the training set and there is an increasing trend initially and then a slow decrease in the accuracy of predictions which is not desirable. Moreover, the model fails to learn the past trend in the stock pricing of the data and continues predicting resulting in poor accuracy and significant deviation in True values and the predicted values. The Moving average model was one of the first few approaches used to analyze the stock market data, but it fails to do justice due to its limitations in its approach for predicting feature values for the stock prices. This general trend is followed for all the Stock Market Prices we analyzed for Apple, Microsoft and Google.

6.2 Stock Data Linear Regression Analysis :

As we can see from the plots for Linear Regression models we can see if we use regression on the date and month columns to classify the data, the algorithm performs poorly. Linear regression is a simple algorithm and is easy to interpret, but has disadvantages. Using regression algorithms has one problem that the model tends to over-fits to data which has to be classified in this case to the date and month column. Instead of taking into account the previous values from the point of prediction, the model will consider the value from the same date a month ago, or the same date/month a year ago. Thus, we need to improve the method we use for classification of such sensitive datasets with respect to the Time-Series and their evaluation should be done with utmost care.

6.3 Stock Data kNN Classifier Analysis :

As we can see from the plots for kNN Classifier models that just like linear regression, kNN also identified a drop in stock prices based on the pattern for the past years of stock market trend. The orange line in the figures shows the kNN output and it shows a drop in prices of stock in a particular area which was based on a pattern and thus the original and the predicted stock prices vary at large. Thus we can say that the kNN algorithm cannot yield better results than this and move further with another approach to solving the problem of stock market analysis price prediction using machine learning and deep learning.

6.4 Stock Data LSTM-Stacked Autoencoders Analysis :

As we can see from the plots for LSTM-Stacked Autoencoder models that this model produces better predictions than the previously mentioned models. The Blue Lines shows the Original values and the Orange Lines shows the Predicted values of the Stock Market Prices using the LSTM-Stacked AutoEncoders. Also we can see a better prediction outputs at the end part of the graphs which is similar to the original values but not that accurate but almost there. The most important feature used is the Long-Short Term Memory which can store and analyze Time-Series Data. Also using wavelet transforms to de-noise that data before processing helps in better learning by the model. We can improve the drawbacks of this model by using a Hierarchical-LSTM-VAE which will be proposed below.

7 LSTM-Hierarchical-VAE Idea Pitching for Stock Market Analysis :

7.1 LSTM-HVAE Bi-Directional Encoder :

As per our understanding of the paper, the Bi-Directional Encoder ideally gives the parameterization of the latent distribution of the longer-term in context with the input sequence provided. If we use this particular model to encode our Stock Market data which is also a long term time series, we can obtain the Multivariate Gaussian Normal distributions **μ** (Mean) and **σ** (Deviation) for the variance in the prices of the Stocks.

Moreover, the two layers of LSTM's which are Bi-Directional in nature will help us encode the data from both the ends and this in turn will help in interpolating the data in between the two ends. In normal encoder this process goes from one side to other, but in Bi-Directional LSTM Encoder we can start building the latent variables and latent space from both directions which can also decrease the loss function and will give us better distributions **μ** (Mean) and **σ** (Deviation).

7.2 LSTM-HVAE Conductor :

As we proceed from the encoder part with all the encodings obtained, we go to the conductor part in the HVAE where we initialize the encoded inputs with the embeddings from latent space setting the hidden and cell state. Also, initializing the input to zeros based on batch size of the samples which is also set as a hyperparameter. Once the input is given the output of first cell provides the next cell state, hidden state and the output and these values are passed to the next LSTM and so on so forth till the last LSTM. This will help us to process the encoded Time-Series Data more accurately and efficiently.

7.3 LSTM-HVAE Decoder :

The next stage after the conductor stage is the decoder stage where each of the conductor's output becomes the input to the decoder which is again an unidirectional LSTM layer just as in the conductor layer. A hyperparameter specifies that the layer length to number of blocks to each conductor is created. The states of first block cell becomes the initial state of the next block within the conductor only. There is no communication happening between the LSTM blocks of different conductors. This stage will provide us the outputs for each latent state of the Stock Market pricing, since we need to obtain the entire time series prediction which is Long-Term in nature the use of multiple LSTM networks will indeed ease our work with not only efficient and accurate learning but also preserving the essence of time for these sensitive natured data structures.

Since each conductor decodes recursively by depth first and the conductors are independent of the base decoder outputs we can expect independent and focused evaluations of the data provided. This forces the base decoder to utilize the latent vectors and thereby helping to learn longer sequences which is very necessary in the case of Time-Series data structures. At the final output stage, where the output from all the decoders are merged to give one single output with the reconstruction loss calculated by comparing the final output with the actual output we can obtain a better performance than the previously used models. The interpolation process will definitely help in a more accurate and successful prediction and evaluation of Time-Series datasets.

7.4 LSTM-HVAE Pitch Conclusion :

Thus, we can conclude from all the above experiments that LSTM-Hierarchical HVAE's can help us model and evaluate the Longer-Term time sequences more accurately and efficiently due to their unique properties. It would be very interesting to see the model up and running on such a dataset. We can expect a far better performance than the LSTM-Stacked Autoencoders, but we can also expect to face a few problems due to limitation of technology available and a thorough understanding of some of the latent layers used. Also, we need to analyze this particular model in more depth for implementation and reuse. Overall, it can be clearly understood and analyzed that the use of such a novel approach can help us solve the problems related to modeling sensitive time series datasets.

8 Conclusion :

Stock Market Analysis has been an interesting topic for many Data Scientists and Computer Scientists since the Time-Series data modeling is not easy and to preserve the essence of time and predicting values based on such data can be very tricky. In the above project we tried to model and predict this type of sensitive data using models like Moving Average, Linear Regression, kNN Classifiers, LSTM-Stacked Autoencoders etc. We also observed the plots of each of the models for 3 companies stock prices i.e. Apple, Microsoft and Google; it was observed that the LSTM-Stacked Autoencoder model performed significantly better than others. Although, that being the case it was not that accurate, so we proposed a better solution to modeling such sensitive data using the LSTM-HVAE's from the original paper Music-VAE's based on Google Brain Teams research Project for music interpolation. Moreover, it will be interesting to see the results obtained after the use of the LSTM-HVAE for modeling time series data such as stock market. Furthermore, based on the understand of the model we can say that it will yield far better results that the previously used models. Lastly, we can conclude by say that since stock market is prone to risks and changes in the surroundings and environmental changes in various sectors of work such sensitive Time-Series data will be difficult to model; but we can try to reach closer to the true values with development in such advanced technique for modeling such sensitive data.

References

[1]https://nips2017creativity.github.io/doc/Hierarchical_Variational_Autoencoders_for_Music.pdf
[2] https://arxiv.org/pdf/1803.05428/
[3] https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf/
[4] https://ntguardian.wordpress.com/2016/09/19/introduction-stock-market-data-python-1/
[5] https://simple.wikipedia.org/wiki/Bar_(music)/
[6] https://en.wikipedia.org/wiki/Loop_(music)/
[7] https://en.wikipedia.org/wiki/Stock_market
[8] https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/
[9] https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/
[10] https://medium.com/analytics-vidhya/using-machine-learning-to-predict-stock-prices-c4d0b23b029a/
[11] https://medium.com/@musicvaeubcse/musicvae-understanding-of-the-googles-work-for-interpolating-two-music-sequences-621dcbfa307c/