HYBRID APPROACH FOR FINANCIAL TIME SERIES ANALYSIS

Session Monsoon 2020

Submitted By

ADITYA PURANIK (1710110025)

VAIBHAV SACHDEVA (1710110369)

Under Supervision

of

DR. MADAN GOPAL

(Professor, Electrical Department)

Department of Electrical Engineering
School Of Engineering
Shiv Nadar University
(Monsoon 2020)

Candidate Declaration

I hereby declare that the thesis entitled “Hybrid Approach for Financial Time Series Analysis” submitted for the B.Tech Degree program. This thesis has been written in my own words. I have adequately cited and referenced the original sources.

Aditya Puranik

(Aditya Shrish Puranik)

(1710110025)

Vaibhav Sachdeva

(Vaibhav Sachdeva)

(1710110369)

Date: November 18, 2020

CERTIFICATE

It is certified that the work contained in the project report titled “Hybrid Approach for Financial Time Series Analysis,” by “Aditya Shrish Puranik, Vaibhav Sachdeva,” has been carried out under my/our supervision and that this work has not been submitted elsewhere for a degree.

Dr. Madan Gopal
Electrical Engineering Department
School of Engineering
Shiv Nadar University
November,2020

ABSTRACT

When constructing and selecting a portfolio for investment, evaluation of its expected returns and risks is considered the bottom line. Minimizing risks and maximizing expected returns are objectives of Portfolio Optimization. In Modern Portfolio Theory correlation is used to include diversified assets that can help reduce a portfolio's overall risk and hence optimize it. Stock price time series data encompasses both linear and non-linear tendencies. The ARIMA model being an ideal approach for tackling linear time series data and Recurrent Neural Networks being capable of understanding temporal dependencies and non-linearity well, we aim to develop a hybrid model (combining ARIMA and RNNs) for predicting the stock price correlation coefficient of two individual stocks. In the past, apart from the predictive mathematical models that assume correlation coefficient to be static and not changing with time, traditional Neural Networks have also been used to predict future stock returns, which have given noteworthy results. Through our project, we try to do a comparative study of previously existing models with our own proposed hybrid model.

Keywords: Recurrent Neural Network, Long Short-Term Memory cell, ARIMA model, Stock Correlation Coefficient, Portfolio Optimization

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

A **portfolio** is a collection of financial investments like stocks, bonds, commodities, cash, and cash equivalents, including closed-end funds and exchange-traded funds (ETFs). People generally believe that stocks, bonds, and cash comprise the core of a portfolio. **Portfolio optimization** is the process of selecting the best portfolio (asset distribution), out of the set of all portfolios being considered, according to some objective. The objective typically maximizes factors such as expected return, and minimizes costs like financial risk. Factors being considered may range from tangible (such as assets, liabilities, earnings or other fundamentals) to intangible (such as selective divestment). **Investment** is a forward-looking activity, and thus the covariances of returns must be forecast rather than observed. Portfolio optimization assumes the investor may have some risk aversion and the stock prices may exhibit significant differences between their historical or forecast values and what is experienced. In particular, financial crises are characterized by a significant increase in correlation of stock price movements which may seriously degrade the benefits of diversification.
We have worked with Python language. We chose python because of its versatility, expandability and availability of numerous open source packages that have been tested thoroughly by thousands of users over a lot of years. Python being a general purpose programming language also gives the advantage of instant deployability in almost every working environment: Linux, Windows, Mac, etc. without worrying about compatibility and cost of deployment.

1.1 Motivation

In recent times, individuals have been investing more of their time in handling their monetary assets as compared to a couple of decades back. The 'middle-man' in the investment process is becoming obsolete. Although financial advisors hold their importance, with abundance of domain knowledge available, independent research on the individual's side has become rife. Portfolio selection is one of the major steps in investment. While constructing the portfolio we need to take in consideration the expected returns and risks. An investor's goal is to maximize the profits/returns, while minimizing the risks.

1.2 Problem Statement

We aim to develop a hybrid model involving ARIMA and Neural Networks to predict future correlation coefficients of stock pairs, which would help in portfolio optimization. These stock pairs are randomly chosen from the S&P500 companies. In the first phase, the ARIMA model catches the linear tendencies in the time series data. Then, the LSTM model attempts to capture non-linearity in the residual values, which is the output of the former phase. We focus on the hybrid model's versatile predictive potential to capture both linearity and non-linearity. The developed Hybrid model will be compared with other presented mathematical model and the best model will then be chosen.

1.3 Dataset

For the purpose of this paper, we consider the S&P500 firms adjusted closing price. The data is obtained using Yahoo! Finance API for a period of 01-01-2010 to 31-12-2019. The data has no missing values, and thus no pre-processing was needed at the forefront. Out of the 504 stocks on the S&P500 index list, we randomly chose 150 stocks. To make the process more robust in terms of randomness, we first chose 200 stocks randomly and then chose 150 stocks randomly from among them. The data set, which spans 2517 active days in total, is then used to calculate correlation coefficient values pairwise. Now, since there are 150 stocks, we would get C_2^{150} pairs. We calculate correlation-coefficient of each pair of assets with a 100-day time window. To add variation, we chose 5 different starting points, 1st, 21st , 41st, 61st and 81st and apply a 100-day rolling window with a 100-day time stride until end of the dataset. This provide us with 55875 sets of time-series data (C_2^{150}* 5), each with 24 time steps. [1]

Chapter 2

Literature Review

2.1 Literature Survey

In Modern Portfolio Theory proposed by Markowitz [2], we look at various ways to compute the returns and risks of a portfolio. The proposed method was a first-shot at quantifying the attributes of the portfolio. The two-stage approach suggested by Markowitz presents us with derived return and risk, on which we develop our efficient decision bound. The investor thus chooses his/her own decision boundary from the curve of efficient combinations of returns and risks, aiming at highest reward-to-risk ratio while maintaining his/her risk tolerance. Although Markowitz mentions the use of correlation co-efficient in computation of expected risks, he assumes them to be constant and fixed [2]. His assumption proved to be wrong, as it was otherwise proved that correlation is not stable term [3] [4]. This attempt, although unsuccessful at giving a perfect picture, paved the way from further improvements to be made on the similar base. One of them is Constant Correlation model, which sets which equates all the asset relationships with the mean of all the asset correlation coefficients in the portfolio. Many other traditional and statistical models have sprung up like Multi-group model, Single-index model etc. [5][6], but we have seen very few implementations using the neural networks. If any of the previous studies in the financial domain is to be considered, neural network might just perform better than the mathematical models.

2.2 Statistical Models for Correlation Prediction

2.2.1 Full Historical Model

Full Historical Model, one of the simplest models intuitively and to implement, assumes that the correlation of two assets for a certain period of time in future would equal to the correlation values in the past [5]. This assumption is based on a general idea that the past values of a certain securities would best predict the future values. Implementation wise, we just need to calculate the pairwise correlation coefficient of the historical period and use this value to base our estimate of the future. Although this model has been enhanced and is no longer used in

real-life predictions, we have kept it as a benchmark.

$$\hat{\rho}_{ij}^{(t)} = \rho_{ij}^{(t-1)}$$

where *i, j*: asset index in correlation coefficient matrix.

2.2.2 Constant Correlation Model

As mentioned before, many models have developed keeping base as the historical model. One of them is Constant correlation model. Here we just suppose that historic data convey only knowledge about the mean coefficients of correlation, and that the average differences in pairs observed are random or sufficiently unpredictable, making zero a better approximation of its future (than its historical level). Therefore, while applying the constant correlation model, all assets in a single portfolio have the same correlation coefficient [1][5].

$$\hat{\rho}_{ij}^{(t)} = \frac{\sum_{i>j} \rho_{ij}^{(t-1)}}{n(n-1)/2}$$

2.2.3 Single-Index Model

One of the most interesting and widely used model developed was Single-Index model. Here the key assumption made is that any kind of securities move together in a systematic manner only in response to single index [1]. The single index can be anything from GDP to petrol prices, we would be assuming that as the market return here [5]. This specialization of the single-index model is known as 'market model'. The 'market model' relates the return of asset i with the market return at time t, which is represented in the following equation:

$$R_{i,t} = \alpha_i + \beta_i R_{m,t} + \varepsilon_{i,t}$$

where,
$R_{i,t}$: return of the asset i at time t
$R_{m,t}$: return of the market at time t
α_i: risk adjusted excess return of asset i
β_i: sensitivity of asset i to the market
$\varepsilon_{i,t}$: residual return, error term
such that $E(\varepsilon_i) = 0$; $Var(\varepsilon_i) = \sigma_{\varepsilon_i}^2$

Here we use the β of asset i and j to estimate the correlation coefficient.

$$Cov(ij) = \rho_{ij}\sigma_i\sigma_j = \beta_i\beta_j\sigma_m^2$$

thus,

$$\rho_{ij} = \frac{\beta_i\beta_j\sigma_m^2}{\sigma_i\sigma_j}$$

β in simple terms is just the variance of the stock/security with the market value. Betas we assume here are unadjusted Betas. There are number of ways the Betas can be adjusted [5], but we have proceeded with the unadjusted Betas. As mentioned, these Betas are from a least square regression of security returns on a market index, here S&P500 index, over a historical period of time.

2.2.4 Multi-group model

The multi-group model assumes that the correlation coefficients between two firms in one group are identical for the members of the same group, which equals to mean correlation of industry sector's pair correlation value. This is done to take the industry of the firms into consideration since when the market moves the assets of firms belonging to same industry generally perform similarly. It derives a lot from the Constant correlation model, just applies to industry sector instead of the overall market. In past studies, this model has outperformed the single-index model, constant correlation model, multi-index model, and obviously the historical model [1][6].

$$\hat{\rho}_{ij}^{(t)} = \begin{cases} \dfrac{\sum_{i\in\alpha}^{n_\alpha} \sum_{j\in\beta;i\neq j}^{n_\beta} \rho_{ij}^{(t-1)}}{n_\alpha(n_\beta-1)}, & \text{where } \alpha = \beta \\ \dfrac{\sum_{i\in\alpha}^{n_\alpha} \sum_{j\in\beta;i\neq j}^{n_\beta} \rho_{ij}^{(t-1)}}{n_\alpha n_\beta}, & \text{where } \alpha \neq \beta \end{cases}$$

where,

α/β: industry sector notation

n_α/n_β: the number of assets in each industry sector

2.3 Multi-Layer Perceptron

Multilayer Perceptron (MLP) is a kind of primary artificial neural network which consists of a finite number of continuous layers. One MLP at least has three layers including input layer, hidden layer, and output layer. In many cases, there could be more hidden layers in the MLP structure which can deal with approximate solutions for many complex problems such as fitting approximation. One MLP can be thought of as a directed graph mapping a set of input vectors to a set of output vectors, which consists of multiple node layers connected to the next layer. These connections are generally called links or synapses [7]. In addition to the input nodes, each node is a neuron with a nonlinear activation function. A supervised learning method (BP algorithm) called backpropagation algorithm is often used to train MLP. MLP is a generalization of the perceptron, which overcomes the weakness that the perceptron can't recognize inseparable linear data. The layers of MLP are fully connected, which means every neuron of each layer is connected with all neurons of the previous layer, and this connection represents a weight addition and sum.

2.3.1 Algorithm

For stock price predictions ANN technique is used with backpropagation. During training phase forward propagation is done in neutral network. After the forward pass output value is generated at the output layer nodes. During the forward pass initially total input to node is calculated, then the output of the node is calculated using activation function.

In feed-forward ANN, the neurons receive a number of inputs, the neuron total input is calculated using formula –

Total Input -

$$n_1 w_1 + n_2 w_2 + ... + n_m w_m + w_b$$

where, $n_1, n_2 ... n_m$ - input neurons
$w_1, w_2 .. w_m$ - Weights associated with input neurons
w_b is the weight associated with bias
Output of the neuron is calculated using Activation function. In this case we have used tanh function. The tanh function is the hyperbolic tangent. Unlike the sigmoid, which renders value between 0 and 1, the hyperbolic tangent outputs value between -1 and 1 which is suited for our regression problem. **The tanh function:**

$$\frac{e^{TotalInput} - e^{-TotalInput}}{e^{TotalInput} + e^{-TotalInput}}$$

2.4 ARIMA - LSTM Model

Time series data is believed to be composed of the linear and non-linear components which can be expressed as [8]:

$$x_t = L_t + N_t + e_t$$

where L_t and N_t represents the linearity and non-linearity of data at time t. The e is the error term.

The ARIMA performs well on the linear problems in the data, whereas LSTM has proven to perform well on the non-linear aspect of data. Therefore to encompass both linear and nonlinear tendencies in the model, the two models are consecutively merged. The ARIMA model is the former sector, and the LSTM model is the latter one.

2.4.1 ARIMA Modeling

ARIMA models are generally used in the forecasting a time series which can be made "stationary" by differencing, whenever necessary. A time series is called *stationary* if:

1. it has no trend

2. its variations around its mean have a constant amplitude.

3. its short-term random time patterns always look the same in a statistical sense.

In terms of power spectrum representation, an ARIMA model can be viewed as a "filter" that tries to separate the signal from the noise, and then extrapolate that signal into the future to obtain forecasts. Basically, the ARIMA model is a linear regression model modified to track linear trends in stationary time series results. The forecasting equation of ARIMA is represented as ARIMA(p,d,q). This equation depends on *lags of the dependent variable* and/or *lags of the forecast errors*. That is:

Predicted value of Y = a constant and/or a weighted sum of one or more recent values of Y and/or a weighted sum of one or more recent values of the errors.

The acronym **ARIMA** stands for **Auto-Regressive Integrated Moving Average**. Lags of the stationarized series in the forecasting equation are called "*autoregressive*" terms, lags of the forecast errors are called "*moving average*" terms, and a time series which needs to be differenced to be made stationary is said to be an "*integrated*" version of a stationary series. In ARIMA(p,d,q), p, q is each is the order of the AR model and MA model, and d is the order of differencing applied to the data. These all values are integers. In terms of y, the general forecasting equation is:

$$\hat{y}_t = \mu + \phi_1 y_{t-1} + .. + \phi_p y_{t-p} - \theta_1 e_{t-1} - .. - \theta_q e_{t-q}$$

The term μ is a constant; ϕ_k and θ_k are coefficients value of AR model variable y_{t-k}, and MA model variable e_{t-l}. The signs of the moving average parameters $\theta's$ are negative, following the convention introduced by Box and Jenkins [9]. We have used the Alkaline Information Criterion(AIC) for the paramter estimation step.

$$AIC = -2ln(\hat{L}) + 2k$$

The $ln(\hat{L})$ notation is the value of the likelihood function, and k is the degree of freedom, that is, the number of parameters used. A model that has a small AIC value is generally considered a better model. There are different ways to compute the likelihood function, $ln(\hat{L})$. We use the maximum likelihood estimator for the computation. This method tends to be slow, but produces accurate results. Finally, we use as the input for the corresponding LSTM model the residual that is determined from the ARIMA model. The residual is considered to encompass the non-linear characteristics as the ARIMA model has defined the linear pattern.

$$x_t - L_t = N_t + e_t$$

The e_t would be the final term of the model.

2.4.2 Recurrent Neural Network

Recurrent Neural Networks (RNNs) were introduced in 1982 by John Hopfield. They were introduced to tackle problems wherein the input has a dynamic temporal nature. RNNs are essentially designed for data having dependencies within itself. Traditional Feed Forward Neural Networks were unable to lay hold of the sequential aspect of the data while Recurrent Neural Networks take previous outputs into account to handle the sequential nature of inputs.

Architecture of Recurrent Neural Network

RNNs are also known as feedback networks in which connections between units form a directed cycle. RNNs can use their internal memory to process arbitrary sequences of inputs. In RNN, the signals travel both forward and backward by introducing loops in the network as shown in figure 2.1.

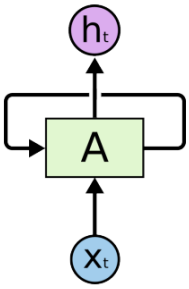


Figure 2.1: RNN Unit

RNNs can be seen as enveloped Feed Forward Neural Networks (FFNNs) where each cell has connection with previous cell and a new input coming in. It can be thought of as multiple copies of the same network; each of them passing a message to a successor. Every cell's output is connected to the next cell as shown in figure 2.2. The fact that RNNs can use their internal state to process sequences of input makes them applicable to numerous tasks including but not limited to predicting stock price correlation coefficients.

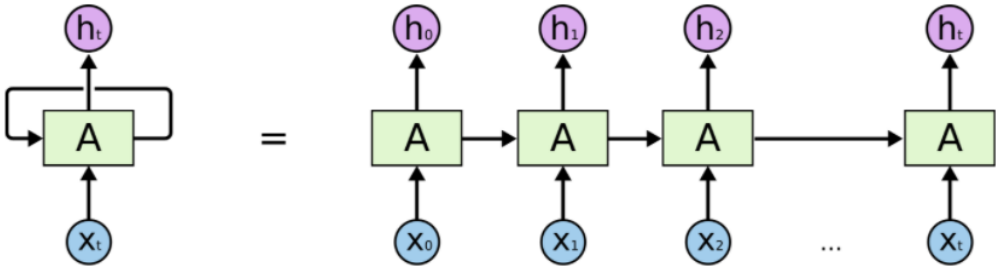


Figure 2.2: RNN Network

Working of Recurrent Neural Networks

RNN takes input in the form of a 3-D matrix in which all the instances are stored row wise. Instances are fed to the network (layer of neurons) and errors are calculated using predicted output and actual output. This error is then back propagated to update the weights by computing gradients similar to FFNNs.

Back Propagation in Time

To reduce the errors by backpropagation method, RNN uses Back Propagation Through Time (BPTT) to compute gradients for updating the weights. This means that the gradient needs to be calculated over time. Since there is no direct connection of output to the very first input, BPTT follows a chain like sequence of partial derivatives to observe the change. The whole point of introducing RNN was to take into account the sequential data and learn from the past to apply in the future. In theory, RNNs are absolutely capable of handling long term dependencies, but, in practicality, RNNs face problems of vanishing and exploding gradients in handling long term dependencies and tend to forget things that need to utilize memory from long back.

Short Term Dependencies

Sometimes, we only need to look at recent information to perform the present task. Consider a word predicting problem which has very few sentences and length of sentences is also very less. This problem does not require a lot to be remembered that the problem of vanishing or exploding gradient could arise. In such cases, where the gap between the relevant information and the place where it's needed is small as shown in figure 1.3, RNNs can learn to use the past information. In figure 2.3, it is shown that the output h_3 is at max. dependent on inputs x_0 and x_1 which are not very far away to cause vanishing and exploding gradient problems.

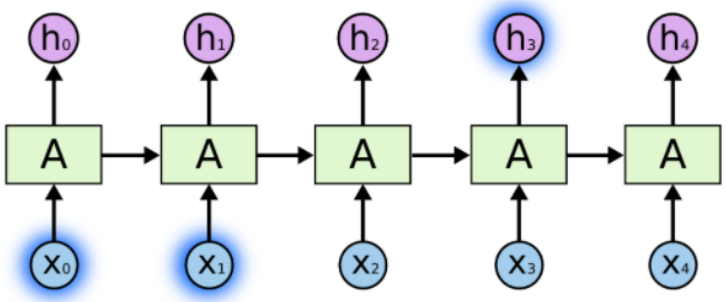


Figure 2.3: Short Term Dependencies

Long Term Dependencies

There are times when we need to look at information very far away from our current point. An intuitive example would be to suppose that there is a million-word long text file in which the

last word depends on an information provided in the very first line. This type of dependency is called long term dependency and RNNs practically face vanishing and exploding gradient problems. A network model depicting this type of dependency is shown in figure 2.4 in which the output h_{t+1} is dependent on the inputs x_0 and x_1 far away. This could cause our model the mentioned gradient problems.

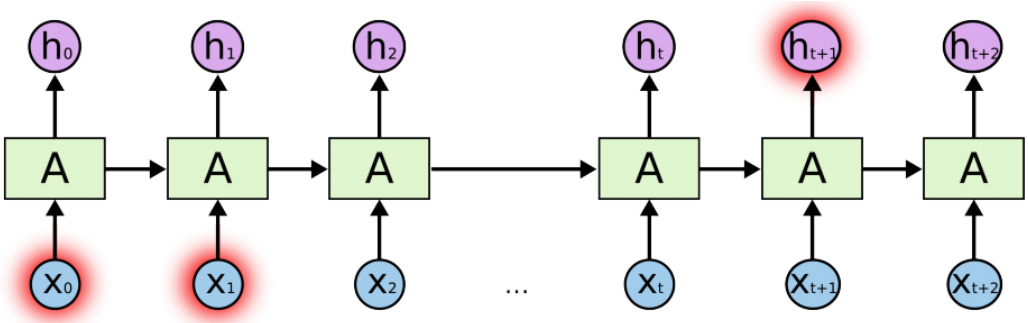


Figure 2.4: Long Term Dependencies

Hence, it can be said that RNNs remember things for just small durations of time, i.e. if we need the information after a small time, it may be reproducible, but once a lot of words are fed in, this information gets lost somewhere. This issue can be resolved by applying a slightly tweaked version of RNNs – the LSTM.

2.4.3 Long Short Term Memory

As mentioned by Razwan et. al. [10] and Bengio et. al. [11], it is very difficult for RNN models to recollect the information from the initial states for long term time dependencies due to vanishing gradients. Hochreiter and Schmidhuber [12] proposed the Long Short Term Memory (LSTM) approach in 1997 to tackle the problems faced by RNNs. LSTM is a modified version of Vanilla Recurrent Neural Networks but the basic principle has remained the same. It is basically a RNN with extra control of the memory through internal mechanism of several gates inside each LSTM cell. We can control the amount of information relevant to us (which is to be retained) and the amount that we can forget. The architecture of LSTM is shown in figure 2.5

As shown in the above figure, an LSTM cell consists of cell state, hidden state and several gates. The components of LSTM will be explained further.

Components of LSTM

1. **Cell State :** Cell state or the Memory state is responsible for solving gradient problems. It is one of the major components of the LSTM structure which acts as transport highway. Basically, it holds critical information that it has learned over time. The cell state carries information throughout the sequence processing. Information through earlier time steps

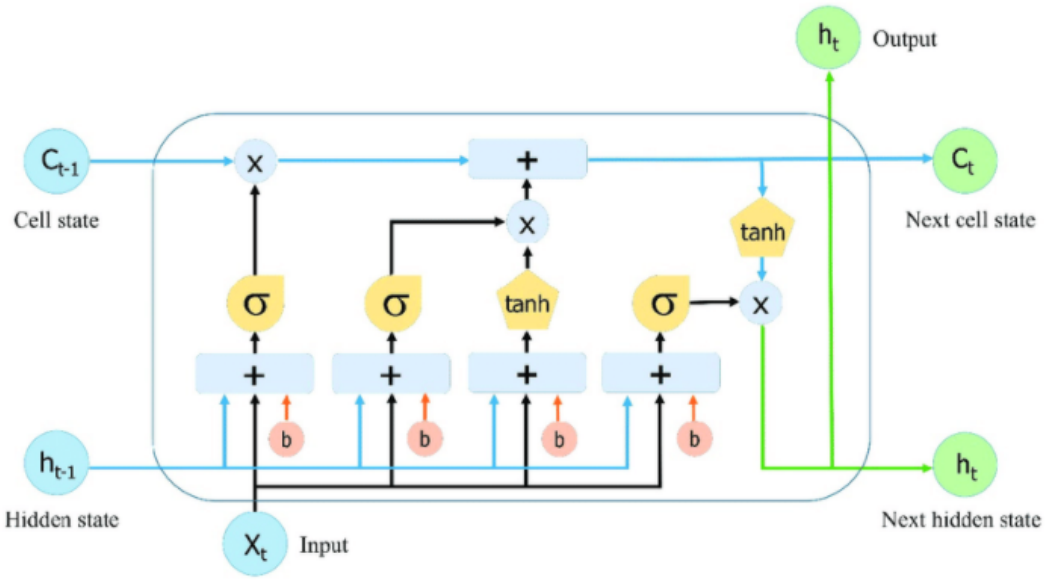


Figure 2.5: LSTM Cell

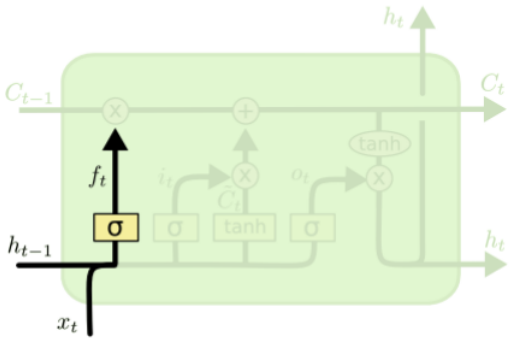
could be carried all the way to the last time step. The network is designed to effectively maintain useful information in the memory cell over many time steps. Thus, cell state eradicates the problem of memory loss due to vanishing gradients. The information gets added or removed from the cell state via gates.

2. **Hidden State :** The output from the previous cell is called the hidden state. Similar to the output that was being fed to the RNN's next layer as input along with the actual current input as shown in figure 2.2. The only difference is that the output for RNN was based on the memory that it had recollected from just previous output while here the hidden state contains the output based on the memory collected by the cell state. The LSTM units are engineered to provide more flexibility by emitting an output that is an "interpretation" or "external communication" of what that cell state represents.

3. **Gates :** Gates are a way to optionally let information through the cell state. They are composed out of a sigmoid neural network layer and a pointwise multiplication operation. A value of zero means "let nothing through" while a value of one means "let everything through". LSTM has 3 gates namely input gate (write gate), output gate, and a forget gate (keep gate) to control the information content of the cell state.

    - **Forget/Keep Gate**

      First, the unit must determine how much of the previous memory it has to keep. The cell state from the previous time step h_{t-1} is rich with information, but some of that information might be stale and therefore might not be needed. So, the forget gate decides which information to keep and which to throw away. The gate shown in figure 2.6 consists of a neural net with sigmoid function as the activation function,



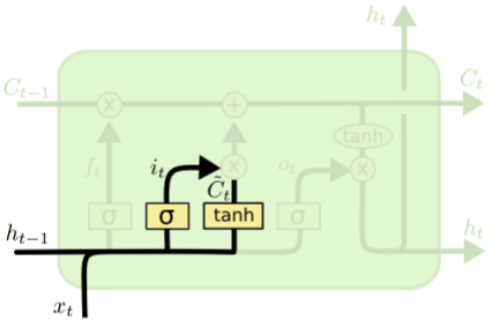
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.6: Forget/Keep Gate

and current input x_t and hidden state of previous time step h_{t-1} after concatenation as the input with W_f as weight of the neural net and bf as bias. The neural net then creates a bit tensor mask of values close to 0s and 1s due to sigmoid activation. The sigmoid output of this neural net is then multiplied with the previous cell state. If a particular location in that tensor holds a value close to 1, it means that location in the memory cell is to be kept while if that particular location instead held a 0, it means that location in the memory cell is no longer relevant and has to be erased.

- **Input/Write Gate**

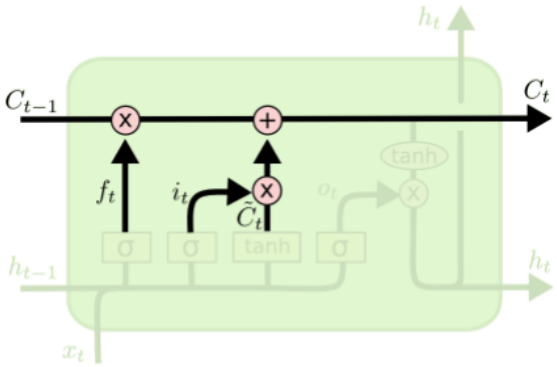
  This gate decides what new information we are going to update in our cell state. This gate actually consists of two parts. One is a tanh layer that creates a vector of new candidate values C (an intermediate tensor), which could be the possible values to be added to the cell state and decides what information we want to write in the cell state. Another is a sigmoid layer that decides on the importance of the candidate values to be added to the cell state and figures out which components of the intermediate tensor we actually want to include into the cell state and which components we want to toss by creating a bit tensor mask using the same strategy as done in the forget gate.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 2.7: Input/Write(Input) Gate

In order to update the previous cell state which, we had multiplied with the forget gate, we add the candidate values that we had just obtained post deciding their



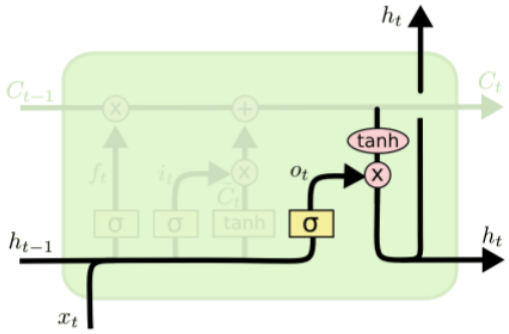
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.8: Input/Write(Write) Gate

importance with the previous cell state. The equation of the input gate's neural net is given by figure 2.7 and figure 2.8.

- **Output Gate :**

  The output gate decides what our output is going to be which is also the hidden state for the next cell. This output is used for prediction, if required, else is passed on to the next LSTM cell as the hidden state. First, the sigmoid layer decides the importance of the output values then the current cell state is passed through a tanh layer (to keep the values between -1 and 1) and then both the outputs are multiplied to get the final output or next hidden state. The new obtained hidden state and the current cell state are passed to the next LSTM layer as the input along with the actual input to predict the output. The equations of the output state are given by figure 2.9.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.9: Output Gate

Hyperparameters of LSTM

As a part of every machine learning/deep learning task, tuning of hyperparameters is extremely important in order to achieve desired accuracy. We are using LSTM layers from keras which gives flexibility of many hyperparameters so that we can tune them as per our requirement. The

task of hyperparameter tuning is highly selective and value of every hyperparameter changes according to the type of data with us.

1. **Optimizer**

   The optimizer is responsible for minimizing the objective function of the neural network. A commonly selected optimizer is stochastic gradient descent (SGD), which has proved itself as an efficient and effective optimization method for a large number of published machine learning systems. However, SGD can be quite sensitive towards the selection of the learning rate. To eliminate the shortcomings of SGD, other gradient based optimization algorithms have been proposed. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data. Adam is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks [13]. It can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum. Adam optimizer has given promising results in literature and thus we will use this as our optimizer.

2. **Epochs**

   Number of epochs are defined as the number of times we are training our model or the number of times we are going back and reuse the dataset to update weights once the whole training is done. It is advised not to keep the number very high as it causes the model to overfit and costs poor testing accuracy. The value should not be too less either as we don't want our learning to stop while our model has not converged yet. This parameter is rather an empirical one and is selected based on hit and trial methods and can be reduced once for high validation accuracy, you start getting lower testing accuracy.

3. **Number of LSTM Layers**

   Decision of Number of LSTM layers or rather depth of an LSTM model has a trade-off between computational complexity and fine tuning. It is possible that we achieve higher accuracy with increase in the depth but, this increases the time required for our model to train substantially. We have used 20 LSTM layers (for 20 time steps) in our model.

4. **Number of Recurrent Units**

   This is a parameter that varies from problem to problem and can only be set with domain knowledge. For our dataset of Stock Price Correlation Coefficients, we will be using 25 LSTM units in one layer.

5. **Dropout**

Dropout is a popular technique for neural networks to deal with overfitting [14]. No dropout, naive dropout, and variational dropout are the three most widely used dropout variants. We apply a randomly selected dropout mask for each LSTM output in Naive Dropout. The mask varies from time step to time step, although the recurrent connections are not dropped. This type of dropout is suboptimal for recurrent neural networks, as noted by Gal and Ghahramani [15]. They suggested using Variational Dropout instead, in which they propose to use the same dropout mask for all the recurrent layer timesteps i.e. at each timestep the same positions of the output are dropped. They also propose to use the same strategy for dropping recurrent connections. The fraction p of the measurements dropped is a hyperparameter which is randomly selected from 0.0, 0.05, 0.1, 0.25, 0.5 , but , as specified in the Keras documentation, any float value from 0 to 1 can be used. For variational dropout, p is selected independently for the output units as well as for the recurrent units.

Chapter 3

Work Done

3.1 Data Scraping

We scraped the list of S&P500 stocks from Wikipedia. The scraped list comprised of Ticker, Company Name, GICS Sector and GICS sub-industry. Among the extracted data, 200 stocks were first randomly selected. Extraction of stock data for these 200 stocks was done through Yahoo! Finance API. The stock data comprised of Opening price, Closing Price, High, Low, Volume and Adjusted Closing Price of each stock from 1^{st} January, 2010 to 31^{st} December, 2019, spanning 10 years. Finally, the S&P500 index value was also extracted for the same time period.

3.2 Data Pre-Processing

Out of the randomly chosen 200 stocks, a check was performed for the completeness of the data. It was found that only 185 out of the 200 companies had complete data in the specified time interval. As a result, the remaining 15 companies were discarded and were not considered further. Out of the 185 companies that were left, 150 were chosen at random, again. A check for null values was performed on this data. It was observed that there were no missing and/or null values in the data. Adjusted closing price obscures the impact of key nominal prices and stock splits on prices in the short term and hence, only the adjusted closing price (out of the multiple features of a stock) was chosen for calculating the correlation coefficient between two stocks. The data set, which spans 2517 active days in total, is then used to calculate correlation coefficient values pairwise. Now, since there are 150 stocks, we would get C_2^{150} pairs. We calculate correlation-coefficient of each pair of assets with a 100-day time window. To add variation, we chose 5 different starting points, 1^{st}, 21^{st}, 41^{st}, 61^{st} and 81^{st} and apply a 100-day rolling window with a 100-day time stride until end of the dataset. This provide us with 55875 sets of time-series data (C_2^{150}* 5), each with 24 time steps. The dataset produces as a result of pre-processing has a dimension of (1117500, 24). This dataset was then split into 3 components namely the training set, test set and the validation set. The training set would be used to train the model, while the validation/development set will be used for hyper-parameter

tuning. Finally, the accuracy of the different models will be calculated using the test sets.

3.3 Implementation of Statistical Models for Correlation Prediction

All the statistical models mentioned in the literature review were implemented. MSE and MAE are two common metrics, generally used to evaluate models trained for regression problems. The Mean Absolute Error (MAE) measures the absolute average distance between the real data (test data) and the predicted data, but ususally fails to punish large errors in prediction. On the other hand, the Mean Squared Error (MSE) measures the squared average distance between the real data and the predicted data. In case of MSE, the larger errors are well noted and hence is preferred over MAE. The respective MAE and MSE for all the above listed models. refer 4.1, were calculated for further comparison.

Model	**Validation**		**Test 1 Set**		**Test 2 Set**	
	MSE	MAE	MSE	MAE	MSE	MAE
Full Historical Model	0.463	0.551	0.450	0.535	0.428	0.524
Constant Correlation Model	0.259	0.444	0.214	0.403	0.277	0.423
Single - Index Model	0.331	0.481	0.379	0.501	0.378	0.492
Multi - group Model	0.269	0.447	0.255	0.427	0.294	0.434

Table 3.1: MSE & MAE of Statistical Models

Considering the above table, it can be clearly observed that the Full Historical model fails in capturing the variation of correlation coefficient, hence giving a high value of MSE and MAE as compared to other models. The other 3 models are fairly comparable, and hence would be considered for comparing with our developed hybrid model. Even out of the three comparable models, the Constant Correlation model outperforms the Multi-Index model and the Single Index model.

3.4 Implementation through Traditional Neural Network

Before understanding how the data to the neural network was fed, we need to understand the data in much more depth. We have calculated the rolling correlation with a window of 100 days. With a window of 100 days, we would get first 99 values as **NaN**. So total data points with rolling correlation calculated would be 2417. Now to maintain uniformity, we ignore the last 17 values, and thus make a vector of 24 correlations. For every pair, the rolling correlation is calculated for 1-100 values, then 101-200 and so on. Since taking each value in consideration does not help much intuitively and computation wise, we would go with 5 starting points for each stock-pair coefficient value namely 1st, 21st, 41st, 61st and 81st and similarly for all other pairs. Thus, we would have a total of all the pairs which is (C_2^{150}) and their five different

starting points, which completes our data set of 55875 points, which would comprise our total number of input neurons. Now input to each of the neuron would be (20,1) vector which consist of values with a time stride of 100.

Train Set: $\vec{x}$- Index 0-19 , $\vec{y}$ - Index 20
Validation Set: $\vec{x}$ - Index 1-20 , $\vec{y}$ - Index 21
Test Set: $\vec{x}$ - Index 2-21 , $\vec{y}$ - Index 22

Remark : We have considered the first value returned to us as 0^{th} *value which is actually the rolling correlation obtained on* 100^{th} *day.*

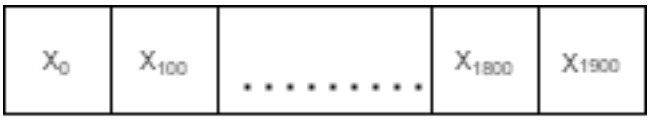


Figure 3.1: First input vector for calculation of stock-pair rolling correlation of **train** set. Each x_i represents rolling correlation at that day between the pair. The dimension are (20,1).

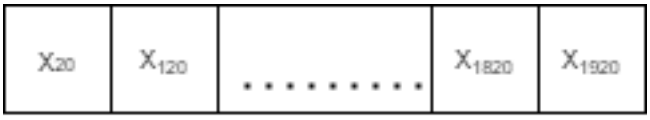


Figure 3.2: Second input vector for calculation of stock-pair rolling correlation of **train** set. Each x_i represents rolling correlation at that day between the pair. The dimension are (20,1).

These input vectors (each of dimension (20,1)) are fed to the neural network of 55875 units, which return a output of **y** which is of dimension **(55875, 1)**. This **y** is then compared with the next column of data set and the MSE and MAE are calculated.

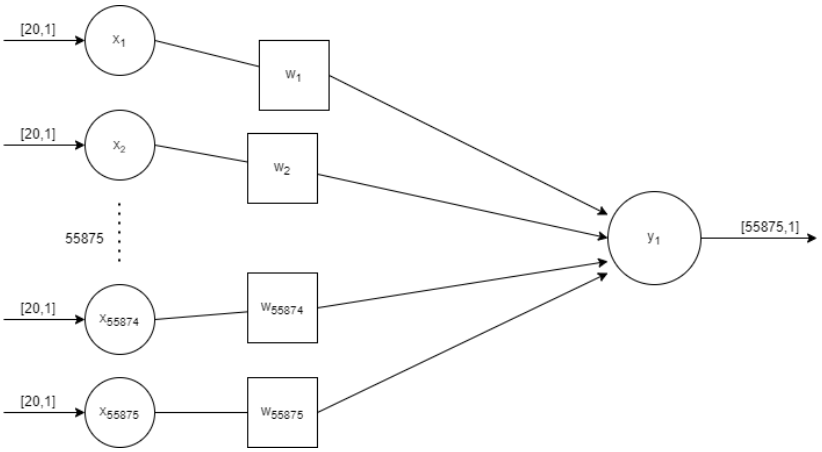


Figure 3.3: MLP Model

3.5 Implementation of Hybrid Model

3.5.1 ARIMA Modeling

The order of the model needs to be defined before fitting an ARIMA model. The judgment process is supported by the ACF plot and the PACF plot. An oscillatory pattern that appeared similar to a white noise was seen in most of the dataset. An increasing/decreasing trend, intermittent big dips with the correlation coefficient being stable stable, and mixed oscillatory-steady cycles are other noteworthy trends. While the plots of the ACF/PACF show that a large part of the dataset is similar to white noise, some orders (p, d, q) = (1, 1, 0), (0, 1, 1), (1, 1, 1), (2, 1, 1), (2, 1, 0) seem applicable. With these five orders, we fit the ARIMA model and pick the model with the least AIC value for the data from each train/validation/test dataset. For the AIC metric, the method we use to compute the log likelihood function is the maximum - likelihood estimator.

We generate predictions for each 21 time steps to calculate the residual value after fitting the ARIMA model. Then each data's last data point will act as the target variable Y, and the remainder as the variable X. For the next LSTM model sector, the new X/Y-split datasets will be the input values.

3.5.2 LSTM Modeling

Input Data

For the LSTM model, we use the residual values, derived from the ARIMA model, from the 150 randomly selected S&P500 stocks as input. Train X/Y, validation X/Y and test X/Y are used as the datasets. Every X dataset has 55875 lines with 20 time steps, each time series having a corresponding Y dataset. The data points have a value of about 0, as the input is a residual dataset.

Model Training

For our project, the model architecture is an Recurrent Neural Network that employs 25 units of LSTM. The final outputs of 25 LSTM units are combined into a single, fully connected layer value. Then the value would be transferred to produce a single final prediction via a doubled-hyperbolic tangent activation function. The hyperbolic tangent function scaled by a factor of 2 is essentially the doubled-hyperbolic tangent. Figure 5 demonstrates the model's simplified architecture. Keeping an eye on overfitting is important when training the model. Overfitting happens when the model fits poorly during testing on the dataset. Predictive quality on the train dataset will therefore be high, but on other newly added results, it will be low. A separate collection of validation datasets would be used to track this issue. For the train dataset, we train the LSTM model until the predictive outputs are identical to each other on the train dataset and validation dataset.

One of the commonly used approaches to avoid overfitting is choosing the optimal value of Dropout during hyperparameter tuning. It prevents interdependence from occurring among the neurons, which causes overfitting. This is achieved by simply shutting off the network's neurons during training with probability p. Then, dropout is disabled in the testing process and each weight value is multiplied by p to scale the output value down to the appropriate boundary. In addition, dropout has the effect of training and combining the outputs of several neural networks. [14]

Other than dropout, to avoid overfitting, we considered more regularization. Two types of methods of regularization are primarily present: Lasso regularization (L1) and Ridge regularization (L2). Such regularizers prevent the weight values of each network from being too high in the LSTM model. Each layer's high parameter values can cause the network to concentrate severely on a few features, which can lead to overfitting. The following is a general expression of the error function with regularization.

$$\sum_{i=1}^{n} Y_i - (W\dot{X}_i + b)^2 + \lambda_W \sum_{i=1}^{k} \sum_{j=1}^{k'} W_{ij}^2 + \lambda_b \sum_{i=1}^{l} \sum_{j=1}^{l'} b_{ij}^2$$

Parameters λ_w and λ_b determine the strength of the cost function's regularization. The model would be under-trained if the lambda values are too high. On the other hand, if they are too small, the effect of regularization would be marginal. In our model, it turned out that not implementing any regularization worked better after trial and error. With regularization, we sought more complex architectures, but models with no regularization had superior outputs for all architectures. The vanishing/exploding gradient is another thing to pay attention to while training a neural network model. For RNNs, this is particularly emphatic. The gradients far away from the output layer appear to be very small or large because of a deep propagation over time, stopping the model from training properly. The LSTM cell itself is the solution for this problem. The LSTM is able to connect large time intervals without loss of data [12]. Other miscellaneous details about the training process includes the use of mini-batch of size 500, the ADAM optimization function.

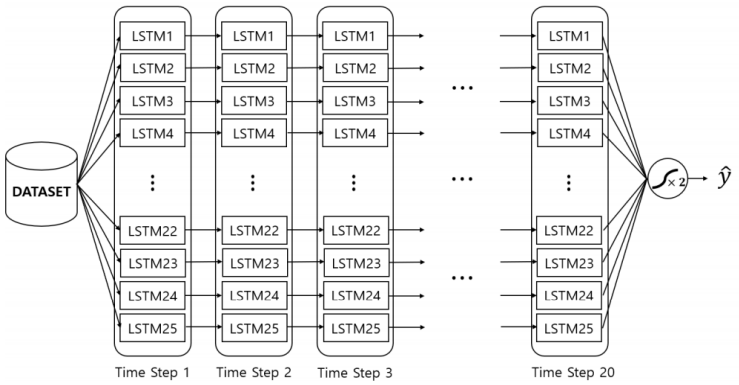


Figure 3.4: LSTM Model Sector Architecture

Chapter 4

Results and Conclusion

4.1 Results

4.1.1 ARIMA-LSTM Hybrid Model

The MSE curve for the training set began to converge after about 250 epochs (Figure 4.1). A similar pattern was also shown by the MAE learning curve. Based on both the overfitting and the efficiency metric, the optimal epoch was determined. Optimal results were obtained at the 258th epoch. Hence, the 258th epochs's model was chosen as the optimal model. With the selected ARIMA-LSTM hybrid model, the MSE and MAE values of the prediction were calculated. The MSE value for the train and test were 0.012 and 0.015 respectively. Because the values had a minute difference, it can be said that the model has been generalized sufficiently. The scatter plot of Actual v/s Predicted values for train and test set(Figure 4.3 and 4.4) provide further justification.

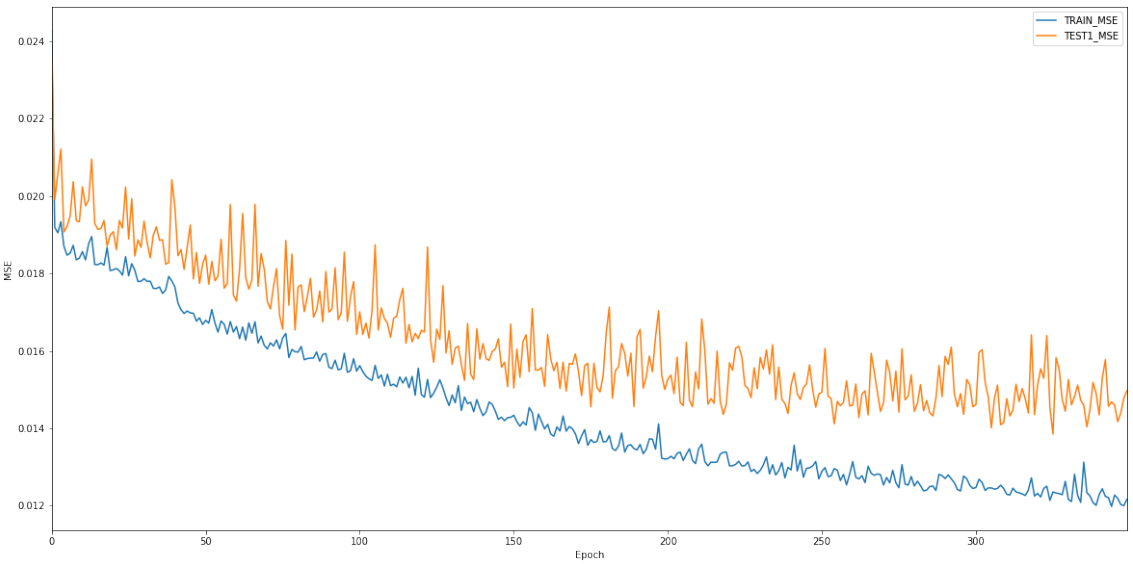


Figure 4.1: Change in MSE with change in Number of Epochs

We then compared the metric values of the developed hybrid model with those the Constant

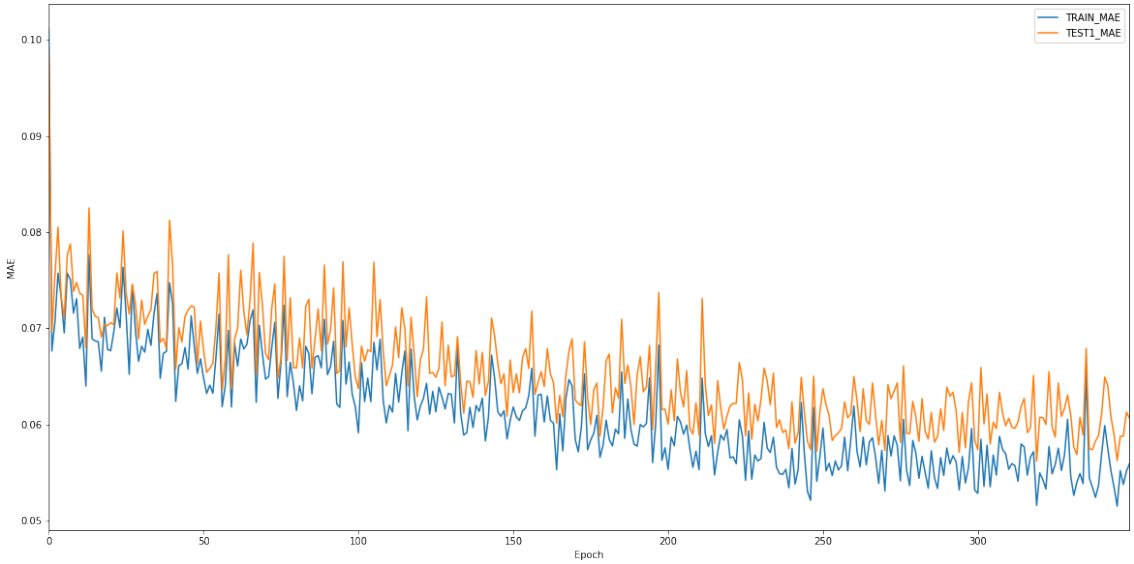


Figure 4.2: Change in MAE with change in Number of Epochs

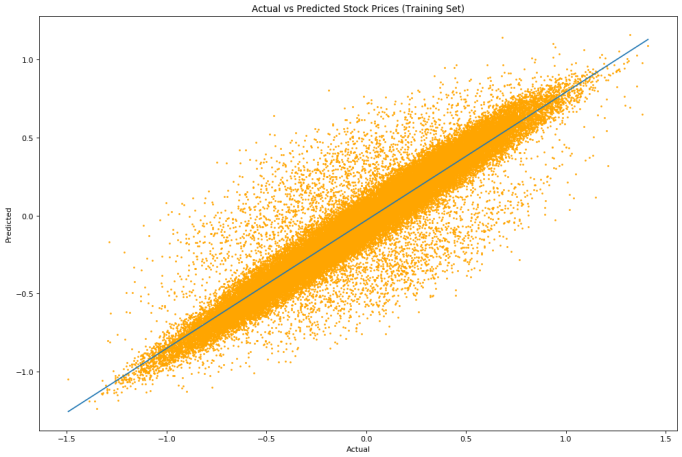


Figure 4.3: Scatter plot of Actual vs Predicted values of Train Set

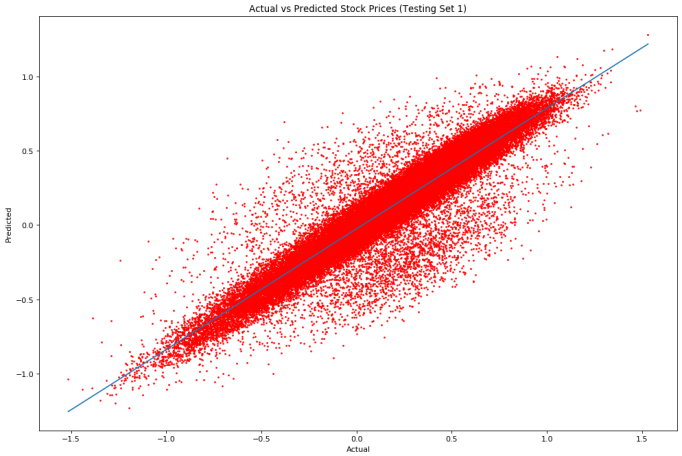


Figure 4.4: Scatter plot of Actual vs Predicted values of Test Set

Correlation Model, Traditional Feedforward Neural Networks and LSTM. The Constant Correlation model performed the best on the dataset of our 150 S&P500 stocks among the financial models, as did the analytical analysis of E.J. Elton et al. has shown [6]. Its performance, however, was far from the predictive potential of even the traditional neural networks. As expected, the traditional neural networks performed better than all the financial models while LSTM provided better results than that of traditional neural networks because of their capability to handle sequential data well. Surprisingly, the MSE value of the ARIMA-LSTM hybrid model was almost one-tenth of all the other models considered for comparison. The MAE metric showed similar results as well. Table 4.1 demonstrates all metric values for both train and test set, for each model. For each metric, the lowest values are boldfaced. Here we can conveniently see how all the ARIMA-LSTM model metric values are in boldface.

Model	**Train Set**		**Test Set**	
	MSE	MAE	MSE	MAE
Constant Correlation Model	0.259	0.444	0.278	0.456
Multi Layer Perceptron	0.247	0.423	0.255	0.435
LSTM	0.223	0.398	0.246	0.418
ARIMA-LSTM Hybrid	**0.012**	**0.056**	**0.015**	**0.060**

Table 4.1: Hybrid Model results and its comparison

Here we can conveniently see how all the ARIMA-LSTM model metric values are in boldface. For further investigation, we tested our final model on various assets of S&P500 companies. Except the 150 assets we have already chosen to train our model, we randomly selected 10 assets and created data sets with the same structures as those used in model training and testing. Since we choose only 10 assets, we have a total of 180 data points. We then pass the data into our hybrid model ARIMA-LSTM, predict the correlation coefficients and finally calculate the MSE and MAE. Figure 4.5 shows that our hybrid model is robust for new data and performs exceptionally well.

Figure 4.5: New Asset Testing

4.2 Conclusion

The aim of this project was to propose and develop a model that performs superior to extant correlation coefficient predictive models. In order to first filter out linearity in the ARIMA modeling step, then predict nonlinear tendencies in the LSTM recurrent neural network we adopted the ARIMA-LSTM hybrid model. The testing results showed that the ARIMA-LSTM hybrid model performs far superior to traditional neural networks and even LSTM (when used solely). Model performance was validated on both different combinations of assets and on different time periods with metrics such as the MSE and the MAE. The values nearly reduced to one tenth of the metric values obtained for other models taken into consideration. Judging from such outperformance, it can be presumed that the ARIMA-LSTM hybrid model has ample predictive potential. Therefore, the ARIMA-LSTM model as a correlation coefficient predictor for portfolio optimization is considerable. With a better predictor, the portfolio is optimized more precisely, thereby enhancing returns in investments.

4.3 Future Prospects

Considering the results obtained using the developed hybrid ARIMA-LSTM model, it is undeniable that hybrid models, in future, might be capable of providing exceptional results for a plethora of domains. Talking about predicting financial time series, a further extension of the above developed hybrid model could be using Bidirectional LSTM or Gated Recurrent Units (which have given noteworthy results in time series prediction) instead of LSTM. Another modification worth consideration, is using another variant of ARIMA such as the SARIMA (Seasonal ARIMA) or the VARIMA (Vector ARIMA). Undoubtedly, hybrid models developed after making the suggested modifications/enhancements will give results either better or comparable to the hybrid model developed in this project.

Chapter 5

Appendix

5.1 List of 150 stocks for model training

LEG	MS	IPGP	AIG	MYL	CTXS	STT	WST	UHS	UAA
CL	BKR	OMC	JBHT	CMG	NOC	CMS	ALXN	REG	YUM
T	AMD	KO	KR	CERN	CHD	ECL	SPG	NI	INCY
SIVB	GS	ROL	REGN	ODFL	SEE	COF	WEC	ADM	CME
FCX	DLR	TRV	HBAN	CI	MSFT	OXY	BLL	CHRW	DISCA
EXR	VRSK	MGM	SNA	BIO	STZ	AEE	ED	KSU	SLG
TFC	TMO	PCAR	LRCX	PPL	URI	PH	CCI	MKTX	SWKS
JPM	AEP	TEL	BDX	ATVI	CINF	RTX	CMCSA	HAS	COST
CMI	DAL	HES	PG	UNP	DG	GOOGL	CF	HSY	NOV
PBCT	UNM	PEG	IP	AES	INTU	DRI	LDOS	MO	KIM
HON	CB	BWA	WY	AVGO	IFF	DE	MCK	BAX	TAP
GD	XRX	HST	NLOK	SYY	ILMN	MAS	VAR	AVB	TIF
EIX	KMB	JNPR	SWK	AON	LB	SPGI	PXD	BXP	CLX
HFC	UNH	SNPS	CNC	WYNN	IVZ	ADI	HOLX	TT	PFE
PRGO	ARE	KMX	TDG	RSG	FAST	IEX	AMZN	BBY	VNO

Figure 5.1: List of Tickers used for Model Training

5.2 Code

You can find the source code along with the data here: Link

Bibliography

[1] Hyeong Kyu Choi. Stock price correlation coefficient prediction with arima-lstm hybrid model, 2018.

[2] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 2020/09/24/ 1952. Full publication date: Mar., 1952.

[3] Francois Chesnay and Eric Jondeau. Does correlation between stock returns really increase during turbulent periods? *Economic Notes*, 30(1):53–80, 2001.

[4] Frank J. Fabozzi, Francis Gupta, and Harry M. Markowitz. The legacy of modern portfolio theory. *The Journal of Investing*, 11(3):7–22, 2002.

[5] Edwin J. Elton, Martin J. Gruber, and Thomas J. Urich. "are betas best?"†. *The Journal of Finance*, 33(5):1375–1384, 1978.

[6] Edwin J. Elton, Martin J. Gruber, and Manfred W. Padberg. Simple criteria for optimal portfolio selection. *The Journal of Finance*, 31(5):1341–1357, 1976.

[7] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[8] G.Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159 – 175, 2003.

[9] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Inc., USA, 1990.

[10] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013.

[11] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[15] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks, 2016.