SELINUS UNIVERSITY
OF SCIENCES AND LITERATURE

Forecasting SOXX with Long Short-Term Memory (LSTM) Neural Networks Based on Varying Sampling Periods

By Lester A. Leong

A Dissertation

Presented to the Department of
Finance & Economics
program at Selinus University

Faculty of Business & Media
in fulfillment of the requirements
for the degree of
Doctor of Philosophy in Finance & Economics

2023

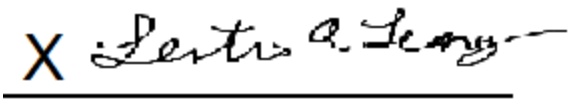
Declaration

The thesis titled “Forecasting SOXX with Long Short-Term Memory (LSTM) Neural Networks Based on Varying Sampling Periods” submitted for the award Doctor of Philosophy in Finance & Economics at Selinus University of Sciences and Literature, Faculty of Business and Media; is my original work.

I do hereby attest that the work reported therein has been carried out by me and all sources of information have been specifically acknowledged by means of references.

In my capacity as supervisor of the candidate’s dissertation, I certify that the above statements are true to the best of my knowledge and the research wok fulfils the requirements of standards set out by the University for the award of PhD.

Student ID: UNISE2175IT

X 

Lester A. Leong

01/05/2023

Signature Date

Dedication

I dedicate this study to a few individuals. First, to my beloved Lydiana. Lastly, to my first research professors who fostered my interest in research: Dr. Buhse and Dr. Whelan.

Acknowledgments

I am grateful to Elvira Di Mauro and Dr. Sabrina Mazza. From their feedback, I was able to mold scattered writings into this study presented. In addition, I would like to thank Selinus University for the opportunity to perform this research.

Abstract

This paper modeled and predicted the iShares Semiconductor ETF (SOXX) stock price using LSTM. The historical data of the SOXX were transformed into a rolling sequence starting from 4180 daily closing prices to an additional 465 daily closing prices out of sample. Compared with random prediction, the LSTM model improved the accuracy of stock returns prediction 19.7% over 50% random chance. The work showed how powerful LSTM is at predicting the stock market in SOXX, which is mechanical but much less predictable due to the varying results of hyperparameter turning.

Table of Contents

Chapter 1: Introduction

Semiconductors essentially are the backbone of many industries including cryptocurrencies, artificial intelligence, automobiles, smart appliances, etc. In light of the numerous demand for semiconductors, difficulties in analyzing non-linear patterns of pricing for semiconductors emerge. The problem stems from many issues but common examples include forecasting demand in light of technological advances, seasonality effects, price shocks due to supply chain issues, etc. In light of this complex issue and the rise of machine learning, this study aims to apply deep neural networks forecasting, specifically Long Short-Term Memory (LSTM), on iShares PHLX Semiconductor ETF (SOXX) from historical prices. The type of data will only be SOXX historical price data collected from Yahoo Finance from the dates of 2009-2020, and the frequency of the data collected will be daily. The data will be analyzed with Python and modeled with a deep learning neural network time series forecasting approach: Long Short-Term Memory (LSTM) with varying training periods. Training in data science refers to the sampling period that the model takes in before creating a forecast. The forecast period will be 12 months, effectively an annualized forecast.

Background of the Problem

A common starting point in modern finance includes the Efficient Market Hypothesis (EMH). EMH states that market participants cannot accrue above-average profits without accepting additional risk (Ţiţan, 2015). EMH is normally combined with the Random Walk Theory to claim that prices cannot be forecasted, since the new information was not within the past or present prices (Weng, et al., 2018). EMH claims that prices follow a random walk, which makes accurate price predictions impossible. The topic of EMH has been popularly debated and has had both academic and practical evidence against it. For example, more evidence shows statistical time series and machine learning models making accurate future predictions from past prices, and famed investor Warren Buffett has consistently delivered abnormal returns (Weng, et al., 2018).

What appears to be randomness might be a non-linear pattern, which SOXX has plenty of. Given the right model, these complex patterns may be analyzed for an accurate forecast. The semiconductor business is known for large fluctuations and predicting the turning point of the business cycle is of greater importance for business owners. The typical product life cycle for

semiconductors is roughly 18 months and a forecasting advantage of a business cycle turning point would drastically improve profits (Liu, 2006). Prior models applied to forecast semiconductors have only included autoregressive moving average, Markov switching, and vector autoregressive (Aubry, 2014).

Asset price forecasting models based on past prices tend to fall into two broad categories: statistical time series and machine learning-based. For statistical time series models, the most popular ones include autoregressive integrated moving average (ARIMA) and generalized autoregressive conditional heteroscedasticity (GARCH). The limitations of these statistical models include a requirement for model pre-specification, increasing estimation error with increased model complexity, and inferior predictive performance compared to machine learning (Weng, et al., 2018). On the other hand, machine learning models for time series forecasting can be classified into four broad categories: artificial neural networks, classification and regression, ensembles, and hybrid approaches. This paper will focus on a specific type of artificial neural network known as long short-term memory (LSTM).

The first type of neural network to forecast time series events was a recurrent neural network (RNN), but the problem with RNNs is their inability to store long time series (Su, 2020). In light of this long-term neural network problem, researchers improved RNNs into what is known today as a long short-term memory (LSTM) network (Hochreiter, 1997). In a normal RNN, long time series forget their beginning data, since more recent data take up higher weights that are calculated through the neural network's forecast error as gradients. These gradients that happen long ago end up being forgotten, which coined the popular term vanishing gradients (Su, 2020). LSTMs solve the vanishing gradient problem by including another component with the neural networks known as the cell state. The cell state keeps track of all major information from all the neural networks which solves the long-term memory problem of RNNs.

This paper aims to further examine the performance of LSTM time series forecasting with financial prices. Other research has shown the performance of LSTM for large indices such as the S&P 500, Dow Jones Industrial, and Nasdaq with promising results, yet similar papers are not too transparent in their model creation for independent replication (Shen, 2020). Other time-based LSTM models have displayed promising results such as a 53% accuracy with an annualized return of 46% from forecasting US indices (Su, 2020). This study aims to add to the existing literature

of LSTM performance in forecasting asset prices along with providing a more transparent model for future studies.

Problem Statement

The problem to be addressed is the stochastic behaviors that asset returns display with non-stationary and non-linear features make price forecasting a difficult task. The problem of successful asset pricing alludes to most retail traders since various broker data shows roughly 80% of retail traders are unprofitable (Ninja, 2019). Professional money managers do not fare any better, since 50% of hedge funds close within two years of operation (Wetering, 2020). For example, supporters of the efficient market hypothesis state that predicting future asset prices with past prices is a fruitless endeavor (Mehtab, 2020). This task is even more challenging once combined with the cyclical factors within the semiconductor industry due to the focus on the SOXX index (Liu, 2006).

The literature search revealed that advancements with stochastic models such as deep neural networks can make time series forecasts, yet there has not been a dominant model that provides high accuracy and consistent forecast. As a result, time series forecasting of asset prices with deep neural networks is still a contested topic (Su, 2020). Even more so, deep neural networks such as long short-term memory networks have various complexities that make finding a decent forecasting model challenging. As for semiconductors, vector autoregression models show practical forecasts (Aubry, 2014). Yet results in other forecasting research from other industries hint at the promise of an even more accurate machine learning forecast that may translate into the semiconductor industry (Bagnall et al, 2016).

Purpose Statement

The purpose of this study is to add to the ongoing discussion that past prices aid in future price prediction, expanding the models used for forecasting in the semiconductor industry, and exploring the limits and uses of varying LSTM architectures for time series forecasting with asset prices. In particular, this paper will examine iShares PHLX Semiconductor ETF (SOXX) daily prices with long short-term memory neural networks (LSTM) for researchers and practitioners interested in financial time series forecasting with deep learning given this econometric study located virtually with Selinus University. Data will be obtained via Yahoo Finance and will be

managed with Application Programming Interface (API) calls within Python with Pandas Data Frames.

Nature of the Study

This study will be predominantly quantitative because of the nature of time series forecasting. But some qualitative approaches are applied such as action research, which allows studies created to link theory to practice to drive a change (Bhandari, 2020). In this study's case, findings from this time series forecasting model can be applied by semiconductor practitioners, researchers, or investment professionals applying to hedging or investing decisions. Also, the research presented here will add as a case study to the ongoing discussion of the grounded theory of the Efficient Market Hypothesis along with the semiconductor forecasting literature.

The design of the LSTM time series forecast will mainly comprise data collection of the daily ETF prices and analysis of forecasting quality with industry-standard metrics. As stated before, the data will be gathered from Yahoo Finance via API and stored as a comma-separated file (CSV). The analysis will occur in an open-source programming language called Python. With Python, open-sourced libraries such as TensorFlow 2 and Pandas data frames will be applied to create LSTM deep learning models that will in turn output forecasted ETF prices. The forecasted output will be analyzed by conventional machine learning time series metrics and return on investment compared to a buy and hold strategy.

Research Questions

Out of the entire scope of this study, there are three major research questions. The first question asks how well the forecasting technique performs. The second question asks how well the forecasts can perform about other strategies concerning investing decisions. The third question focuses on how much data is required if the forecasts prove to be fruitful.

Research Question 1: Does LSTM with SOXX prices offer accurate forecasting?

Research Question 2: Can LSTM SOXX price forecasts be deliver higher returns than buy and hold?

Research Question 3: How sample size or training period LSTM forecasting accuracy?

Hypotheses

The hypotheses aim to further drill down from the research questions to provide a clear demarcation for a successful or unsuccessful experiment. In this case, the study tests how well an accurate forecast is, what practical investment benefit LSTM models have against more traditional methods, and how much daily pricing data is required to maintain an accurate forecast. The first hypothesis aims to evaluate the accuracy of the overall forecast. The 50% level was chosen for practical purposes. With a model that forecast greater than 50% accuracy, a trader can take financial derivatives that offer a ratio of 1 to 1 risk-reward set up. Meaning if the trader can have predict better than 50% accuracy, they can set up a trade that offers at one unit of risk for at least one unit of reward for profit. Effectively, the trader would get odds similar to being the casino in a game of roulette. The second hypothesis targets the practical benefit of choosing a more complex model (LSTM) against a more simple strategy. The last hypothesis tests how much data is required to operate the model, since excess data leads to increased costs on operating the model is taken into a business setting.

H01: LSTM models do not produce accurate (<=50%) forecasts.

HA1: LSTM models do produce accurate (>50%) forecasts.

H02: LSTM models do not generate higher absolute returns compared to buy and hold.

HA2: LSTM models do generate higher absolute returns in comparison to buy and hold.

H03: Sample size does not affect LSTM forecasting accuracy (<= +/-10%).

HA3: Sample size does affect LSTM forecasting accuracy (> +/-10%).

Conceptual Framework

When discussing modern finance, the concept of the Efficient Market Hypothesis (EMH) tends to enter the discussion. EMH states that investors cannot gain superior returns to the market (Ţiţan, 2015). There are many positions to EMH's claims, but this study aims to test research against the idea that investors cannot gain superior returns to the market. Superior returns in this paper will refer to alternative strategies that outperform a buy and hold strategy of the same asset. The challenge to EMH for this study started from a combination of improvements in time series forecasting methods, increased computing power, and low cost to access asset pricing data.

The difficulties with time series forecasting derive from the non-linear patterns that may occur and changes to underlying fundamentals of the time series in question. One known statistical

solution to handling time series forecasting is known as Auto-Regressive Integrated Moving Average (ARIMA), in which the time series is differenced and autocorrelation with partial autocorrelation is analyzed (Siami-Namini et al., 2018). With the culmination of the three technological advances stated earlier, machine learning, specifically deep learning, methods can identify non-linear patterns and other complexities with time series prediction.

LSTM is a particular type of deep learning neural network, which is known to analyze an entire time series compared to prior discovered models (Bandara et al., 2020). LSTMs have before been used to forecast large indices like the S&P 500 and a few stocks. Prior studies have never forecasted a semiconductor-related asset index with deep learning. A comparable model was a vector autoregressive model that applied macroeconomic factors with factors indices (Aubry, 2014). Besides, prior research of LSTM models applied to financial forecasting lacks full transparency for forecast reproduction. This study aims to build off of prior research of older semiconductor forecasting models, machine learning techniques, and present findings where other studies fell short such as a detailed model architecture.

Operational Definitions

Alpha. An investment manager or investment strategy that out performs a benchmark (CFI, 2020).

API call. An endpoint to a URL that sends a request to a server for data (Rapidapi, 2020).

Application Programming Interface (API). A set of protocols, procedures, and tools that allow applications to communicate (Rapidapi, 2020).

Autocorrelation. A statistical representation of the degree of similarity over successive time periods between a given time series and a delayed version of itself (Smith, 2020).

Buy and hold. A passive investing approach in which an investor purchases and retains stocks or other forms of shares for a long time, independent of market volatility (Beers, 2020).

Deep Learning. An AI functionality that mimics the human brain's roles in the processing of data for object identification, speech recognition, language translation, and decision-making (Hargrave, 2020).

Differencing. A type of transformation that makes a time series stationarity and stabilizes the mean of the time series (Hyndman et al., 2005).

Exchange traded fund (ETF). A basket of exchange-trading shares, much like a portfolio. When the ETF is acquired and sold, ETF share rates fluctuate every day; this is separate from mutual funds that only exchange once a day after the market ends (Chen, 2020).

Long short-term memory (LSTM). Architecture of an artificial recurrent neural network (RNN) used in the field of deep learning (Siami-Namini et al., 2018).

Machine Learning. A data processing methodology that automates the development of computational models. It is a subset of artificial intelligence focused on the premise that systems, with minimal human interaction, can learn from data, recognize trends and make decisions (Siami-Namini et al., 2018).

Neural Network. A collection of algorithms that, through a mechanism that mimics the way the human brain works, attempt to identify underlying associations in a set of data (Siami-Namini et al., 2018).

(Pandas, also referred as dataframe) Data Frame. Two-dimensional size-mutable, with named axes, theoretically heterogeneous tabular data structure (rows and columns). A data frame is a two-dimensional arrangement of data, i.e., rows and columns coordinate data in a tabular format (Geeksforgeeks, n.d.).

Partial autocorrelation. A description of the relation between an observation in a time series with observations excluded at previous time periods with the associations of intervening observations (Brownlee, 2020).

Time Series. A series of values taken at successive moments (Hyndman et al., 2005).

Assumptions, Limitations, and Delimitations

Given the restrictions placed on how this study aims to apply LSTM models for SOXX price forecasting, there are inherent assumptions, limitations, and delimitations. The main assumption for this research is that past prices can aid in predicting future prices. As for limitations and delimitations, only one index is studied within a specific time period. Hence any application

for other indexes and other periods such as exogenous events like business cycles may not be fully captured.

Assumptions

The process of time series forecasting assumes that the past observed values can be used to predict future values (Hyndman et al., 2005). In this study, we assume that this pattern will apply to financial asset prices to a certain degree and that asset prices can also exhibit non-linear patterns (Bandara et al., 2020). Assumptions on the modeling perspective assume that LSTMs are less sensitive to noise since they capture patterns within the time series and preserve that knowledge through various time steps. Lastly, assumed is that data preparation before the model begins a forecast will greatly affect the predictive power (Mehtab, 2020). These LSTM nuances will be manipulated and reported with careful data preparation and model architecture setup.

Limitations

This study will include SOXX ETF daily price data from January 2009 to December 2020. The ETF is traded on the Nasdaq exchange and is composed of an index of stocks from U.S. equities in the semiconductor sector. The number of stocks with the SOXX is determined by iShares (BlackRock , 2021). Regardless of the complexity of neural networks, the forecasting model will be within the bounds of an LSTM deep learning model. Even in the unlikely event that SOXX becomes delisted, this study aims to offer further improvements and recommendations with asset price forecasting with volatile equities such as semiconductors.

Delimitations

This study would be limited to SOXX ETF and no other assets. All sampling and forecasting will be compared within the same time period as the data was gathered. This study will not consider factors outside of the price history for the forecast such as news of political events, corporate actions, sentiment analysis, etc. Only pricing data will be considered for the forecast. Also, the LSTM model will be created and operated within Python and the TensorFlow 2 package. Other LSTM models from differing languages or packages will not be covered.

Significance of Study for Applied Financial Time Series

The results of this study will aid researchers and practitioners interested in financial time series forecasting or LSTMs applied for time series modeling. Within financial time series

forecasting, the results of this study will add to the going discussion of Efficient Markets, forecasting techniques on asset pricing, and bridging machine learning with finance. Practitioners can apply the findings for engineering LSTM models about asset pricing, risk management, or alpha generation. Lastly, semiconductor business leaders may apply this forecasting research to improve their planning in the long product times in fabricating semiconductors.

Summary

Chapter 1 introduced the aim of this study, which is to apply statistical analysis and machine learning methods to forecast future prices of the SOXX, iShares PHLX Semiconductor ETF, from historical daily prices ranging from 2009-2020. SOXX was chosen due to two main reasons. First, was the ETF was only covered with momentum strategies in prior literature (Tse, 2015). Although the semiconductor industry was forecasted with other models such as ARMA, Markov, and vector autoregressive, none have applied deep neural networks for forecasting (Aubry, 2014). Second, the heightened volatility in the semiconductor industry and cyclical nature increase the forecasting difficulty. Semiconductor businesses may allocate over 13% of sales that is highly impacted on what phase the business cycle is in (Liu, 2006). The results of this research will add to the ongoing discussion in finance about efficient markets and the capabilities of LSTM time series forecasting on asset prices (Ţiţan, 2015). In particular, the accuracy of LSTMs will be examined more thoroughly along will sample sizes for forecasts and if LSTM forecasts can outperform a buy and hold strategy. From a business perspective, accurate forecasts from LSTM could save semiconductor businesses millions if not billions of dollars for semiconductor fabrication.

In further chapters, such as chapter 2, this paper will briefly review time series forecasting methods in the financial realm leading up to LSTM models. As of the last decade, there has been a large amount of literature done combining machine learning with time series forecasting (Siami-Namini et al., 2018). Chapter 3 will approach the problem if LSTM models can accurately predict future asset prices only using prior historical data. The LSTM model will first be trained with varying sample sizes. Then a separate out of sample historical period will be forecasted and compared to the actual historical data.

There are three research questions examined: do LSTM with SOXX prices offer accurate forecasting, can LSTM SOXX price forecasts be used for superior returns, and how sample size

or training period LSTM forecasting accuracy? In this study, an accurate forecast will be any predictions above 50%. Superior returns will be evaluated against the performance of the buy and hold strategy of SOXX. The question on sample size effect on forecast accuracy will be evaluated with any accuracy values greater than 10% from another sample size. All analysis will be performed with Python and data gathered via Yahoo Finance. In chapter 4, developments in the LSTM's performance and evaluations will be presented. Lastly, chapter 5 will include implications, concluding remarks, and further research opportunities with an appendix.

Chapter 2: Literature Review

Introduction

Machine learning is significant today, and the world has transformed with the introduction of artificial intelligence. The business practices in the current world are not those that were used in the past five decades. Machine learning has become a part of the organization, and leaders are working to incorporate deep learning skills into the organization while educating employees on how to use them.

Besides, the models have been used in financial forecasting. Research shows that the machine learning market is increasing rapidly. In 2017, it made approximately $1.4 billion, and it's expected to reach about $8.8 billion in 2022 (BCC Publishing, 2021). The growth rate is around 43.6% from 2017 to 2022. The machine learning market is oversaturated, and the demand for the ML profession is high in different companies. Machine learning features four main areas: algorithms, training data, deep learning, and supervised and unsupervised learning. Many businesses are investing heavily in ML technology and using it to improve their operations (Pan, 2018). Spending on ML is estimated to be $100 billion annually by 2025, showing an annual growth rate of approximately 40%.

In 1959, Arthur Samuel introduced machine learning, an American interested in artificial intelligence and computer gaming. The introduction mainly dealt with pattern classification and the interest in pattern recognition until the 1970s, when teaching strategies were introduced in 1981 (Kewat et al., 2017). Tom Mitchell expanded the definition of algorithms where machine learning focused more on operational purposes than cognitive terms. The innovation has continued up to the modern machines, which are recognized as having two main objectives. The first step is to classify the data based on the models developed. The second aim is to predict the future outcome based on the presented model. Algorithms used computer vision of moles combined with supervised learning to train them to classify cancerous moles. Machine learning was invented by the passion of artificial intelligence users when some tried to approach a given problem using different symbolic methods known as neural networks (Dutta, 2014). In the process, they developed generalized linear and probabilistic models commonly used in the medical industry. Today, machine learning is used in the stock market to inform investors about the future through predictions. The objective of machine learning has changed from artificial intelligence to focus on solvable problems of a practical nature.

Overview of Statistical Methods of Financial Forecasting

Financial forecasting is a sensitive area that needs good skills and the application of a quality algorithm that will be of value in the market. The stock market contains buyers and sellers, and each is looking for quality, which means when the correct software predicts the accuracy of the values, it will attract many sellers or buyers. There are many algorithms in the market to help in the prediction of different stock prices. Some products are challenging in price prediction; therefore, they require enough research before engaging in any of the algorithms to avoid losses. The accuracy of their predictions drives investors in the stock market (Jackson et al., 2018). Therefore, a subset of investors is willing to spend on a particular algorithm software to improve accuracy in their returns. The stock market is challenging, and individuals can sustain a competitive advantage by using the correct algorithm and prioritizing accuracy. Financial forecasting uses different algorithms because manual prediction is a difficult task.

There are different types of financial forecasting, such as the straight-line forecasting method, commonly used when the company's growth rate is constant. This involves basic math with historical data and needs growth prediction, so it's essential to select the best algorithm to serve the task. Another method is the moving average forecasting method; this calculates the average performance in particular metrics over a short time frame in terms of days, quarters, and months. It is not reliable over more extended periods, such as years. Simple linear regression is another forecasting approach. This method is trustworthy for charting a trend line depending on the connection between dependent and independent variables. It shows the changes in the Y-axis (dependent variable) to the changes in the X-axis (explanatory variables) and later creates a graph line (Altan & Karasu, 2019). Lastly are the multiple linear regression forecasting methods. It uses more than two independent variables to project. It creates a model of the relationship between the independent explanatory variables and the outcome. The investor should select the best algorithm for the forecast using different methods to be successful. The commonly used algorithms are ARIMA, GARCH, SVM, XGBoost, RNN, and LSTM.

Financial Forecasting with Traditional Econometric Methods

Autoregressive integrated moving average (ARIMA)

The ARMA model is a frequently used time series analysis tool. ARIMA is an Autoregressive Integrated Moving Average based on the ARMA Model. The ARIMA model is

unique in that it transforms non-stationary data into stationary data before using it. The ARIMA model is often used to forecast linear time series. The ARIMA method is very flexible in identifying, parameterizing, and predicting univariate time series models.

In their article, Kumar and Thenmozhi (2014) carried out research to establish and identify the most suitable hybrid model for predicting stock index returns. The authors developed three distinct hybrid models, merging ARIMA and non-linear models like artificial neural networks (ANN), support vector machines (SVM), and random forest (RF), to be utilized in forecasting stock index returns. The performance of the three models is compared against that of ARIMA-SVM, ARIMA-ANN, and ARIMA-RF. The competing models are then assessed based on trading performance and statistical metrics through a specific trading strategy. The authors conclude that the hybrid ARIMA-SVM model is best suited to forecasting stock index returns due to its high level of accuracy and improved returns.

To determine the Shanghai securities composition stock index, Du (2018) utilized the ARIMA model and combined it with several nonlinear models to ensure better accuracy in forecasting and improved results. The author compared two models in his study: the ARIMA model and the BP neural network technique. In his conclusion, the author describes the ARIMA-BP neural network as a superior model compared to the BP neural network on the basis of accuracy in forecasting.

Wang & Guo (2020) use a hybrid model called DWT-ARIMA-GSXGB to forecast stock prices. The authors utilize the discrete wavelet to divide the data into error and approximation parts. They incorporate four models into their study: ARIMA (0, 1, 1), ARIMA (1, 1, 0), ARIMA (2, 1, 1), and ARIMA (3, 1, 0) to handle partial data, while the improved Xgboost model controls the error in partial data. Prediction results from the various models are merged using wavelet reconstruction. Based on the study research, the authors conclude that the DWT-ARIMA-GSXGB has more minor errors compared to other models. In addition, they also outline that the hybrid model has better approximation ability and can be used to predict the stock opening price index.

In the article "A Prediction Approach for Stock Market Volatility Based on Time Series Data," the author’s primary aim is to design a well-structured forecasting model for two indices on two distinct Indian markets, the Sensex and the Nifty. The authors apply a logarithmic transformation to the data, and two AFRIMA models are evaluated to predict the two indices. The authors chose two principal ARIMA to represent the models (0, 1, 0) with a drift to guarantee

accurate results and conclusions. In their conclusion, the authors outline that a well-chosen AFRIMA model is accurate enough to forecast time series data. The author's conclusions are formed under the predicted values of the incorporated models, whose deviation margin averaged 5% of the actual outcome (Idrees et al. 2019).

In the article "ARIMA: An Applied Time Series Forecasting Model for the Bovespa Stock Index," the authors used MAPE to dictate the most accurate model among several forecasting models that would be most accurate in forecasting the Brazilian stock index - Bovespa. In the suggested models, an autoregressive model is compared with two distinct exponential smoothing models and an ARIMA (0, 2, 1). In the article, when designing the ARIMA model, the Box-Jenkins methodology is followed. The authors determined that an AR (1) was the most precise model based on the data because it had the lowest out–of–sample MAPE. They also concluded that for the Bovespa model, an AR (1) was a suitable tool to forecast the index (Junior et al. 2014).

In their research, Jackson et al. (2018) utilized the Box-Jenkins methodology to construct a seasonal autoregressive integrated moving average, also known as SARIMA. The main aim of building the model was to forecast the short-term power flows of transmission entities in the United States. According to the authors, a SARIMA is an advanced AFRIMA that should be used in the event of a seasonal pattern in the particular time series intended to be forecasted. The study concluded that by administering the Box-Jenkins methodology approach, building a model compatible with the data of the selected model in the research is achievable, and the model would provide precise forecasting for the time series. In the scenario of a seasonal pattern in the time series, a SARIMA would be ideal for forecasting the time series. In addition, the authors also outlined that a SARIMA would be more accurate in short-run forecasting compared to the long-run.

Generalized autoregressive conditional heteroskedasticity (GARCH)

Volatility is a critical element in finance. It is essential in disciplines such as risk management, portfolio modification, and security pricing. Volatility is a fundamental aspect of the Black and Scholes formulation. The error variance in financial determination cannot be constant; instead, the series displays volatility in clustering. (Reider, 2012). Clearly, heteroscedasticity pervades financial time sequences. Therefore, future volatility is an essential determinant for

financial investors. That is why financial prototypes are analyzed and factored in by their capacity to issue accurate financial forecasts. (Andersen et al., 2013).

When assessing the accuracy of models, analyzing previous research is necessary for the application of various evaluation measures. The most common applied measures are the Mean Absolute Percent Error, Mean Square Error, and the Root Mean Square Error. When determining which model is sufficient, it is inaccurate to weigh which model is dominant over the other in relation to the evaluation measures. The appropriate approach to resolve the issue is to first calculate the average figures of some of the statistical measures and then evaluate the forecast models based on the data and requirements factored into the forecast analysis.

The GARCH volatility models are an essential toolkit for reasonable asset valuation and financial risk management. Based on the input that Engle and Bollerslev provided, massive econometric research has played a substantive role in volatility assessment and prediction. Heston and Nandi create a specific GARCH measurement that produces a systematic solution and issues a pragmatic analysis of the model. The volatility model validates the addition of the leverage effect, and volatility grouping is influential in enhancing financial estimation performance.

Regardless of whether the GARCH model does not issue a theoretical description of volatility or it does, it only issues limited information about the volatility generating process. Based on this, early attempts to issue a theoretical explanation of the volatility process included the error distribution hypothesis that Clark and Epps advanced. The variation of stock returns while using the MDH at a given time is proportionate to the frequency of information arrival, resulting in volatility clustering that is factored by the information arrival frequency. All the trading parties receive price indicators concurrently, which leads to a new equilibrium.

Harvey and Sucarrat's (2014) research on GARCH theory models, their main idea on the asymmetric model was based on models' leverage effects, which affected the level of either good or bad news in equal measures and had distinct effects on the market’s volatility. The EGARCH model captures the asymmetric properties of stock return volatility. In their conclusion, they pick the EGARCH model as the best forecasting model for stock volatility due to three parameters.

Liu et al. (2016) validate that the GARCH model is a convenient model for predicting stock market volatility. Using the GARCH model, Liu et al. studied the prediction of stock market precariousness in China. The study realized the forecast findings by implementing the GARCH-SGED model were more feasible than using the GARCH-N model. This indicated the importance

of tail thickness and skewness in the conditional assessment of returns for evolving financial markets. The GARCH-SGED model issues fewer mean absolute percent errors and mean square errors as compared to when using the GARCH-N model when studying the Chinese stock market.

Christoffersen et al. (2013). They also examined the predictability of stock market precariousness in Israel while using the GARCH model. The result would indicate the skewness of the GARCH model with fat-tailed thickness increases the accuracy and general estimation for measuring conditional variance. The forecasts tested the GARGH model, confirming its reliability over the EGARCH, GJR, and APARCH models. GARCH's reliability in forecasting the Israeli stock market was considered accurate as compared to other models based on its ability to eliminate significant errors in forecasting stock market volatility.

Vošvrda and Žıkeš (2004) utilized the GARCH-time model to portray the volatility of stock returns in the Hungarian and Czech markets by utilizing their weekly data, which was recorded from 1996 to 2002. In addition, the two used index series as a replacement for their returns. In the aftermath of deriving the ARCH test from the Hungarian index, the authors concluded that tests in both markets showed a conditional heteroskedasticity in the approximated figures.

Angabini and Wasiuzzaman (2011) assess the forecasting performance of the Malaysian stock market using several models, including GJR, EGARCH, and GARCH, on the financial crisis to show how volatility changed in the stock market during the global financial crisis that occurred in 2008. With all markets hit by the financial crisis, the authors concentrated on the Kuala Lumpur Composite index. The two came to the conclusion that the Kuala Lumpur Composite Index exhibited asymmetry, leptokurtosis, and the leverage effect. In addition, they outlined that there was a considerable increase in the volatility and leverage effect in the market caused by the financial crisis experienced in a short period of time. The authors also compared how the EGARCH and the GJR evaluated the change in volatility. The authors noted that the two models produced similar results in their results, which showed an increase in volatility from 11.5% to 18.5% for the Kuala Lumpur composite index.

Regardless of these findings by Angabini and Wasiuzzaman, GJR and EGARCH made up of their nonlinear asymmetric extensions, the two were outdone by the GARCH model, which, according to the author’s evaluation, measures the most accurate volatility closest to the realized volatility of the index. This is contrary to the findings of Hassan (2007), whose findings portrayed

the GJR as the best-suited model for forecasting the Malaysian stock market under normal conditions.

Previous research aimed at evaluating the best in-sample fit model proved that it does not always produce the best out-of-sample predictions. Mantalos (2013) outlined that it was statistically necessary to specify the conditional mean process, lag order, and distribution of the error to reflect on the historical movements of the series. The author concluded that the minor lag order was most suitable to capture changing volatility, hence providing accurate results. Furthermore, he highlighted that when forecasting the volatility of any index stock market, the GARCH model is usually utilized due to its small lag order.

In their research to determine volatility in the Shanghai Composite Index and Shenzhen Composite Index returns, Wang et al. (2021) conducted an empirical analysis using the general autoregressive conditional heteroskedasticity (GARCH) model. The authors established an autoregressive moving average (ARMA) model with a time distribution for selected sample series used to compare the models on different distributions and orders. They further recommended a threshold-GARCH (TGARCH) and an exponential-GARCH (EGARCH) to be utilized in collecting information on the index. The authors also evaluated the prediction results and error degree of other models based on mean squared error (MSE), mean absolute error (MAE), and rot-mean-squared error (RMSE). Derived results indicated that ARMA (94, 4) and GARCH (1, 1) outperformed other models in predicting the Shanghai Composite Index return series. In the Shenzhen Component index case, ARMA (1, 1) and T GARCH (1, 1) depicted the most suitable forecasting performance compared to all other models.

Lim & Sek (2013) carried out a study to determine the best-suited model to forecast the Malaysian, Philippines, Singapore, and Thailand stock markets. The authors used several models in their research that revealed the ARCH model was superior compared to other models in capturing the stock market volatility in Malaysia and Singapore markets. The study further revealed that the TGARCH and EGARCH models were more suitable for the Philippines market. Furthermore, the two concluded that the asymmetry of the market returns was not crucial in the selected markets forecasted by EGARCH and TGARCH models.

Dutta (2014) carried out research on the exchange rate parties of two countries, the United States and Japan, for a period of 12 years from 1 January 2000 to 31 January 2021. The author estimated the collected data using both symmetric and asymmetric GARCH models, and his results

showed that positive shocks were standard compared to negative shocks in both countries' return series. In addition, Dutta concluded that asymmetric tests for volatility tests showed a sizeable effect on stock news. Furthermore, he outlined that the market risks and return index would be different from one country to another due to the different market backgrounds.

Panait & Slavescu (2012), in their effort to predict the stock volatility index of seven Romanian companies listed in the Bucharest stock market, utilized their daily, weekly, and monthly data from 1997 to 2012 in their research. The two incorporated the GARCH-in-man model to compare volatility among the companies in three stages. The results from the study established persistency and consistency in the daily returns, contrary to the weekly and monthly series, which were expected. In addition, they conclude that the utilized GARCH model failed to confirm that arise of future returns is caused by an increase in volatility.

Financial Forecasting with Machine Learning

For many years, financial time series forecasting and its related applications have been the subject of substantial research. When machine learning (ML) began to gain popularity, financial prediction apps based on soft computing models became accessible as a natural consequence. It would be good to quickly describe the current surveys covering financial time series prediction studies based on machine learning in order to obtain historical context, even if our emphasis is on deep learning (DL) implementations of time series prediction studies for financial time series.

I did not include any survey articles in our analysis that were focused on particular financial application areas other than forecasting studies since I felt they were redundant. But I came across certain review papers that covered not just financial time-series research but also other financial applications, which I found to be problematic. I opted to add those pieces because I wanted to ensure that our coverage was as extensive as possible.

Examples of the aforementioned publications are given in the next section of the website. Stock market forecasting, trading system development, and practical examples of forex and market forecasting applications using machine learning models such as Artificial Neural Networks (ANNs), Evolutionary Computation (EC), Genetic Programming (GP), and Agent-based models were all covered in books that were recently published.

There were also other surveys from previous journals and conferences that were included. A study of financial prediction and planning research, as well as other financial applications

employing different Artificial Intelligence (AI) approaches, including artificial neural networks (ANNs), expert systems, and hybrid models, was conducted by other researchers. In addition, the authors compared machine learning approaches in several financial applications, including stock market prediction research. Soft computing models for the market, forex forecasting, and trading systems were all investigated in the paper. Mullainathan and Spies conducted an assessment of the prediction process in general from an econometric standpoint in their paper.

Several survey articles focused on a single ML model in particular, which was also presented during the conference. Despite the fact that these publications concentrated on a single approach, the implementation areas often included a wide range of financial applications, including financial time series forecasting research. EC and ANN were the soft computing approaches that drew the greatest attention overall among the participants.

Chen produced a book on Genetic Algorithms (GAs) and Genetic Programming (GP) in Computational Finance in preparation for the EC study. Later, Multiobjective Evolutionary Algorithms (MOEAs) were intensively studied in a variety of financial applications, including the prediction of financial time series, among others. Meanwhile, Rada examined EC applications, as well as Expert Systems for the financial investment model, under Rada's supervision.

Li and Ma examined implementations of artificial neural networks (ANNs) for stock price forecasting and other financial applications as part of their ANN research. The authors of this paper conducted a review of several ANN implementations in financial applications, including stock price predictions. In a recent review, Elmsili and Outtaj included ANN applications in economics and management research, as well as economic time series forecasting, in addition to other topics.

Additionally, there were various text mining surveys that were geared at financial applications (and specifically, financial time series forecasting). To make predictions about the market, Mittermayer and Knolmayer analyzed several text mining methods that extract the market's reaction to the news. In their review, the authors concentrated on news analytics research for the prediction of anomalous returns for trading techniques, which they found to be particularly useful. Nassirtoussi and colleagues analyzed text mining research that was conducted for the purpose of stock or currency market prediction. Text mining-based time series forecasting and trading methods based on textual sentiment were also investigated by the authors of this paper. In the same way, Kumar and Ravi examined text mining research for the prediction of FX and stock

market prices. Xing and colleagues have conducted a survey of natural language-based financial forecasting research.

Finally, there were survey papers that were focused on specific financial time series forecasting implementations, which were called application-specific survey studies. Stock market forecasting was the study that drew the most attention out of all of them. Many surveys for stock market forecasting research using various soft computing technologies have been released at various periods throughout the last few years. As previously stated, Chatterjee and colleagues and Katarya and Mahajan focused on artificial neural network-based financial market prediction research, while Hu et al. worked on EC implementations for stock forecasting and algorithmic trading. The researchers conducted a study of currency prediction studies employing artificial neural networks (ANNs) and other soft computing approaches in a separate time series forecasting application.

Despite the fact that several surveys exist for ML implementations of financial time series forecasting, DL implementations have not yet been thoroughly studied despite the fact that there have been many DL implementations in recent years. As a result, the poll was primarily motivated by this need. At this point, we'd want to go through the several DL models that have been utilized in financial time series forecasting research in general.

Support Vector Machine Theory

Support vector machines play a significant in machine learning, and the devices are connected to learning algorithms for analyzing the data. SVM is useful in data classification and regression analysis. The approach was developed by bell laboratories and was first used in 1992. The inventors thought it was the best way to create non-linear classifiers using the Kernel trick to the maximum margin hyper-lanes (Tao et al, 2018). The commonly used software is a soft margin which was built in 1993 and became officially in use in 1995.

These machines are from the family of generalized linear classifiers. They are interpreted as the perceptron extension. These machines have the property of minimizing the empirical classification error and increasing the geometric margin, and therefore they are also known as maximum margin classifiers. Chen et al (2017) note that the support vector machine is one of the essential prediction tools used in the statistical framework. The theory training algorithm creates a model which has one example for a single section or the other. SVM is recognized as a non-probabilistic binary linear classifier in addition to other methods such as Platt scaling. The support

vector machine positions training examples to point in space to utilize the width of the gap between two sections. The new models are mapped in that space and assumed to belong in the section where the gap falls. The margins support vectors are significant because they are hard to classify considering they are points within the groups which is closest to the other group. It's a prediction tool that uses machine learning theory to produce accuracy and avoid overfitting of the data.

Varatharajan et al (2018) state that the support machine theory approach helps perform linear classification. At the same time, SVM performs in non-linear classification using an approach called the Kernel trick, which works effectively. Unsupervised learning is needed in areas where the data is unlabeled, and this helps in finding the natural clustering of data to various groups. Support vector machine is among the preferred clustering algorithms, and it's helpful in industrial applications. Classification of data is essential and regression; the main goal is to find the effective hyperplane that separates the data points. This is a robust algorithm that many scientists have used. Creating lines that go to various classes and avoiding splitting observations from the same class while keeping the considerable distance possible from the classes. The primary fundamental of SVM is margins and hyperplanes.

Kalantar et al (2018) note that the SVM learning algorithm finds the hyperplane, which maximizes the margin; by doing that, it creates a reasonable boundary that splits the classes. The hyper-plan is recognized to work effectively as a decision boundary. The machine has gained popularity in the world and is currently used in learning research.

Further, the SVM has gained popularity in performing sensitive features such as empirical performance. The SVM foundation was developed by Vapnik and had many features useful in research. The formulation of the approach uses a superior principle known as structural risk minimization (Deng et al, 2019).

Another principle is empirical risk minimization used by the conventional neural network; however, it is more minor superior. The SRM has a significant role in minimizing the upper bound of the unexpected risk, and the ERM minimizes the errors in the training data. This describes why the SVM is unique and more preferred due to its high ability to generalize. This is the goal in all statistic learning to achieve the best and solve the classification problems effectively. However, in the current days, it has extended, and it can solve regression problems.

In the old days, the machines used in those days focused on learning representations of simple functions. That means the main aim was to output a hypothesis that did the correct

classification of the training data. Also, the early learning algorithms were designed to find an accurate fit for the data. Generalization is essential, and the hypothesis can effectively classify the data that's not in training. The SVM shows a good performance of less over-generalization when the neutral network is overgeneralizing easily. To create a hyperplane, the SVM uses an iterative training program used to minimize the error function. Based on the error function, the SVM model is categorized into four main groups. First is C-SVM that's type 1; second is nu-SVM type 2 classification; third is epsilon-SVM, a regression type 1; and finally, is nu-SVM, a regression type 2.

Kernel tricks are significant in this SVM, and they are used in non-linear, mapping the input data to a higher-dimensional space. For instance, the linearly separable uses this concept. When the data is transformed into feature space, it becomes easy to define the similarity measures based on the dot products. When the feature space is selected effectively, then recognizing the pattern becomes easy. The Kernel trick has steps to follow for successful results; its performance allows SVM to create non-linear boundaries. When expressing the algorithm in the Kernel trick, it should use only one inner product of the data sets. The concept is known as a dual problem.

Besides, the original data is passed through the non-linear maps to form new data in line with the new dimension. A pair of wise products are added from the original data dimension to every data vector. The dot product of the data can be represented when non-linear mapping is done on them. This is a significant kernel function that has made learning machines more effective in high demand.

The complexity of the Kernel function impacts the normal functioning of the datasets. The SVM supports the idea of controlling complexity. However, it doesn't tell how these parameters are set, and determining these parameters is done by applying cross-validation on a particular dataset. In statistical learning, the theory is a practical approach designed to provide frameworks used in studying relevant issues to gain knowledge and make decisions and predictions from a set of data. It supports choosing the hyperplane space so that it closely represents the underlying function in the target space. In this theory, the problem of supervised learning is solved using a specific formula.

Support Vector Machine Forecasting Research

Financial time series forecasting is a challenging application in the present days of modern time series forecasting. The ability of SVM to solve non-linear regression estimation problems makes it more successful and reliable in time series forecasting (Altan & Karasu, 2019). Financial series forecasting has not found a tool that can capture the financial market price of the future and the past. The financial time series is classified into two main parts that are multivariate and univariate analysis. Multivariate shows any indicator and its relation to the output were direct or indirect. Whereas the univariate input variables are limited to the time series forecasted. In this concept univariate is commonly used with the autoregressive integrated moving average method. On the other hand, multivariate depends on a lot of information such as technical indicators, Intermarket indicators, and time series being forecasted, they are combined to serve as predictors. This approach is mainly used with neural networks. This idea of a generalization of the neural network has been a concern with researchers.

SVM has innovated a novel approach with the aim of improving the generalization experienced in neural networks. The SVM has improved and with the launching of the insensitive loss function it has created room for it to solve non-linear regression problems (Calvi et al, 2019) SVM is more than the traditional machines, it uses structural risk minimization different from the old empirical risk minimization principle. The SVM principle aims at minimizing the upper bound of generalization error instead of minimizing the training error. When this is done it leads to a better generalization instead of conventional techniques. The SVM is good in prediction and gives good accuracy, it performs prediction faster compared to other algorithms. They use less memory because of the subset of training points in the decision phase. The system works effectively with a clear margin of separation and high dimension space.

Making a financial decision in the world is important, however, it depends on the approach used in making the same decision. The financial time series prediction has noisy data and non-stationary information and that's what makes it more important to use SVM. Prediction of stock market indices is a place of interest since the day the stock market was launched. Researchers have come up with motivational ways how to predict prices, and the effective way is to implement superior systems such as SVM that will bring more returns. SVM is useful in regression for financial forecasting, and it introduces an alternative to the loss function (Jaramillo et al, 2017).

The loss function can be designed to have distance measures. The regression can be either linear or non-linear, and the kernel function can be applied to address the dimensionality of the curse.

The successful financial series prediction is based on the following factors such as first; the last price of the trade performed during the day, second; the highest and the lowest traded price, and finally; the total number of goods sold during the day. These conditions are better handled with SVM considering the nonlinear problem and the uncertainty (Khairalla, & Ning, 2017). The SVM is good in prediction and Comparing SVM to other classifiers shows that the SVM is superior to other classifiers. Classifying and predicting data is expected in machine learning and the same concept is applied in forecasting. When a given data point each belongs to one of the two classes, the aim will be to decide which class a new data point will be in. in the SVM, the data point is viewed as p dimensional vector and knows whether the point can be separated with (P-1) hyperplane dimension; this is known as a linear classifier. Several hyperplanes classify the data, and the best choice represents the margin between the two classes. A maximum margin hyperplane exists where the distance to the nearest data point is maximized on each side. The classification of the task done by the SVM technique involves training and testing data consisting of data instances.

The main aim of SVM is to design a method that predicts the target value for data instances in the testing set that is given in the attributes only (Kewat et al, 2017). The known label in supervised learning is essential in informing whether the system is performing well or not. The information is meant to help the system act in the right way and validate the system's accuracy. The non-linear classification was created in 1992 and used the Kernel trick concept, and Aizerman and other researchers originally proposed the ideas. The purpose was to find a way of having a maximum margin hyperplane. This concept uses the same algorithm; however, each dot product is replaced by the Kernel function. This gives room for the algorithms to fit in the maximum margin hyperplane in the transformed feature space. Xiao et al (2020) state that the transformation can be non-linear and the transformed space with high dimensional. The classifier is hyperplane; the difference is that it features space in the transformed, and the original input space is non-linear. The generalization error of the SVM increases when working with the higher dimensional feature space. The presence of the kernel function has raised the SVM advantage in financial forecasting in the stock market.

XGBoost Theory

XGboost means extreme gradient boosting, and it uses a more improved regularization that's L1 and L2 to improve the capability of model generalization. It's a gradient boosting method that employs an accurate approximation intending to develop an excellent tree model. XGboost is known for its speed and performance that has dominated the applied machine learning and Kaggle competitions for structured or tabular data (Mitchell & Frank, 2017). The software is available; it can be downloaded and installed on machines. The system supports several interfaces such as Julia, command-line interface, C++, the python interface, an approach in sci-kit learn; the R interface, which is a model in caret package, Java, and JVM languages and platforms such as scala and Hadoop.

The software is mainly focused on computational speed and model performance, for instance, few frills. The machines have many improved features. The model is helpful to the features R implementation, and also sci-kit learn. The primary gradient boosting supported entails gradient boosting, stochastic gradient boosting, and regularized gradient boosting. The system has many features which provide room to be used in the computing environment.

The features such as parallelization involve tree construction used in all the CPU cores at the time of training (Li & Zhang, 2020). Another feature is distributed computing useful in training large models where cluster machines are used. Out-of-core computing is used in massive datasets which can't fit into the memory, and finally, cache optimization of structures in data and algorithms to maximize the use of hardware.

The implementation of algorithms was done to create efficiency in computing and memory resources. The goal of the design is to utilize the available resources for training the model. The primary features of algorithms implementation entail sparse awareness. It's the implementation that automatically handles the missing data values (Pan, 2018). Another is block structure that is meant to support parallelization of the tree construction and, finally, continuous training to boost the existing fitted model in the new data. Using XGboost is effective in the learning machine it has high speed and achieves the model performance. In speed, the XG boost is fast and more efficient than other implementations of gradient boosting. The XG-boost is commonly used; it is fast, has a good memory, and has accuracy. In the model performance, it dominates the structured and tabular datasets.

XGBoost is a practical approach where the new models are added to correct the mistakes of the existing models. This model is added in sequence until there is no other improvement made. XGboost uses a gradient boosting decision tree algorithm with different names such as multiple additive regression trees, gradient boosting machines, and stochastic gradient boosting. In this method, new models are designed to predict errors of the previous models and then compiled to get the final prediction. From the name “gradient boosting," the algorithm's gradient descent is used in minimizing the loss when new models are added (Dong et al, 2020). The method is useful in classification and regression to predict the model problem. In applications, decision trees are helpful in grouping units of data using questions.

Each question in the decision tree will deliver a smaller group of units. The grouping is done to recognize the units with resembling characteristics with respect to the outcome variable. A single question asked in every decision node has only two possible choices (Zhang, et al2019). Besides, at the bottom of every decision tree, there is a single possible decision. Every possible decision will automatically lead to a choice, and some decisions lead to a choice sooner than others. These are tree-like graphs, and the XGBoost uses classification and regression trees.

The XGBoost handles the missing values present in the data set. Therefore, in data wrangling, an individual doesn't need to do a separate treatment of the missing values; the reason is that the XGboost is in an excellent position to handle the missing value effectively. XGboost delivers an accurate approximation, and it uses the strengths of the second order of derivative L1 and L2 regularization and parallel computing (Mo et al, 2019). It's a popular algorithm due to its features and is more regularized. The second order of gradient it uses provides information on the gradient direction and how it can get the minimum of the loss function.

XGBoost Forecasting Research

XGBoost is used directly for regression predictive modeling. Research shows that the XGBoost is useful in machine learning, and it has become a preference for many training pieces. Their primary function is to minimize the loss function. They are essential in improving the performance of the algorithms by using ensemble learning; they are boosting algorithms. Due to its regression predictive model the machine is highly preferred in time series forecasting (Zheng et al, 2017). The XGboost manages the numeric vectors with the characteristics of the Santander dataset. This algorithm works well when large trees are created and combined to form an excellent

predictive model. The basis of the performance of this model depends on the selection of the correct parameters.

The machine has the ability to predict numerical values such as the number of dollars or height. From history, The XGboost started as a terminal application that was configured using the libsvm configuration file. It became famous in the ML competition, and now it has a package of several implementations such as Java, Perl, Julia, and other languages.

The idea attracted many developers, making it famous in the Kaggle community, where it was applied for several competitions. Presently, it has more features and packages and thus making it more efficient to use (Wang & Guo, 2020). The Scikit-learn in python programing language was developed in 2007 and has been in the market for a decade. This is one of the most commonly used machine learning libraries. It's written in python, python, C++, and C. It uses Numpfy for higher performance in linear algebra and array operations. The

XGboost is a dominating competitive machine and is important in financial prediction. Kaggle competition has attracted thousands of teams and individuals to public datasets and code snippets. There are several researchers have documented the performance of the Kagle competition, and XGboost has emerged among the best. Paliari et al (2021) note that the most successful algorithm that wins in the competition is based on various factors before it takes the top position. The most trending algorithm is the gradient boost machine and the neural network, and for the past five years, it has led to competition. XGboost has won the competition severally due to its features, and besides, it is scalable and considers accuracy in implementing gradient boosting machines. XGboost pushes the limit of the computing power to the booted trees algorithms so that it is assumed that it was created mainly for XGBoost performance and speed. Its property system-wise allows portability and flexibility and thus has a wide range of computing environments. The block structure's presence helps parallelize tree construction and its bale to fit and enhance the new data in the training model. The XG boost is well structured, and it doesn't sacrifice speed over accuracy; it balances all its operations for better results (Gumelar et al, 2020). Their effectiveness has made it feature well in financial time series forecasting.

Managing time series forecasting involves solving regression, classification, and ranking. Users can also use it to predict financial problems in the stock market. The reason for this is that it's portable and runs smoothly on windows, OS and Linux. This makes it more preferred over other algorithms. In terms of languages, it supports all the programming languages such as python,

C++, R, Java, Julia, and Scala (Jabeur et al, 2021). Cloud integration works well with ecosystems, and it supports AWS, Yarn clusters, Spark, and Flink. Based on the system optimization, research shows that the algorithm uses several optimizations such as parallelization; this approach uses the sequence tree building with a parallelized implementation. This is possible due to the interchanging nature of the loops employed for building base learners and the outer loop, which enumerates the leaf node of the tree, and the second inner loop that calculates the features. Another approach is tree pruning, which is based on tree splitting within the GBM framework, and it relies on the negative loss criteria at the point of the split (Yuan et al, 2021). Hardware optimization is another optimization specifically designed for the hardware resources to make them more effective and efficient. This process successfully uses cache awareness and allocates internal buffers in every thread to store the gradient's statistics.

The algorithm works well in financial time series forecasting because of its improved gradient boosting machine framework, which works through system optimization and enhances algorithmic. Generally, this approach has played a significant role in classification, and many people in the stock market have used the concept to get desired results. When it comes to machine learning, selecting the best algorithm is essential if the user wants accuracy.

Improvements with Deep Learning

Deep learning (DL) is a type of artificial neural network (ANN) that consists of multiple processing layers and enables high-level abstraction to model data. The key advantage of DL models is extracting the good features of input data automatically using a general-purpose learning procedure. Therefore, in the literature, DL models are used in lots of applications: image, speech, video, audio reconstruction, natural language understanding, sentiment analysis, question answering, and language translation. The historical improvements on DL models are surveyed.

For more than 40 years, financial time series forecasting has been a hot topic among machine learning researchers. In recent years, the advent of DL models for financial prediction research has given the financial community a much-needed boost, as seen by the influx of new papers in the field. The superior performance of DL models over ML models is the most appealing feature for finance researchers. New deep learning approaches will be presented when more financial time series data and other deep architectures become available. In our survey, we

discovered that DL models performed far better than their ML counterparts in the great majority of trials.

There are many types of deep learning models described in the literature, including the Deep Multilayer Perceptron (DMLP), RNN, LSTM, CNN, Restricted Boltzmann Machines (RBMs), DBN, Autoencoder (AE), and DRL. It has been well acknowledged in the literature that financial time series forecasting is mostly a regression issue. However, in the area of trend prediction, that employed classification models to solve financial forecasting difficulties and were successful. Different DL implementations are offered, as well as the model options that they use.

Deep multilayer perception (DMP)

DMLPs were one of the earliest artificial neural networks to be built. The distinction between DMLP and shallow nets is that DMLP is composed of more layers. DMLP models are composed primarily of three layers: the input layer, the hidden layer, and the output layer. Specific model topologies may change based on the needs of the issue being addressed. The hyperparameters of the network are the number of neurons in each layer and the number of layers in the network as a whole. As a rule, each neuron in the hidden layers contains three terms: an input (x), a weight (w), and a bias (b). Aside from that, each neuron has a nonlinear activation function, which results in a cumulative output of the neurons that have come before. Nonlinear activation functions are classified into many categories. The nonlinear activation functions sigmoid, hyperbolic tangent, Rectified Linear Unit (ReLU), leaky-ReLU, swish, and softmax is the most often used nonlinear activation functions.

DMLP models have begun to arise in a number of different application areas. Depending on the needs of the situation, there are benefits and downsides to using a DMLP model. Through the use of DMLP models, issues such as regression and classification may be handled by modeling the data that was provided. However, owing to the fully linked nature of the model, if the number of input features is increased, the parameter size in the network will rise in proportion, resulting in decreased computing speed and higher storage requirements. Different sorts of Deep Neural Network (DNN) approaches have been presented in order to address this problem. Classification and regression operations may be carried out considerably more efficiently with the help of DMLP.

The backpropagation method is used to accomplish the DMLP learning step. When errors occur in the neurons in the output layer, the amount of error transmitted back to the preceding

layers is calculated. Finding the optimal parameters for neural networks is accomplished via the use of optimization methods. They are used to update the weights of the connections between the layers, which are made between the layers. There are a variety of optimization methods being developed, including Stochastic Gradient Descent (SGD), SGD with Momentum, Adaptive Gradient Algorithm (AdaGrad), Root Mean Square Propagation (RMSProp), and Adaptive Moment Estimation (ADAM). Progressive descent is an iterative approach for finding the optimal parameters of the function that reduces the cost function to its smallest value. SGD is an algorithm that, for each iteration, picks a small number of samples from the whole data set rather than the entire data set. The SGD with Momentum technique speeds the gradient descent method by remembering the update in each iteration. AdaGrad is a modified SGD method that outperforms the regular SGD algorithm in terms of convergence performance. RMSProp is an optimization technique that allows for the customization of the learning rate for each of the parameters in the optimization problem. In RMSProp, the learning rate is divided by a running average of the magnitudes of previous gradients for that weight, which is a constant across time. ADAM is an upgraded version of RMSProp that use running averages of both the gradients and the second moments of the gradients as well as the second moments of the gradients. The RMSProp (which performs well in both online and non-stationary environments), as well as the AdaGrad, are combined in ADAM (works well with sparse gradients).

The influence of the backpropagation is carried over to the layers that came before it. When the impact of SGD progressively diminishes as the influence propagates through the early layers of the network during backpropagation, this is referred to as a vanishing gradient issue in the literature. As a result, updates between the early levels are no longer accessible, and the learning process is brought to an end. The vanishing gradient issue is caused by the large number of layers in a neural network, as well as the rising complexity of the network.

The hyperparameters of the networks, as well as the technique of adjusting these hyperparameters, are significant considerations in the DMLP. Hyperparameters are network variables that have an impact on the network's design as well as the performance of the networks they touch. These parameters include the number of hidden layers used, the number of units used in each layer, regularization techniques (dropout, L1, L2), network weight initialization, activation functions (Sigmoid, ReLU, hyperbolic tangent, etc.), learning rate, decay speed (the rate at which the network learns), number of epochs, batch size, and optimization algorithms (SGD, AdaGrad,

RMSProp, ADAM, etc.). Better network performance is achieved by selecting better hyperparameter values/variables for the network. As a result, determining the optimal network hyperparameters is a considerable challenge. To discover the optimum hyperparameters, many approaches have been proposed in the literature, including Manual Search (MS), Grid Search (GS), RandomSearch (RS), and Bayesian Methods.

Convolutional Neural Networks (CNNs)

CNN is a variety of deep neural networks that consist of convolutional layers that are based on the convolutional operation. CNN is a type of DNN that is based on convolutional operation. Meanwhile, CNN is the most prevalent model that is widely used for classification issues that are based on vision or image processing techniques. Comparing the use of CNN to traditional deep learning models such as DMLP, the number of parameters is the primary benefit of using CNN. By implementing image processing using the kernel window function, CNN architectures with fewer parameters, which are advantageous for computing and storage, gain an edge in image processing. There are many layers in CNN designs, including convolutional, max-pooling, dropout, and a fully connected Multilayer Perceptron (MLP) layer that is completely linked. The convolutional layer is comprised of the convolution (filtering) operation and several related operations.

CNN model learning is accomplished via the use of the backpropagation technique. The most widely utilized optimization techniques (SGD and RMSProp) are employed to identify the optimal parameters of the CNN model. There are some differences between CNN and other DL models in terms of hyperparameters. The number of hidden layers, the number of units in each layer, network weight initialization, activation functions, learning rate, momentum values, the number of epochs, batch size (minibatch size), decay rate, optimization algorithms, dropout, kernel size, and filter size are all similar to other DL models. For the purpose of determining the optimal CNN hyperparameters, the following search methods are employed: MS, GS, RS, and Bayesian Methods.

Restricted Boltzmann Machines (RBMs)

In this paper, we describe RBM, which is a productive stochastic artificial neural network that can learn probability distributions on an input set. RBMs are mostly utilized for unsupervised

learning, which is the majority of their use. RBMs are utilized in a variety of applications, including dimension reduction, classification, feature learning, and collaborative filtering, to name a few examples. The benefit of RBMs is that they may be used to discover hidden patterns using an unsupervised technique. The drawback of RBMs is the time-consuming training procedure required. RBMs are difficult to understand because, although there are excellent estimators of the log-likelihood gradient, there are no known low-cost methods of estimating the log-likelihood in general.

RBM is a two-layer, bipartite, and undirected graphical model that is composed of two layers: visible and hidden layers. RBM is a graphical model that is composed of two layers: visible and hidden layers. The layers are not interconnected with one another at all. Essentially, each cell is a computational point that receives information and makes stochastic judgments about whether or not to transfer the signal to the next nerve node in line. Specifically weighted inputs are multiplied by certain threshold values, and then the resulting computed values are routed via an activation function, which is a mathematical function. After being output, the findings of the reconstruction step re-enter the network as an input, and then they depart from the visible layer as an output. After the operations are completed, the values from the prior input and those from the results are compared. The goal of the comparison is to narrow the gap between the two groups.

On the network, the learning process is repeated numerous times to ensure success. The training of RBMs is carried out by reducing the negative log-likelihood of the model and data in conjunction with each other. The Contrastive Divergence (CD) algorithm is used for the stochastic approximation method, which substitutes the model expectation for an estimate using Gibbs Sampling with a restricted number of repetitions using the Contrastive Divergence (CD) algorithm. The Kullback Leibler Divergence (KL-Divergence) method is used in the CD algorithm to determine the distance between the reconstructed probability distribution and the original probability distribution of the input.

The hyperparameters of RBMs include the following: momentum, learning rate, weight-cost, batch size, regularization method, number of epochs, number of layers, initialization of weights, size of visible units, size of hidden units, type of activation units, loss function, and optimization algorithms. The hyperparameters are searched for using techniques like MS, GS, RS, and Bayesian in the same way as the other deep networks. Additionally, Annealed Importance

Sampling (AIS) is utilized to estimate the partition function in addition to these methods. The optimization of RBMs is also accomplished via the use of the CD method.

Deep Belief Networks (DBNs)

DBNs are a form of deep artificial neural network that is composed of a stack of RBM networks. DBN is a probabilistic generative model composed of latent variables that may be used to predict the future. In DBN, there is no connection between the units in each tier of the network. Unsupervised learning is used to discover discriminating independent characteristics in the input set, and DBNs are utilized to do so. The capacity to encapsulate higher-order network structures, as well as the ability to do quick inference, are two of the benefits of DBNs. DBNs have the same training drawbacks as RBMs, which are discussed in more detail under the RBM section.

Stacked RBM learning and backpropagation learning are the two processes that make up the DBN training process. The iterative CD method is utilized in stacked RBM learning. Optimization techniques are used to train the network in backpropagation learning. Similar to RBMs, DBNs have hyperparameters that are quite comparable to RBMs. DBNs' hyperparameters include their momentum, learning rate, weight-cost distribution, regularization method, batch size, the number of epochs, the number of layers, initialization of weights, the number of RBM stacks, the size of visible units in RBM layers, the size of hidden units in RBM layers, the type of units, network weight initialization, and the optimization algorithms. The hyperparameters are searched for using MS, GS, RS, and Bayesian approaches, the same as they do with the other deep networks. When it comes to DBN optimization, the CD algorithm is also applied.

Auto Encoders (AEs)

AE networks are a sort of artificial neural network that is utilized as an unsupervised learning model. In addition, AE networks are often employed in DL models, where they remap the inputs (features) in order to make the inputs more representative for classification and therefore more accurate. That is, AE networks undertake an unsupervised feature learning process, which is very well suited to the DL topic. By using AEs to reduce the dimensionality of a data collection, it is possible to learn a representation of the data set. The design of AEs is similar to that of Feedforward Neural Networks (FFNNs). They are made up of three layers: an input layer, an output layer, and one or more hidden layers that serve to link the three levels. Asymmetrical

networks have a symmetrical structure, with the number of nodes in the input layer equal to the number of nodes in the output layer in the input layer and vice versa. The most significant benefits of AEs are the lowering of dimensionality and the learning of features. Reduced dimensionality and feature extraction in AEs, on the other hand, has a number of downsides. Because of the emphasis placed on reducing the loss of data linkages during the encoding of AE, certain important data connections have been lost. As a result, this might be considered a disadvantage of AEs.

In general, AEs are made up of two parts: an encoder and a decoder (or decoder and encoder). By using the encoder's weight matrix W1, bias vector b1, and element-wise sigmoid activation function, the input x between [0, 1]d may be transformed to the desired output x[0, 1]. The encoded component of AEs (code), latent variables, or latent representation is represented by the output h. Using the inverse of function f(x), known as function g(h), the reconstruction of output r is achieved (where W2 indicates the weight matrix of the decoder, b2 denotes the bias vector of the decoder, and 2 denotes the element-wise sigmoid activation function of the decoder). In the literature, AEs have been utilized for a variety of tasks, including feature extraction and dimensionality reduction.

AEs are a subset of FFNNs that have been further refined. The updating of the weights in the network is accomplished by the use of backpropagation learning. The learning process of AEs is aided by the use of optimization algorithms (SGD, RMSProp, and ADAM). In AEs, the MSE loss function is utilized as a loss function. In addition, recirculation algorithms may be employed to train the AEs over the course of the experiment. The hyperparameters of AEs are quite similar to the hyperparameters of DLs. A number of parameters, including the learning rate, weight-cost (decay rate), dropout fraction, batch size (minibatch size), number of epochs, layer count, number of nodes in each encoder layer, type of activation functions, number of nodes in each decoder layer, network weight initialization, optimization algorithms, and the number of nodes in the code layer (size of latent representation), is controlled by AEs. Similar to the other deep networks, the hyperparameters are searched for using the MS, GS, RS, and Bayesian approaches, as well as other deep network techniques.

Deep Reinforcement Learning (DRL)

In contrast to the supervised and unsupervised learning models, reinforcement learning is a sort of learning strategy that uses positive reinforcement to motivate students to learn. It is not

necessary to have a preliminary data collection that has been tagged or clustered before. RL is a machine learning technique that is inspired by learning action/behavior. It is concerned with determining which actions should be made by subjects in order to gain the largest reward possible in a given environment. It is utilized in a variety of application fields, including game theory, control theory, multi-agent systems, operations research, robotics, information theory, portfolio management, simulation-based optimization, Atari gameplay, and statistics. Some of the benefits of employing RL for control issues include the ease with which an agent can be re-trained to respond to changes in the environment and the fact that the system is continuously enhanced while training is being done on a continuous basis. It is via contact with its environment and observation of the effects of these interactions that a real-time agent learns. This learning approach is based on the fundamental manner in which individuals learn.

The Markov Decision Process (MDP) is the foundation of RL (MDP). MDP is used to codify the RL environment in a standardized manner. State transition probability matrix (p(s 0, r|s, a), where s 0 denotes the next state, r denotes the reward function, the reward function is denoted by the state, and the action is denoted by the action), discount factor (the present value of future rewards), and discount factor (the present value of future rewards). The agent's goal is to maximize the total amount of money he or she receives.

The range of RL solutions and methodologies available in the literature is just too extensive to cover in detail in this work. As a result, RL concerns are only briefly discussed. Model-based techniques and model-free methods are the two primary categories of RL approaches. Model-based methods are those that use a model to solve the problem. The model-based strategy makes use of a model that the agent has already encountered, as well as value/policy and experience. The experience might be actual (a sample taken from the environment) or simulated (a sample taken from the environment) (sample from the model). For the most part, model-based approaches are used in the application of robotics and control algorithms. Model-free approaches may be classified into two categories: value-based methods and policy-based methods. Value-based methods are those that are based on values. If you use a value-based approach, the policy is generated straight from the value function. If you use a policy-based approach, the policy is explicitly parameterized. There are three basic solutions for MDP issues in value-based methods: Dynamic Programming (DP), Monte Carlo (MC), and Temporal Difference (TD).

Recurrent Neural Network (RNN) Theory

A further sort of DL network is the RNN, which is utilized for time series or sequential data, like language or voice. Even though standard machine learning models (such as Back Propagation Through Time (BPTT) and Jordan-Elman networks, among others) use RNNs, the time duration of these models is often shorter than those of deep RNNs. It is preferable to use deep RNNs since they have the capability of including longer time periods. Instead of using external memory to process incoming inputs, Reinforcement Learning Neural Networks (RNNs) make use of internal memory. RNNs are utilized in a variety of disciplines, including handwriting recognition, voice recognition, and others, to analyze time-series data. The research states that RNNs are effective for predicting the next character in a text, language translation applications, and sequential data processing.

In each layer of the RNN model architecture, there are a variable number of layers with a different sort of unit in them. Because each RNN unit takes in both the current and past input data at the same time, there is a significant difference between RNN and FNN in terms of performance. The output of the RNN model is dependent on the preceding data. During the course of their operation, the RNNs process the input sequences one by one at any given moment. They store information about the history of the input in the state vector in the units on the hidden layer of the hidden layer. DMLP is created by dividing the output of the units in the hidden layer into distinct discrete time steps and converting the RNNs into DMLPs in the process.

Training RNNs may be accomplished via the use of the BPTT method. Weight adjustment algorithms are employed in the process of adjusting the weight. Because of this, while using the BPTT learning approach, the error change at any given time is reflected in the input and weights of the subsequent t times. This is owing to the fact that the RNN structure has a backward reliance over time, which makes it difficult to train RNN models. As a result, when it comes to the learning stage, RNNs become very complicated. However, despite the fact that the primary goal of utilizing RNN is to learn long-term dependencies, research in the literature has shown that when information is held for extended periods of time, it is difficult to learn using RNN. The development of LSTMs with various ANN architectures was undertaken in order to address this specific difficulty.

The recurrent neural network is categorized in the class of artificial neural network. It uses the technology from the feedforward and it can use internal state memory in processing the

variable-length sequences of inputs. Sherstinsky (2020) says that in RNN, the connection between the two nodes forms a direct graph along the temporal line; and it's this characteristic that allows it to create a unique material behavior. Research made by Yin et al (2017) shows that the RNN can run arbitrary programs to process random input considering they are Turing complete. RNN is significant in society today, and it's used in various areas such as speech recognition, connected handwriting recognition, and unsegmented. Li et al (2018) research identifies the RNN in two main classes: finite impulse and infinite impulse, and they all have the same function in that they release temporal dynamic behaviors. In his studies, these two classes FIR and IIR, have additional stored states, and a neural network directly controls the storage. Any network or graph can replace the warehouse when there is a delay in time or feedback loops. The entire controlled state is known as gated memory or gated state. The RNN network was discovered in the late 1980s and became famous in the early 1990s when it solved a deep learning task.

According to Weiss et al (2018) research notes that finite impulse RNN is a directed acyclic graph and it can be unrolled or replaced with a feedforward neural network. The finite impulse settles at zero finite time in response to the finite-length input of any limited time. The finite inspiration can be digital or analog and simultaneously be with a continuous-time or discrete. Weiss et al (2018) research in his study of Infinite in precision, he states the properties which the finite impulse has that have made it useful in the current generation. The FIR doesn't require feedback, and a summed iteration does not compound any rounding error. Besides, the same relative error is in the calculation, making its implementation easier and simpler. It is inherently stable, and it can efficiently be designed into a linear phase by making the co-efficient sequence symmetric. FIR is designed by matching the filter orders and co-efficient, which meets given specifications in the time and frequency domains. Several designs are used when a specific frequency response is needed, such as window design, frequency sampling methods, mean square error, and optimal way. On the other hand, infinite impulse response doesn't go to zero when it passes a given point; instead, it continues indefinitely. Merrill et al (2020) assert that it is a directed cyclic graph that can't be unrolled. The approach is used in several linear time-invariant systems, such as in digital filters and electronics. The analog electronic filters use IIR technology. The transfer function IIR lets a person know whether the system is bounded input or bounded output stable. Its stability requires the ROC of the system, such as the unit circle.

Further, Shen et al (2018) research states that RNN is an artificial neural network that uses time-series data or sequential data. Deep learning algorithms are mainly used in solving temporal or ordinal problems. They are placed in typical applications such as voice search, google translate, and Siri. The RNN utilizes the training data to learn just like another artificial neural network does. However, they are different from others based on their memory, and they take information from the previous input to influence the present input and output. These works are different from the traditional deep neural networks input, which believes that the information and the result are independent (Karita et al, 2019). In this case, the RNN output relies on the previous elements in the sequence. Language and speech recognition have used the technique to get the desired results, which explains why the method is actively used in forecasting.

In the application, the RNN accounts for each word in the idiom, and it uses the same information to predict the next possible word in the sequence. Shin et al (2017) note that the rolled RNN represents the entire neural network, the whole indicated phrase. In contrast, the unrolled in the visual represents a single time step or layer of the neural network, and each layer matches a single word, and the previous inputs are hidden. Moreover, RNN shares parameters across each layer of the network. Miao et al (2015) state that the RNN has the same weight parameter within each network layer, unlike the feedforward network, which has different weights across each node.

However, The RNN weight is adjusted through a gradient descent process and backpropagation to enhance reinforcement learning. The RNN uses the principle of the backpropagation that's BPTT to find out the gradients, and their approach is different from the traditional way, which is specific to sequence data (Manaswi, 2018). The system has trained itself in calculating errors from its output to input layers, and the calculations give room for adjustment and fit the parameter in the model effectively. The BPTT sums errors in each step differently from feedforward, which doesn't add mistakes because they don't share parameters across each layer. The RNNs are of different types because different RNNs are used for other purposes. The RNNs are expressed as One to one, one to many, many to one, many to many, and many to many.

The RNN deals with two main problems that are exploding gradients and vanishing gradients. They are the size of the angle along the error curve, which is the slope of the loss function. When the incline is slight, they continue reducing to smaller as it updates the parameters weight until it becomes insignificant zero; when that happens, the algorithm stops learning (Liu et al, 2020). The exploding gradients happen when the angle is considerable and creates an unstable

model. That means the weight will be too large and represented with NaN. To solve this is to minimize the number of hidden layers in the neural network and remove some of the complexity in RNN.

Recurrent Neural Network (RNN) Forecasting Research

The financial market doesn't allow simple models to predict the future; therefore, using high accuracy and quality model brings reliable results. RNN has proved to be helpful in data sequential analysis. In forecasting, no data remain permanent, and it tends to change with time. When the RNN is familiar with the change in data, then prediction becomes more accessible. The neural networks have received a higher advantage in the forecasting of financial data series (Qin et al, 2017). There are several classical methods used, such as ARIMA and Box-Jenkins; however, RNNs have the advantage over them because they can approximate the non-linear functions.

Forecasting works well where the prediction technique is identified. Gallicchio et al (2018) explore the effects of epochs and several neurons available in the time series prediction. The study's main aim was to find out the practical approach to predicting the best results in the financial market. The findings showed that the fewer epochs don't provide RNN learning. The number of neurons plays a significant role in setting the recurrent neural network learning process. A large number of neurons generates stunning prediction results. However, for this to be successful, it needs more training and time.

The neural network has helped apply signal processing, and the same method is employed in predicting daily foreign exchange rates. In predicting the noisy time-series data, the RNNs are preferred because they have feedback connections and the ability to represent a given computational structure effectively. RNN focuses on the temporal relationship of the input and thus maintaining the internal state. Besides, they are less subjected to random learning correlation, which doesn't happen in correlation order. The use of RNN is effective based on its temporal relationship with series and its modeled through internal states (Tokgöz et al,2018). Besides, it can get rules from the trained recurrent networks in a deterministic finite-state.

The foreign exchange market is one of the largest markets known in the world today since its introduction in 1997. The most recognized and valued currency is US dollars. These foreign exchange rate releases high noise, and are non-stationarity. Most financial investors use quality prediction algorithms in forecasting (Canizo et al, 2019). In prediction, they use the current

percentage which the algorithm predicts from the present day to the future. In this case, RNN is the best choice because it has connected layers that are a reservoir and this makes RNN have a short memory and it can capture information about what is calculated. This characteristic is essential in dealing with complex issues like financial forecasting. The RNN in the forecasting has successfully managed to minimize the high percentage of test errors which is done through linear regression.

Long Short-Term Memory (LSTM) Theory

LSTM is a form of RNN in which the network can recall both short and long-term data. When it comes to complicated tasks such as automated voice recognition and handwritten character recognition, LSTM networks are the favored option of many DL model developers. Time-series data is the most common kind of data for which LSTM models are utilized. A variety of applications, such as Natural Language Processing (NLP), language modeling, language translation, voice recognition (including sentiment analysis), predictive analytics (including financial time series analysis), and others make use of this technology. The use of attention modules and AE structures may improve the performance of LSTM networks when used for time series data processing, such as language translation. LSTM networks are made up of LSTM units that communicate with one another. Each LSTM unit combines with the others to produce an LSTM layer. An LSTM unit is made up of cells that have three gates: an input gate, an output gate, and a forget gate. The information flow is controlled by three gates. Each cell retains the required values throughout an arbitrary number of time periods as a result of these qualities.

The LSTM algorithm is a customized version of the RNN. So the weight updates and recommended optimization approaches are the same in this case. Furthermore, the hyperparameters of LSTM are the same as those of RNN: the number of hidden layers, the number of units in each layer, network weight initialization, activation functions, learning rate, momentum values, the number of epochs, batch size, decay rate, optimization algorithms, sequence length for LSTM, gradient clipping, gradient normalization, and dropout are all available. The hyperparameter optimization strategies that are utilized for RNN are equally relevant to LSTM in order to discover the optimum hyperparameters for the LSTM.

LSTM is used in deep learning, and it's an artificial RNN design. It’s different from feedforward because it has connections and it can process the whole sequence of data. The research

by Xue et al, (2018) states that the LSTM units namely the cell, input gate, output gate, and forget gate are critical to speech and handwriting recognition. The cell will remember the values over arbitrary time intervals, whereas the three gates control the flow of the information in and of the cell. The LSTM network is effective in processing, classifying, and making predictions based on time series data. Considering there can be lags of unknown duration between important events in time series. Altché & de La Fortelle (2017) in their studies, assert that the LSTM is developed to handle the vanishing gradient problem. This is possible when traditional RNNs relative insensitivity to gap length gives them an advantage over RNNs. Juergen Schmidhuber and Hochreiter founded LSTM through their passion for artificial intelligence. The LSTM has primarily contributed to deep learning, and it's currently used by tech heavyweights such as Facebook, Google, and other significant sites for speech translation. The idea of LSTM was invented in the 1990s.

Moreover, Tian et al (2018) note that the RNN can't predict the word stored in the long-term memory, but it gives more accurate predictions from the previous information. When they're an increase in their RNN, it doesn't provide any efficient performance. The LSTM is good at retaining information for an extended period. The machine is effective in processing, predicting, and classifying based on the time-series data. In structure, the LSTM contains four neural networks with different memory block cells. The information is retained in the cells, and the memory manipulation is done with gates. In application, the LSTM handles complex problems in different domains. It is complex in deep learning, and it’s known to overcome technical issues and deliver quality on the recurrent neural networks. Long-term memory is helpful to have and can solve several tasks that recurrent neural networks can't solve. It deals with any sequential processing tasks where the hierarchal decomposition is present, though it does not know the decay. In speech recognition, the concept supports the recognition and translation of the spoken language into texts using computers. The approach is known as automatic speech recognition, speech to text, or computer speech recognition (Zhao et al, 2017). This method needs a combined knowledge of computer, computer engineering, and linguistics to be successful. Some speech recognition requires training for a person to have clear information on how to handle it. The training is called enrollment, where a person speaker reads the information or the isolated texts into the computer system. The system will analyze the individual voice and recognize the specific voice,

Furthermore, the three gates in LSTM are useful, in DiPietro & Hager's (2020) research, in the forget gate, the information is not helpful in the cell state, and the forget gate removes it. The two inputs are input at a particular time, and the previous cell output is fed to the gate and then multiplied by the weight matrices, followed by addition bias. The resultant passes through the activation function to give binary output. In the input gate, adding helpful information to the cell state is done by the input gate. The data is managed with sigmoid, and values filtered are remembered with the same case like forget gate. The output gate is tasked with extracting the relevant information from the current cell state is presented as at the output in the output gate. The vector is released by applying the tanh function to the cell. The information is controlled by using the sigmoid function and filtered by they are remembered using input. The LSTM is good in capturing long-term temporal dependencies without suffering from optimization issues, they have been used to solve many difficult tasks (Li et al, 2019). Some of the complex tasks entail recognizing and generating handwriting, language modeling and translation, acoustic modeling of speech, and speech synthesis. Besides, it is also used to predict protein secondary structure and analyze the audio and videos. These are among the few tasks LSTM does. The approach has gained a lot of popularity in many industries due to its role.

In training, the RNN that uses LSTM and is trained in a supervised method on the set of training sequences. It uses algorithms such as gradient descent, and it is combined with backpropagation through time in order to compute the gradient required in the optimization process (Sherstinsky, 2020). This is necessary for changing the weight of the LSTM network is in line with the derivative of the error to the corresponding weight. There are many reported success stories of the training in the non-supervised fashion of RNNs with LSTM units. Most of this training happens without teachers, and that is known as training labels. However, there is an added advantage of training LSTM with neuroevolution. Most of the applications use LSTM RNNs and train using a connectionist temporal classification approach to develop an RNN weight matrix that can maximize the label sequence probability in the training set. The connectionist temporal classification approach is effective in recognition and alignment.

Since the introduction of the LSTM in 1997, it has gone through many versions and what is currently used is more advanced with many valuable features. One of the success stories is with Bill Gates, who took part to improve artificial intelligence and the development of Open AL transformed the world. It uses five independent neural networks; however, they are coordinated.

A policy gradient trains each network without any supervising teacher, and it has a single layer (Tian et al, 2018). The LSTM monitors the game and releases actions through action heads. The Open AL in 2018 trained another LSTM using a gradient policy approach to control a human-like robot hand that manages the physical objects with un-paralleled capability. Another program, deep mind, initiated by Alpha-Star, used the LSTM approach to succeed in the complicated video game Starcraft II. This move has been viewed as a significant transformation in artificial intelligence. There among many success stories used by this LSTM technology. Even in the medical industry, they have applied the concept, and it has worked effectively in solving various issues.

Long Short-Term Memory (LSTM) Forecasting Research

LSTM is suitable for forecasting, and several models are effectively used on every specific time series forecasting problem. This approach was introduced in the 1990s and today is one of the powerful techniques used in forecasting. LSTM offers many features such as generalization on memory-based, which gives an advantage over ARIMA and HWES, the commonly known methods for forecasting. According to the research made by Bouktif et al (2018) in their study on forecasting, he notes that LSTM is commonly known for natural language processing, and that makes it more useful in time series forecasting. LSTM comes with a solution that RNNs suffer from, and that's a short memory. Considering the LSTM has three gates and each with its RNNs, it is easier to keep, forget or ignore some data points using the probabilistic approach. Besides, LSTM provides a solution to vanishing and the exploding issues of the gradient. This issue arises due to continuous weight adjustment as the neural network trains (Chen et al, 2018). It results in smaller or larger gradients; however, this issue LSTM manages it. This is a powerful technique highly reliable in the current generation.

Further, with LSTM, the prediction is made, and it is fed into the model so that it can predict the next value in the sequence. Several errors are introduced into the model whenever the prediction is made, and they are squashed through sigmoid and tanh activation to prevent the exploding gradient. The sigmoid function before the gate entry and output.

Financial forecasting is important. From the research made by Yang et al (2020) predicting data is easy; it involves taking the prepared input data X and using one of Kera's prediction methods on the loaded model. The input for making the prediction of data X is the only one affected in the sequence data needed to make the prediction and not the entire training data. The

study by Chen et al (2015) emphasizes the LSTM use in making a prediction in the financial stock market. The prediction of values using this approach has proved to work effectively in the market. Time series forecasting is part of us in the current world, and predicting the value based on the previously observed values is the only solution to the problem at hand. Regression analysis with the help of LSTM gives a reliable answer after testing the relationship between two or more time series. The time series have natural temporal ordering, and therefore, LSTM is the most preferred in classifying, processing, and making predictions because it has a delay of unknown time between the relevant events in the series.

In predicting stock prices, LSTM is powerful and reliable in sequence prediction because they are able to store past information. This is a crucial matter because the previous price is very critical in predicting the future price. Predicting the stock price is a process whereby it starts by loading the dataset, scaling, creating data with timesteps, and then building the LSTM (Zhao et al, 2017). LSTM is used in forecasting because it has large accuracy, and this has increased its demand by the forecasters. The best decision-making in forecasters in business depends on the best tool selected for prediction, and the majority goes for LSTM due to its advantage in the market. The LSTM has the ability to capture data of different seasons since it demands different patterns of data it can yearly or at intervals of months.

Building LSTM for forecasting requires the use of the Keras approach; it observes a number of steps first, sequential for initializing the neural network; second, a dense to load the densely connected neural network layer, an LSTM to add a layer of Long short-term memory; and finally, a dropout to add dropout layers which will prevent overfitting. Predicting the future stock when the test set is loaded has a number of procedures it follows, which are very critical (Altché & de La Fortelle, 2017). Such as merging the training set with the test set at the 0 axes, setting the time step to 60, and use the minmaxscaler in transforming the new dataset, and finally reshaping the dataset. When these steps are followed keenly, it leads to a successful prediction of the market stock price. The result is plotted to compare the real stock price and the predicted stock price, and it shows some similarities.

Empirical Financial Time Series Forecasting

The forecasting of a specific financial time series, and in particular the forecasting of asset prices, is the financial application field that has received the greatest attention. Despite the fact

that there are various variances, the major emphasis is on anticipating the future movement of the underlying asset. More than half of the current DL solutions were focused on this particular area of application. Despite the fact that there are several subtopics of this general problem, such as stock price forecasting, index prediction, forex price prediction, commodity (oil, gold, etc.) price prediction, bond price forecasting, volatility forecasting, and cryptocurrency price forecasting, the underlying dynamics are the same in all of these applications.

The research may also be divided into two primary divisions depending on the projected outcomes they are intended to produce: price prediction and price movement prediction (trend prediction). Despite the fact that price forecasting is fundamentally a regression issue, in most financial time series forecasting applications, the accuracy of the price prediction is not considered as significant as the accuracy of the directional movement identification. Because of this, experts regard trend prediction or anticipating which direction the price will move, to be a more important study topic than accurate price prediction, as opposed to the former. The challenge of trend prediction is transformed into a classification problem in this way. Only up or down motions are taken into account in certain research (2-class issue), however, there is also a 3-class problem that takes into consideration up, down, and neutral movements.

The LSTM and its variants, as well as several hybrid models, are the most often used models in the financial time series forecasting arena. Because LSTM, by its very nature, makes use of the temporal properties of any time-series signal, financial time series forecasting is a well-studied and effective use of LSTM in the financial domain. Although some researchers prefer to extract appropriate features from the time series, others prefer to transpose the data in such a way that the resulting financial data becomes stationary from a temporal perspective, i.e. we can still properly train the model and achieve successful out-of-sample test performance even if the data order is shuffled. The CNN and Deep Feedforward Neural Network (DFNN) models were the most often used deep learning models in those implementations.

Summary

This literature review was created to survey the financial time series field. In particular, topics focused on forecasting asset prices have their original basis in statistical analysis. The most popular forecasting method is an autoregressive integrated moving average (ARIMA). The

benefits of ARIMA were capturing prior points that predicted future points, seasonal trends, and an overall trend captured via moving average.

Volatility in finance was another issue and could not be solved with ARIMA alone. Hence, generalized autoregressive conditional heteroskedasticity (GARCH) was developed to improve the forecast in highly volatile events.

Machine learning methods became a dominant forecasting tool with the rise of more significant amounts of data with corresponding computing power. The first widespread successful machine learning model included support vector machines which learned from non-linear patterns. Afterward, ensemble methods, combining weak machine learning models to form a stronger one, such as XGBoost, arose with strong empirical forecasting performance. With even more technological efficiencies of computing power, deep learning applying neural networks has surpassed all prior machine learning models in terms of performance and ability to handle more significant amounts of data. As of the time of this writing, one of these variant neural networks is Long Short-Term Memory (LSTM), which can retain essential points in a time series to forecast strong results. In financial time series forecasting, LSTM models are the most used. Financial time series forecasting is a well-studied and successful usage of LSTM in the economic sphere because it uses the temporal features of time-series signals. Others choose to transpose the data such that the resultant financial data is stationary in time. CNN and DFNN were the most often utilized deep learning models.

Chapter 3: Methodology

This chapter describes the research design and methodology performed in the inquiry of time series forecasting for a semiconductor index with a variant of deep neural networks (LSTMs). The data analyzed was the SOXX index closing prices from 2009 to 2020. The data was gathered from Yahoo Finance and was public data, so site permissions were not required. The deep neural network was coded in Python, specifically within a Jupyter notebook. The study was done with reproducibility and transparency, so that the respective code will be attached in the appendix. In particular, the topics included will elaborate on research design, research questions, hypotheses, population with the sample, role of the researcher, location of research, instrumentation applied, data collection procedure, data analysis, coding, hypothesis testing, and trustworthiness of the study.

Research Design

Due to the nature of time series forecasting, this subject will be primarily quantitative. In this study's scenario, semiconductor practitioners, researchers, and financial experts may employ the insights from this time series forecasting model for hedging or investing choices. The research provided here will also contribute as a case study to the continuing debate between the Efficient Market Hypothesis' grounded theory and the semiconductor forecasting literature.

Since this study will only examine price history data to forecast future prices, an introductory statement explaining why the forecast occurs is out of scope for this research.

The price history will be split between an in-sample and out-of-sample data. The machine learning model will only see the in-sample (also known as training data) data, while the future forecast will be evaluated via out-of-sample data (also known as test data). A qualitative study would have been more suitable for a question of why or how the forecast works or for explaining underlying price movements.

For the prior reasons, a quantitative study is most appropriate for this study and applies a comparative analysis of historical asset prices based on a buy and hold performance over a machine learning method.

The LSTM time series forecast will be constructed by collecting data on daily ETF prices and analyzing forecasting quality using accuracy over the next day forecast and profitability in beating the buy and hold performance. In other words, the anticipated production will be examined using traditional machine learning time-series metrics and compared to the buy and hold strategy regarding return on investment. More detail on the metrics are found in Appendix E, but typical time-series metrics include R2, Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Weighted Mean Absolute Percentage Error (WMAPE). The data will be obtained through API from Yahoo Finance and saved in a comma-separated file (CSV). The study will be carried out using the open-source programming language Python.

Python will be used to develop LSTM deep learning models to produce anticipated ETF prices. Open-source frameworks such as TensorFlow 2 and Pandas data frames will be used.

The benefits of this study allow other researchers and practitioners to examine the effectiveness of LSTMs on financial asset forecasting. In addition, the semiconductor index analyzed has higher volatility than other index sectors such as consumer discretionary or utilities. As a result, other parties may use these findings to refine future machine learning models or even use the forecast results of these models as another data point to be fed into their larger forecast model for superior results.

Research Questions

Out of the entire scope of this study, there are three major research questions. The first question asks how well the forecasting technique performs. The second question asks how well the forecasts can perform about other strategies concerning investing decisions. The third question focuses on how much data is required if the forecasts prove fruitful.

Research Question 1: Does LSTM with SOXX prices offer accurate forecasting?

Research Question 2: Can LSTM SOXX daily price forecasts deliver higher returns than a buy and hold of SOXX?

Research Question 3: How does sample size or training period improve LSTM forecasting accuracy; can a 20-year example be practical as a 10-year sample?

Hypotheses

The hypotheses aim to further drill down from the research questions to distinguish between a successful or unsuccessful experiment clearly. In this case, the study tests how well an accurate forecast is, what practical investment benefit LSTM models have against more traditional methods, and how much daily pricing data is required to maintain a precise prediction. The first hypothesis aims to evaluate the accuracy of the overall forecast.

The benchmark chosen was forecasting accuracy greater than 50% since anything less would be worse than random chance or a coin flip in predicting an up or down movement. On a practical level, forecasting correctly over 50% can give a trader a good advantage, like how a casino has a slight over 50% advantage in roulette that leads to profit over a large sample. In a more concrete example, financial products offer one-to-one risk-reward payoffs, so a greater than 50% forecasting accuracy would yield profit in these financial products. The second hypothesis targets the practical benefit of choosing a more complex model (LSTM) against a more straightforward strategy. Since more data leads to higher business costs, the final hypothesis examines how much data is necessary to run the model.

H01: LSTM models do not produce accurate (<=50%) forecasts.

HA1: LSTM models do produce accurate (>50%) forecasts.

H02: LSTM models do not generate higher absolute returns compared to buy and hold.

HA2: LSTM models do generate higher absolute returns compared to buy and hold.

H03: Sample size does not affect LSTM forecasting accuracy.

HA3: Sample size does affect LSTM forecasting accuracy.

Population and Sample

The only data gathered will be SOXX historical price from Yahoo Finance from 2009 to 2020, and the frequency of the data collected will be daily. The sample is representative of the population of a semiconductor index. Since the data covers over a decade, many different cycles are captured, including recessions and business cycles. Other researchers may examine the exact dates from the sample taken to replicate the results performed in this study. In addition, more than 3000 data points are considered due to the time frame of daily closing data. Also, the data is enough to compare LSTM forecast performance against a less complex buy-and-hold strategy.

The characteristic that makes semiconductors interesting is the heightened volatility that makes forecasting this sector difficult.

Role of the Researcher

The role of the researcher in this study is to act as the primary contributor to synthesize all the findings from other literature reviews, data collection, data analysis, and formulate a discovery of whether LSTMs can forecast SOXX prices with a benefit of superior investment performance. Only one researcher is needed to complete this study since the scope and tasks demanded are sufficient for a single person.

Geographical or Online Location

The location of the population and sample is an online location since SOXX prices are reported as public information. The participants in a free market determine the price exchanged at any particular moment, which means that prices represent market forces. The components that make up SOXX are 30 semiconductor companies located in various parts of the world.

Procedure

This study does not have human participants and will only gather historical public data. As stated in previous chapters, this study will only analyze price data of the SOXX index. The data will be collected via Yahoo Finance API. The API used is yfinance, which applies Python and Yahoo Finance stock data to be easily gathered into Python coding. In particular, the API will be used with Python to collect the historical price data. The historical price data includes the price date on a daily time frame along with the open, high, low, close, and volume. The dates gathered will range from 01-01-2009 to 12-31-2019.

Once the data is gathered, the prices will be transformed with a scaler to make the data easier for the LSTM model to process. The scaler applied will be from the Sklearn Python package commonly used with machine learning. In particular, the scaler used will be MinMaxScaler, which scales all the values between 0 to 1. Other standard scalars normalize the data, but I wanted to keep potential outlier data points to see if the LSTM model can notice these complexities for increased forecasting.

Data gathering, model building, and price forecasting will be done with Python. Along with Python, open-sourced libraries will be used, making performing the price forecasts and reproducibility more streamlined. Besides the libraries already discussed, other packages include Numpy, Pandas, Matplotlib, Datetime, and seed. Numpy is for handling matrix calculations, which apply to forecasting and data preparation. Pandas is to take structured data like Excel or CSV file data types. Matplotlib enables plotting to be shown, such as pie or line charts. Datetime aids in the data manipulation for the stock time series data. The Seed package fixes the randomization functions for others to reproduce the research stated here.

The LSTM model applied will be done with the Tensorflow and Keras packages. Tensorflow is a package with the LSTM code that can be customized with a higher level code via Keras. Without Keras, I would have to be more hands-on in the code to operate Tensorflow and use LSTM financial forecasting properly. The final package will apply the Sklearn metrics package to enable the use of desired forecasting metrics like RMSE.

Instrumentation

The instruments used in this study will be Python, Jupyter Notebooks, Yahoo Finance API, and comma-separated files (CSVs). Python will be the coding language used to gather the data via an application programming interface (API), analyze the data, and perform forecasts. Jupyter Notebooks will be the coding environment where Python is used for this study. Yahoo Finance API will be the location where the semiconductor data is gathered. The CSVs will be the database where the pricing data will be stored. Python with Jupyter Notebooks is an industry and research option used by many leading companies and researchers. Yahoo Finance is a common location to gather asset pricing data for free and acceptable research. CSVs are a known storage type for small structured data such as asset prices of this magnitude.

Pilot Testing

For the pilot testing phase, the study will use S&P 500 (SPY) ETF daily historical data taken from Yahoo Finance for the years 01-01-2009 to 12-31-2019. The point of this data is to be very close in terms of the number of data points and data type of the actual research. Only the closing price will be used for forecasting and handled via Yfinance, Pandas, and Numpy. The data will be preprocessed with the MinMaxScaler and then sent to the LSTM model created with

Tensorflow and Keras. Once the forecast is made, the performance will be evaluated with Sklearn metrics to see if the metrics calculations perform as expected, along with the estimates plotted with Matplotlib. The main goal of this test is to have all of the code working without errors. The Python code will explicitly state the errors, which would hinder the SOXX experiment. Assuming no errors occur after all of the debugging, the code will be ready to take the SOXX study.

Data Collection

As stated in the instrumentation section, all the data collected will come from the Yahoo Finance website and be the SOXX index daily pricing data for 2009-2020. The time to gather the data will take at most a few seconds. Python code will collect the data from Yahoo Finance via API. This code will be executed within a Jupyter Notebook for ease of use for data analysis later on. The pricing data will be stored as a CSV, and the CSV will act as a proxy for a database. This data collection method allows for quick data gathering and ease of analysis since the data analysis will also be performed with Python and Jupyter Notebooks. This way, the LSTM and other performance calculations can be done in one location without the need for other more complex processes. The data collected will cover over ten years of SOXX data (e.g., 2009-2020). This data will enable the capturing of an entire business cycle, eight years, and the other shorter boom and bust events in a stock market. Data from 2009 was also taken to capture some recessionary effects and the other years to capture the bull runs in US equities.

Data Analysis

The predominant form of data analysis will occur after the LSTMs makes the forecast for comparative performance against a buy and hold strategy. This is due to the fact that the pricing data of the index has high data quality from Yahoo Finance. In other words, there will be an expectation of very little transformations required other than checking if all the proper trading days were recorded. If missing data is found, then an average between the dates will be taken to fill the missing data point. Once the data is gathered, the LSTM model will train from the majority of the data and a smaller subsample will be used as the performance against buy and hold. From this performance comparison, metrics such as return on investment, precision, recall, and accuracy will be calculated.

Descriptive statistics summarize the central tendency, dispersion, and shape of a dataset's distribution, excluding NaN (missing) values, which will be applied to the SOXX data. Python has a function that allows describing the data in terms of count, mean, standard deviation, minimum value, maximum value, and the interquartile range. These values will aid in describing the nature and composition of the semiconductor sample taken from the years 2009 to 2020.

The coding for this study is built upon the data collection due to the use of the Yahoo Finance API. The order of coding will first be the data gathering, which will also store the data via CSV. After data collection, descriptive statistics will be calculated on the semiconductor index. Once the statistics and data quality checks are complete, the data will be preprocessed via standardization for the LSTM model. The analysis will occur in an open-source programming language called Python. With Python, open-sourced libraries such as TensorFlow 2 and Pandas data frames will be applied to create LSTM deep learning models that will in turn output forecasted ETF prices. The forecasted output will be analyzed by conventional machine learning time series metrics and return on investment compared to a buy and hold strategy.

Triangulation

This study only applied public datasets along with public open source models. The triangulation did not perform any user interviews. Hence the triangulation used a combination of data, theory, and researcher reflection with conventional metrics to create reproducible research that help eliminate and verify against bias.

Informed Consent Process and Ethical Concerns

This research relied exclusively on publicly accessible data and did not include any human subjects. As a result, no permission was necessary.

Trustworthiness of the Study

This study has a high level of trustworthiness due to the fact the credibility of the findings and methods can be confirmed by other researchers applying the same process. The model performance may be transferred to other similar asset prices with similar performance results. The dependability of the results can be replicated due to fixing the randomization of the stochastic models with a seed code that will be provided. The seed is a way to fix the random number

generator that is commonly used in deep learning model to aid the model in learning the data set given. In the same light, the code will be available for other researchers to examine, which will enable full transparency in the methods and findings presented.

Limitations

The stock market contains a large amounts of exchange traded funds, indexes, and even more stocks from those funds. Semiconductor index analysis data cannot be applied directly to other types of tradable assets. The volatility profile of other tradable assets differ from asset to asset and must be evaluated by the researcher or practitioner to adjust their own applications.

Delimitations

This research only applied LSTMs on a semiconductor index. The aim was to examine the performance of the particular deep learning model on a highly volatile asset to see its forecasting performance. Other models were not a focus of the study due to not having the wide impact in time series compared to LSTMs based on latest research at the time of writing this research. In addition, the semiconductor index has been sector that has not been given much research compared to more popular assets such as the S&P 500 or Dow Jones Industrial Average.

Summary

This chapter outlines the study concept and methods used to anticipate a semiconductor index using deep neural networks (LSTMs). The subjects covered research design, research questions, hypotheses, population and sample, researcher function, research location, equipment used, data collecting technique, data analysis, coding, hypothesis testing, and study reliability. The LSTM time series prediction is built using daily ETF price data and industry-standard forecasting quality metrics. Data will be received from Yahoo Finance through API and stored in a comma-separated file (CSV). The research will use Python, an open-source programming language. This study's findings enable other academics and practitioners to evaluate LSTMs for financial asset forecasting. The semiconductor index is also more volatile than other index categories like consumer discretionary and utilities. As a consequence, third parties may utilize these insights to improve future machine learning models or even feed the prediction results into a bigger model for better outcomes.

Chapter 4: Findings

This section examines the LSTM model forecasting performance findings against the SOXX historical price data. The LSTM model was first applied to the S&P 500 ETF (SPY) before being applied to SOXX. The model captured the general trend of each asset. The code used was Python which is attached in the appendix. The LSTM model predicts SOXX daily price movements with approximately 69% accuracy.

General Description of Participants

The data studied did not include any human participants. The data gathered was the price history of the iShares Semiconductor index (see Table A1). The dates of the price history data included the date on a daily time frame with open, high, low, close, and volume is taken from Yahoo Finance.

The SOXX is a time series dataset with a total of 4645 days. For this analysis, only date and close data were analyzed. The dates of data points were from 2001-07-13 to 2019-12-31. These dates were chosen to capture at least one business cycle and capture many. In this case, approximately 3.6 business cycles are due to the definition of a business cycle being an average of 4.7 years (Zarnowitz, 1992). Throughout this time series, the US market, according to the S&P 500, has had a total of 18 recessions and corrections (Yardeni Research, 2022).

Due to the publicly available data, the index time series was clean data and did not require any transformations for null or missing values. Hence all of the 4645 data points were used for this study.

Unit of Analysis and Measurement

The unit of analysis is the LSTM model on asset prices. Of course, the model can be applied to other time series, but this study focuses more on the nuances of the volatile semiconductor ETF. The unit of measurement of this research project is the forecasting accuracy of LSTM on time series asset prices. The accuracy is chosen as the significant unit due to the widespread application of the forecast quality. A higher forecast can lead to better decisions for

researchers or more profits for market participants. Other supporting measurements help evaluate the quality of the model's prediction.

Sample Size

The sample was taken from the population of the semiconductor index (SOXX). The sample amount totaled 4645 trading days from the years 2000 to 2019. The sample was large enough since the period captured two business cycles with many different phases of the market. Saturation of the data can also be achieved with future researchers or practitioners by taking a rolling 20-year data sample to their most recent time event.

Pilot Testing

The study pilot-tested the SPY ETF, the same performance as the S&P 500 index. The data points taken were the same period, frequency, and amount as the SOXX ETF. The LSTM model ran the untuned version to test that the code was functional. Also, hyperparameter turning is a very time-consuming event to get the results for one model run. The data did not have any missing values and was a good test run for the SOXX data due to being very similar in the data structure.

Data Collection

The data source used was public stock price data collected from Yahoo Finance. Python has an open-source package that connects to Yahoo Finance via an application programming interface (API). With an API, stock prices can be gathered programmatically, which results in a more streamlined analysis in preparation for the LSTM model.

Codebook Creation

The codebook creation was coded in Anaconda, a coding software composed of various scientific programming tools. Jupyter notebooks was one of those tools, and the code written inside the notebook was Python. The code contained the Yahoo Finance API, the data preparation, and the LSTM model with predictions. The supported Python code from the Jupyter Notebook is attached in the appendix.

Qualitative Results

This section explains the primary quantitative results of this LSTM SOXX study. The daily accuracy of the forecasting model on SOXX was 69%. Removing the transaction costs, this daily forecast model would have produced a 1,420.70% return instead of 38.93% from buy and hold of the SOXX ETF. Most practitioners use the S&P 500 buy and hold returns as a benchmark, 21.48%, based on the SPY ETF in the same time frame. The out of sample RMSE was 6. This value is in line with other results examined during the literature review. The outliers of the dataset were kept as the goal was for the model to understand the general market for an extended period. For example, recessions might seem like outlier events but not when taken as a whole in a more extensive time frame.

Results of Hypothesis Tests

The hypothesis tests were not of the conventional sense of applying p-values but more of proposal and conjecture. All results are repeatable via the Python code attached in the appendix.

First Hypothesis Test

HA1: LSTM models do produce accurate (>50%) forecasts.

The LSTM models produced forecasts greater than 50%, which would indicate better forecasting accuracy than randomly guessing.

Second Hypothesis Test

HA2: LSTM models do generate higher absolute returns in comparison to buy and hold.

The daily returns applying the forecasted returns without including trading costs resulted in returns greater than a factor of 10. The returns each day were compounded returns.

Third Hypothesis Test

H03: Sample size does not affect LSTM forecasting accuracy (<= +/-10%).

The study used almost 20 years of data. To test the sample sizing while catching at least one business cycle, the sample data size was truncated to 10 years to test the performance. From the 10 year sample size, the LSTM model had similar daily accuracy percentage of 69.48% and, from a 5 year sample size, the LSTM model had similar daily accuracy percentage of 69.11%.

Outliers

All data points were kept, since the goal of this study was for LSTM to generalize SOXX price history to forecast next day prices. As a result, more data was favored, since that non-standardized data point may have been more indicative of the overall macro pattern that are commonly found in business cycles.

Summary

This section compares the results of the LSTM model's predictions to the SOXX's past price data. Before it was used on SOXX, the LSTM model was first used on the S&P 500 ETF (SPY). The model showed how each asset was moving in general. Python was used as the code, which can be found in the appendix. The LSTM model can predict daily changes in the SOXX price with an accuracy of about 69%. Applying these daily predictions would have yielded a return of 1,420.70% excluding transaction costs, which are larger over a buy and hold strategy of 38.93% for the SOXX ETF and a 21.48% return for the SPY ETF during the same time periods.

Chapter 5: Concluding the Study

Summary of the Study

LSTM was used to make a model of the iShares Semiconductor ETF (SOXX) stock price and make predictions about it. The SOXX's historical data were turned into a rolling sequence that went from the 4180 daily closing prices to an extra 465 daily closing prices that were not in the sample. Compared to 50 percent random chance, the LSTM model was 19.7 % more accurate at predicting stock returns than % random chance. The work showed how good LSTM is at predicting the stock market in SOXX, which is mechanical but less predictable because the results of turning the hyperparameters are different each time.

Ethical Dimensions

The study applied strict ethical standard in an attempt to show all research done in a transparent manner. Only public pricing data was used in the whole course of this study.

Overview of the Population and Sampling Method

The entire study with pilot testing used data from 2000 to 2019. The goal was to give the LSTM model a large number of market cycles to learn from. In this case, two market cycles over the span of 20 years. The application of training the model against testing its forecast ability was done with 90% of the 20 year data for training the model and the remaining for out of sample forecasts. This 90/10 split of sampling data is common for LSTM modeling.

Limitations

Due to using public historical stock price data, the limitations were few. The data was clean in the sense that there were no missing values and the data points were true. To format the data for the LSTM model, the closing price data was transformed through MinMax scaling, which transforms the data points between 0 to 1.

Findings

The hypothesis tests were not done in the usual way by using p-values. Instead, they were more like suggestions and guesses. The Python code in the appendix can be used to get the same results every time.

Research Question 1: Does LSTM with SOXX prices offer accurate forecasting?

First Hypothesis - Test (HA1): LSTM models do make accurate (>50%) predictions. The LSTM models made predictions that were more accurate than 50%, which is better than guessing at random.

Research Question 2: Can LSTM SOXX price forecasts be deliver higher returns than buy and hold?

Second Hypothesis - Test HA2: LSTM models do produce higher absolute returns than buy-and-hold strategies. When the predicted returns were used without trading costs, the daily returns were greater than a factor of 10. The returns each day were returns that added up.

Research Question 3: How sample size or training period LSTM forecasting accuracy?

Third Hypothesis - Test H03: The accuracy of LSTM forecasts (= +/-10%) is not affected by the size of the sample. The study looked at data from almost 20 years. To test the sample size and make sure at least one business cycle was caught, the sample data size was cut down to 10 years. From a sample size of 10 years, the LSTM model had a similar daily accuracy of 69.48%, and from a sample size of 5 years, it had a similar daily accuracy of 69.11%.

Reflection

Before completing this study, I believed that prior stock prices could help predict future stock prices. Due to working in credit analysis in a bank, historical trends aided in forecasting future returns. Yet I was unsure how often a model could produce superior returns to standard financial market benchmarks. To my surprise, the LSTM model acts as a decent forecasting tool for predicting daily stock prices on the next day. One casual fallacy I had was the need for more data to predict future prices. Instead of using 20 years' worth of data, even 5 years proved

sufficient to produce accurate daily forecasts. I suspect that macroeconomic trends may have less of an effect over more recent price history on a daily time frame.

Recommendations

On a practical level, 5-year data would be the best approach. As more data is given to the LSTM model, a longer training time is needed, which slows down the time for forecasted results. Analyzing one stock on a daily time frame would not take too long, but every saved minute counts for superior returns on a larger scale.

Suggestions for Future Research

From other studies in machine learning, ensemble models and combining various machine learning models into one is generally known to increase performance. I wonder what other machine learning models can be connected to improve daily forecasting accuracy. In finance, it is well known that stock prices are affected by other variables not seen in price. I would also like to add how negative and positive news affects the daily forecasting accuracy taken in conjunction with the LSTM model.

Concluding the Study

The purpose of this research was to study the predictive power of LSTM models on forecasting SOXX prices. Prior time series models along with machine learning have shown promising results but none before was used on the semiconductor ETF. The surprising result came from the high accuracy of the model along with a superior performance over buy and hold strategies. For further applications, researchers and practioners may look into combining this model with various other models or attempt other LSTM neural network architecture.

References

Aubry, M., & Renou-Maissant, P. (2014). Semiconductor industry cycles: Explanatory factors and forecasting. *Economic Modelling*, *39*, 221–231. https://doi.org/10.1016/j.econmod.2014.02.039

Altché, F., & de La Fortelle, A. (2017, October). An LSTM network for highway trajectory prediction. In 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC) (pp. 353-359). IEEE.

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2016). The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery, 31(3), 606-660. doi:10.1007/s10618-016-0483-9

Bandara, K., Bergmeir, C., & Smyl, S. (2020). Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. Expert Systems with Applications, 140, 112896. doi:10.1016/j.eswa.2019.112896

BCC Publishing. (2021). Machine learning: Global markets to 2026. Machine Learning Market Size, Share & Growth Analysis Report. Retrieved March 30, 2022, from https://www.bccresearch.com/market-research/information-technology/machine-learning-global-markets.html

Beers, B. (2020, August 29). How a Buy-and-Hold Strategy Works. Retrieved from https://www.investopedia.com/terms/b/buyandhold.asp

Bhandari, P. (2020, July 30). What is Qualitative Research?: Methods & Examples. Retrieved from https://www.scribbr.com/methodology/qualitative-research/

Blackrock. (2021). IShares PHLX Semiconductor ETF. Retrieved from https://www.ishares.com/us/products/239705/ishares-phlx-semiconductor-etf

Bouktif, S., Fiaz, A., Ouni, A., & Serhani, M. A. (2018). Optimal deep learning lstm model for electric load forecasting using feature selection and genetic algorithm: Comparison with machine learning approaches. Energies, 11(7), 1636.

Brownlee, J. (2020, August 14). A Gentle Introduction to Autocorrelation and Partial Autocorrelation. Retrieved from https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/

Calvi, G. G., Lucic, V., & Mandic, D. P. (2019, May). Support tensor machine for financial forecasting. In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 8152-8156). IEEE.

Canizo, M., Triguero, I., Conde, A., & Onieva, E. (2019). Multi-head CNN–RNN for multi-time series anomaly detection: An industrial case study. Neurocomputing, 363, 246-260.

CFI. (2020, November 20). Alpha - Learn How to Calculate and Use Alpha in Investing. Retrieved from https://corporatefinanceinstitute.com/resources/knowledge/finance/alpha/

Chen, J. (2020, September 28). Exchange Traded Fund – ETFs. Retrieved from https://www.investopedia.com/terms/e/etf.asp

Chen, K., Zhou, Y., & Dai, F. (2015, October). A LSTM-based method for stock returns prediction: A case study of China stock market. In 2015 IEEE international conference on big data (big data) (pp. 2823-2824). IEEE.

Chen, W., Pourghasemi, H. R., Kornejady, A., & Zhang, N. (2017). Landslide spatial modeling: Introducing new ensembles of ANN, MaxEnt, and SVM machine learning techniques. Geoderma, 305, 314-327.

Christoffersen, P., Jacobs, K., & Ornthanalai, C. (2013). GARCH option valuation: theory and evidence. The Journal of Derivatives, 21(2), 8-41.

Corrêa, J. M., Neto, A. C., Júnior, L. T., Franco, E. M. C., & Faria Jr, A. E. (2016). Time series forecasting with the WARIMAX-GARCH method. Neurocomputing, 216, 805-815.

Dai, S., Niu, D., & Li, Y. (2018). Daily peak load forecasting based on complete ensemble empirical mode decomposition with adaptive noise and support vector machine optimized by modified grey wolf optimization algorithm. Energies, 11(1), 163.

Deng, W., Yao, R., Zhao, H., Yang, X., & Li, G. (2019). A novel intelligent diagnosis method using optimal LS-SVM with improved PSO algorithm. Soft Computing, 23(7), 2445-2462.

DiPietro, R., & Hager, G. D. (2020). Deep learning: RNNs and LSTM. In Handbook of medical image computing and computer assisted intervention (pp. 503-519). Academic Press.

Dong, W., Huang, Y., Lehane, B., & Ma, G. (2020). XGBoost algorithm-based prediction of concrete electrical resistivity for structural health monitoring. Automation in Construction, 114, 103155.

Du, Y. (2018, June). Application and analysis of forecasting stock price index based on a combination of ARIMA model and BP neural network. In 2018 Chinese Control And Decision Conference (CCDC) (pp. 2854-2857). IEEE.

Duan, Y., Yisheng, L. V., & Wang, F. Y. (2016, November). Travel time prediction with LSTM neural network. In 2016 IEEE 19th international conference on intelligent transportation systems (ITSC) (pp. 1053-1058). IEEE.

Dutta, A. (2014) Modelling Volatility: Symmetric or Asymmetric GARCH Models? Journal of Statistics: Advances in Theory and Applications, 12, 99-108

Fattah, J., Ezzine, L., Aman, Z., El Moussami, H., & Lachhab, A. (2018). Forecasting of demand using ARIMA model. International Journal of Engineering Business Management, 10, 1847979018808673.

Gallicchio, C., Micheli, A., & Pedrelli, L. (2018). Comparison between DeepESNs and gated RNNs on multivariate time-series prediction. arXiv preprint arXiv:1812.11527.

Geeksforgeeks. (n.d.). Python: Pandas DataFrame. Retrieved from https://www.geeksforgeeks.org/python-pandas-dataframe/

Gong, X., Si, Y. W., Fong, S., & Biuk-Aghai, R. P. (2016). Financial time series pattern matching with extended UCR suite and support vector machine. Expert Systems with Applications, 55, 284-296.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2016). LSTM: A search space odyssey. IEEE transactions on neural networks and learning systems, 28(10), 2222-2232.

Gumelar, A. B., Setyorini, H., Adi, D. P., Nilowardono, S., Widodo, A., Wibowo, A. T., ... & Christine, E. (2020, September). Boosting the Accuracy of Stock Market Prediction using XGBoost and Long Short-Term Memory. In 2020 International Seminar on Application for Technology of Information and Communication (iSemantic) (pp. 609-613). IEEE.

Hargrave, M. (2020, December 14). How Deep Learning Can Help Prevent Financial Fraud. Retrieved from https://www.investopedia.com/terms/d/deep-learning.asp#:~:text=Deep learning is an AI,is both unstructured and unlabeled.

Harvey, A., & Sucarrat, G. (2014). EGARCH models with fat tails, skewness and leverage. Computational Statistics & Data Analysis, 76, 320-338.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9, 1735–1780.

Hyndman, R. J., Koehler, A. B., Leeds, M., Ord, J. K., & Snyder, R. D. (2005). Time series forecasting: The case for the single source of error state space. , Vic.

Idrees, S. M., Alam, M. A., & Agarwal, P. (2019). A prediction approach for stock market volatility based on time series data. IEEE Access, 7, 17287-17298.

Jabeur, S. B., Mefteh-Wali, S., & Viviani, J. L. (2021). Forecasting gold price with the XGBoost algorithm and SHAP interaction values. Annals of Operations Research, 1-21.

Jackson, E. A., Sillah, A., & Tamuke, E. (2018). Modelling monthly headline consumer price index (HCPI) through seasonal Box-Jenkins methodology. International Journal of Sciences, 7(01), 51-56.

Jaramillo, J., Velasquez, J. D., & Franco, C. J. (2017). Research in financial time series forecasting with SVM: Contributions from literature. IEEE Latin America Transactions, 15(1), 145-153.

Javed, F., & Mantalos, P. (2013). GARCH-type models and performance of information criteria. Communications in Statistics-Simulation and Computation, 42(8), 1917-1933.

Junior, P. R., Salomon, F. L. R., & de Oliveira Pamplona, E. (2014). ARIMA: An applied time series forecasting model for the Bovespa stock index. Applied Mathematics, 5(21), 3383.

Kalantar, B., Pradhan, B., Naghibi, S. A., Motevalli, A., & Mansor, S. (2018). Assessment of the effects of training data selection on the landslide susceptibility mapping: a comparison between support vector machine (SVM), logistic regression (LR) and artificial neural networks (ANN). Geomatics, Natural Hazards and Risk, 9(1), 49-69.

Karita, S., Chen, N., Hayashi, T., Hori, T., Inaguma, H., Jiang, Z., ... & Zhang, W. (2019, December). A comparative study on transformer vs rnn in speech applications. In 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU) (pp. 449-456). IEEE.

Kewat, P., Sharma, R., Singh, U., & Itare, R. (2017, April). Support vector machines through financial time series forecasting. In 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA) (Vol. 2, pp. 471-477). IEEE.

Khairalla, M. A., & Ning, X. (2017, October). Financial Time Series Forecasting Using Hybridized Support Vector Machines and ARIMA Models. In Proceedings of the 2017 International Conference on Wireless Communications, Networking and Applications (pp. 94-98).

Khalil, K., Eldash, O., Kumar, A., & Bayoumi, M. (2019). Economic LSTM approach for recurrent neural networks. IEEE Transactions on Circuits and Systems II: Express Briefs, 66(11), 1885-1889.

Kong, W., Dong, Z. Y., Jia, Y., Hill, D. J., Xu, Y., & Zhang, Y. (2017). Short-term residential load forecasting based on LSTM recurrent neural network. IEEE Transactions on Smart Grid, 10(1), 841-851.

Kumar, M., & Thenmozhi, M. (2014). Forecasting stock index returns using ARIMA-SVM, ARIMA-ANN, and ARIMA-random forest hybrid models. International Journal of Banking, Accounting and Finance, 5(3), 284-308.

Li, S., & Zhang, X. (2020). Research on orthopedic auxiliary classification and prediction model based on XGBoost algorithm. Neural Computing and Applications, 32(7), 1971-1979.

Li, S., Li, W., Cook, C., Zhu, C., & Gao, Y. (2018). Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 5457-5466).

Li, W., Yin, Y., Quan, X., & Zhang, H. (2019). Gene expression value prediction based on XGBoost algorithm. Frontiers in genetics, 10, 1077.

Li, Y., Zhu, Z., Kong, D., Han, H., & Zhao, Y. (2019). EA-LSTM: Evolutionary attention-based LSTM for time series prediction. Knowledge-Based Systems, 181, 104785.

Lim, C. M., & Sek, S. K. (2013). Comparing the performances of GARCH-type models in capturing the stock market volatility in Malaysia. Procedia Economics and Finance, 5, 478-487.

Liu, R., & Yang, L. (2016). Spline estimation of a semiparametric GARCH model. Econometric Theory, 32(4), 1023-1054.

Liu, W., & Chyi, Y. (2006). A Markov regime-switching model for the semiconductor industry cycles. Economic Modelling, 23(4), 569-578. doi:10.1016/j.econmod.2006.02.007

Liu, Y., Gong, C., Yang, L., & Chen, Y. (2020). DSTP-RNN: A dual-stage two-phase attention-based recurrent neural network for long-term and multivariate time series prediction. Expert Systems with Applications, 143, 113082.

Maciel, L., Gomide, F., & Ballini, R. (2016). Evolving fuzzy-GARCH approach for financial volatility modeling and forecasting. Computational Economics, 48(3), 379-398.

Manaswi, N. K. (2018). Rnn and lstm. In Deep Learning with Applications Using Python (pp. 115-126). Apress, Berkeley, CA.

Manavalan, B., Shin, T. H., & Lee, G. (2018). PVP-SVM: sequence-based prediction of phage virion proteins using a support vector machine. Frontiers in microbiology, 9, 476.

Mehtab, S., & Sen, J. (2020). Stock Price Prediction Using CNN and LSTM-Based Deep Learning Models. 2020 International Conference on Decision Aid Sciences and Application (DASA). doi:10.1109/dasa51403.2020.9317207

Merrill, W., Weiss, G., Goldberg, Y., Schwartz, R., Smith, N. A., & Yahav, E. (2020). A formal hierarchy of RNN architectures. arXiv preprint arXiv:2004.08500.

Miao, Y., Gowayyed, M., & Metze, F. (2015, December). EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. In 2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU) (pp. 167-174). IEEE.

Mitchell, R., & Frank, E. (2017). Accelerating the XGBoost algorithm using GPU computing. PeerJ Computer Science, 3, e127.

Mo, H., Sun, H., Liu, J., & Wei, S. (2019). Developing window behavior models for residential buildings using XGBoost algorithm. Energy and Buildings, 205, 109564.

Nava, N., Di Matteo, T., & Aste, T. (2018). Financial time series forecasting using empirical mode decomposition and support vector regression. Risks, 6(1), 7.

Ninja, F. (2019, February 09). Data Confirms Grim Truth: 70-80% of Retail Traders are Unprofitable. Retrieved from https://www.babypips.com/news/almost-80-percent-of-retail-traders-are-unprofitable

Okasha, M. K. (2014). Using support vector machines in financial time series forecasting. International Journal of Statistics and Applications, 4(1), 28-39

Paliari, I., Karanikola, A., & Kotsiantis, S. (2021, July). A comparison of the optimized LSTM, XGBOOST and ARIMA in Time Series forecasting. In 2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA) (pp. 1-7). IEEE.

Pan, B. (2018, February). Application of XGBoost algorithm in hourly PM2. 5 concentration prediction. In IOP conference series: Earth and environmental science (Vol. 113, No. 1, p. 012127). IOP publishing.

Panait, I., & Slavescu, E. O. (2012). Using GARCH-IN-mean model to investigate volatility and persistence at different frequencies for Bucharest Stock Exchange during 1997-2012. Theoretical & Applied Economics, 19(5).

Pyo, S., Lee, J., Cha, M., & Jang, H. (2017). Predictability of machine learning techniques to forecast the trends of market index prices: Hypothesis testing for the Korean stock markets. PloS one, 12(11), e0188107.

Qin, Y., Song, D., Chen, H., Cheng, W., Jiang, G., & Cottrell, G. (2017). A dual-stage attention-based recurrent neural network for time series prediction. arXiv preprint arXiv:1704.02971.

Rapidapi. (2020, February 06). What is an API Call? (API Call Definition): API Glossary. Retrieved from https://rapidapi.com/blog/api-glossary/api-call/

Reider, N., & Fodor, G. (2012). A distributed power control and mode selection algorithm for D2D communications. EURASIP Journal on Wireless Communications and Networking, 2012(1), 1-25.

Sezer, O. B., Gudelek, M. U., & Ozbayoglu, A. M. (2020). Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. Applied Soft Computing, 90, 106181.

Shen, J., Shafiq, M.O. (2020). Short-term stock market price trend prediction using a comprehensive deep learning system. Journal of Big Data. doi.org/10.1186/s40537-020-00333-6

Shen, T., Zhou, T., Long, G., Jiang, J., Pan, S., & Zhang, C. (2018, April). Disan: Directional self-attention network for rnn/cnn-free language understanding. In Proceedings of the AAAI conference on artificial intelligence (Vol. 32, No. 1).

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D: Nonlinear Phenomena, 404, 132306.

Shin, D., Lee, J., Lee, J., & Yoo, H. J. (2017, February). 14.2 DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks. In 2017 IEEE International Solid-State Circuits Conference (ISSCC) (pp. 240-241). IEEE.

Siami-Namini, S., & Namin, A. S. (2018). Forecasting economics and financial time series: ARIMA vs. LSTM. arXiv preprint arXiv:1803.06386.

Siami-Namini, S., Tavakoli, N., & Namin, A. S. (2018, December). A comparison of ARIMA and LSTM in forecasting time series. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 1394-1401). IEEE.

Smith, T. (2020, September 16). Autocorrelation. Retrieved from https://www.investopedia.com/terms/a/autocorrelation.asp

Snipes, M., & Taylor, D. C. (2014). Model selection and Akaike Information Criteria: An example from wine ratings and prices. Wine Economics and Policy, 3(1), 3-9.

Su, Z., Xie, H., & Han, L. (2020). Multi-Factor RFG-LSTM Algorithm for Stock Sequence Predicting. Computational Economics. doi:10.1007/s10614-020-10008-2

Sun, S., Wei, Y., & Wang, S. (2018, June). AdaBoost-LSTM ensemble learning for financial time series forecasting. In International Conference on Computational Science (pp. 590-597). Springer, Cham.

Tao, P., Sun, Z., & Sun, Z. (2018). An improved intrusion detection algorithm based on GA and SVM. Ieee Access, 6, 13624-13631.

Tian, Y., Zhang, K., Li, J., Lin, X., & Yang, B. (2018). LSTM-based traffic flow prediction with missing data. Neurocomputing, 318, 297-305.

Ţiţan, A. G. (2015). The Efficient Market Hypothesis: Review of Specialized Literature and Empirical Research. Procedia Economics and Finance, 32, 442-449. doi:10.1016/s2212-5671(15)01416-1

Tokgöz, A., & Ünal, G. (2018, May). A RNN based time series approach for forecasting turkish electricity load. In 2018 26th Signal Processing and Communications Applications Conference (SIU) (pp. 1-4). IEEE.

Varatharajan, R., Manogaran, G., & Priyan, M. K. (2018). A big data classification approach using LDA with an enhanced SVM method for ECG signals in cloud computing. Multimedia Tools and Applications, 77(8), 10195-10215.

Venna, Siva RamaKrishna Reddy & Tavanaei, Amirhossein & Gottumukkala, Raju & Raghavan, Vijay & Maida, Anthony & Nichols, Stephen. (2018). A Novel Data-Driven Model for Real-Time Influenza Forecasting. IEEE Access. 10.1109/ACCESS.2018.2888585.

Walter, M., & Andersen, C. (2016). Indigenous Statistics: A quantitative research methodology. Routledge.

Wang, Y., & Guo, Y. (2020). Forecasting method of stock market volatility in time series data based on a mixed model of ARIMA and XGBoost. China Communications, 17(3), 205-221.

Wang, Y., Xiang, Y., Lei, X., & Zhou, Y. (2021). Volatility analysis based on GARCH-type models: Evidence from the Chinese stock market. Economic Research-Ekonomska Istraživanja, 1-25.

Weiss, G., Goldberg, Y., & Yahav, E. (2018). On the practical computational power of finite precision rnns for language recognition. arXiv preprint arXiv:1805.04908.

Weng, B., Martinez, W., Tsai, Y., Li, C., Lu, L., Barth, J. R., & Megahed, F. M. (2018). Macroeconomic indicators alone can predict the monthly closing price of major U.S. indices: Insights from artificial intelligence, time-series analysis and hybrid models. Applied Soft Computing, 71, 685-697. doi:10.1016/j.asoc.2018.07.024

Wetering, S. V. (2020, November 13). Why Do Hedge Funds Fail? Blame Poor Operations Management. Retrieved from https://www.empaxis.com/blog/reasons-hedge-funds-fail

Xiao, C., Xia, W., & Jiang, J. (2020). Stock price forecast based on combined model of ARI-MA-LS-SVM. Neural Computing and Applications, 32(10), 5379-5388.

Xue, H., Huynh, D. Q., & Reynolds, M. (2018, March). SS-LSTM: A hierarchical LSTM model for pedestrian trajectory prediction. In 2018 IEEE Winter Conference on Applications of Computer Vision (WACV) (pp. 1186-1194). IEEE.

Yang, R., Singh, S. K., Tavakkoli, M., Amiri, N., Yang, Y., Karami, M. A., & Rai, R. (2020). CNN-LSTM deep learning architecture for computer vision-based modal frequency detection. Mechanical Systems and signal processing, 144, 106885.

Yin, W., Kann, K., Yu, M., & Schütze, H. (2017). Comparative study of CNN and RNN for natural language processing. arXiv preprint arXiv:1702.01923.

Yu, W., Kim, I. Y., & Mechefske, C. (2021). Analysis of different RNN autoencoder variants for time series classification and machine prognostics. Mechanical Systems and Signal Processing, 149, 107322.

Yuan, G., Zhang, T., Zhang, W., & Li, H. (2021, August). Analysis of Stock Price Based On the XGBoost Algorithm With EMA-19 and SMA-15 Features. In 2021 IEEE International Conference on Computer Science, Artificial Intelligence and Electronic Engineering (CSAIEE) (pp. 1-4). IEEE.

Yucong, W., & Bo, W. (2020, April). Research on EA-xgboost hybrid model for building energy prediction. In Journal of Physics: Conference Series (Vol. 1518, No. 1, p. 012082). IOP Publishing.

Zhang, L., Zhu, G., Mei, L., Shen, P., Shah, S. A. A., & Bennamoun, M. (2018, December). Attention in convolutional LSTM for gesture recognition. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (pp. 1957-1966).

Zhang, R., Li, B., & Jiao, B. (2019, April). Application of XGboost algorithm in bearing fault diagnosis. In IOP Conference Series: Materials Science and Engineering (Vol. 490, No. 7, p. 072062). IOP Publishing.

Zhang, X., Liang, X., Zhiyuli, A., Zhang, S., Xu, R., & Wu, B. (2019, July). AT-LSTM: An attention-based LSTM model for financial time series prediction. In IOP Conference Series: Materials Science and Engineering (Vol. 569, No. 5, p. 052037). IOP Publishing

Zhao, Z., Chen, W., Wu, X., Chen, P. C., & Liu, J. (2017). LSTM network: a deep learning approach for short-term traffic forecast. IET Intelligent Transport Systems, 11(2), 68-75.

Zheng, H., Yuan, J., & Chen, L. (2017). Short-term load forecasting using EMD-LSTM neural networks with a Xgboost algorithm for feature importance evaluation. Energies, 10(8), 1168.

Zhou, Y., Li, T., Shi, J., & Qian, Z. (2019). A CEEMDAN and XGBOOST-based approach to forecast crude oil prices. Complexity, 2019.

Appendix A: Tables

Table A1

Date	Open	High	Low	Close	Adj Close	Volume
2001-07-13	72.300003	73.589996	70.580002	72.650002	61.689522	201100
2001-07-16	71.349998	71.550003	68.440002	68.440002	58.114632	60000
2001-07-17	67.949997	71.250000	67.900002	71.250000	60.500717	95700
2001-07-18	69.099998	69.610001	68.300003	68.300003	57.995770	100600
2001-07-19	72.400002	72.699997	71.199997	71.199997	60.458244	135000
...	...	...	...	...	...	...
2019-12-23	253.289993	253.419998	251.800003	252.119995	246.987549	452100
2019-12-24	252.679993	252.740005	251.250000	252.529999	247.389221	168100
2019-12-26	253.600006	253.600006	251.880005	252.820007	247.673294	218000
2019-12-27	253.860001	253.860001	251.470001	252.320007	247.183472	368000
2019-12-30	252.259995	252.259995	248.699997	250.419998	245.322144	372900

4645 rows × 6 columns

Notes. The table above is a snapshot of the price history of the semiconductor index used from Yahoo Finance for the time series study.

Table A2

Date	Open	High	Low	Close	Adj Close	Volume
2000-01-03	148.250000	148.250000	143.875000	145.437500	96.555077	8164300
2000-01-04	143.531250	144.062500	139.640625	139.750000	92.779175	8089800
2000-01-05	139.937500	141.531250	137.250000	140.000000	92.945122	12177900
2000-01-06	139.625000	141.500000	137.750000	137.750000	91.451416	6227200
2000-01-07	140.312500	145.750000	140.062500	145.750000	96.762589	8066500
...	...	...	...	...	...	...
2019-12-23	321.589996	321.649994	321.059998	321.220001	310.219513	52990000
2019-12-24	321.470001	321.519989	320.899994	321.230011	310.229156	20270000
2019-12-26	321.649994	322.950012	321.640015	322.940002	311.880615	30911200
2019-12-27	323.739990	323.799988	322.279999	322.859985	311.803284	42528800
2019-12-30	322.950012	323.100006	320.549988	321.079987	310.084290	49729100

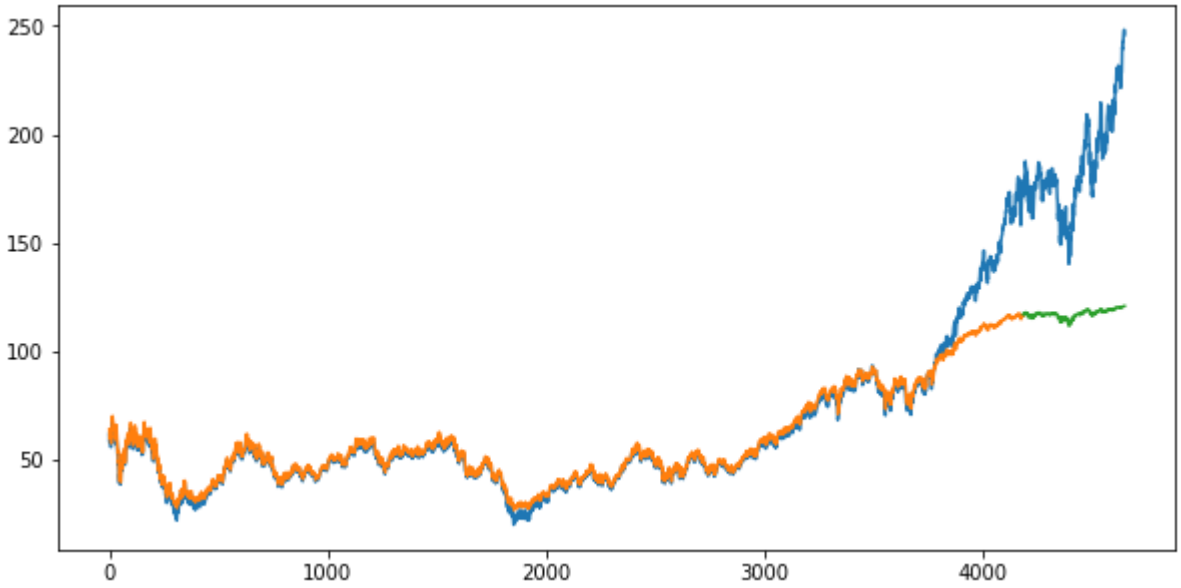
5030 rows × 6 columns

Notes. The table above is a snapshot of the price history of the S&P 500 ETF (SPY) used from Yahoo Finance for the pilot study.

Appendix B: Figures

Figure B1

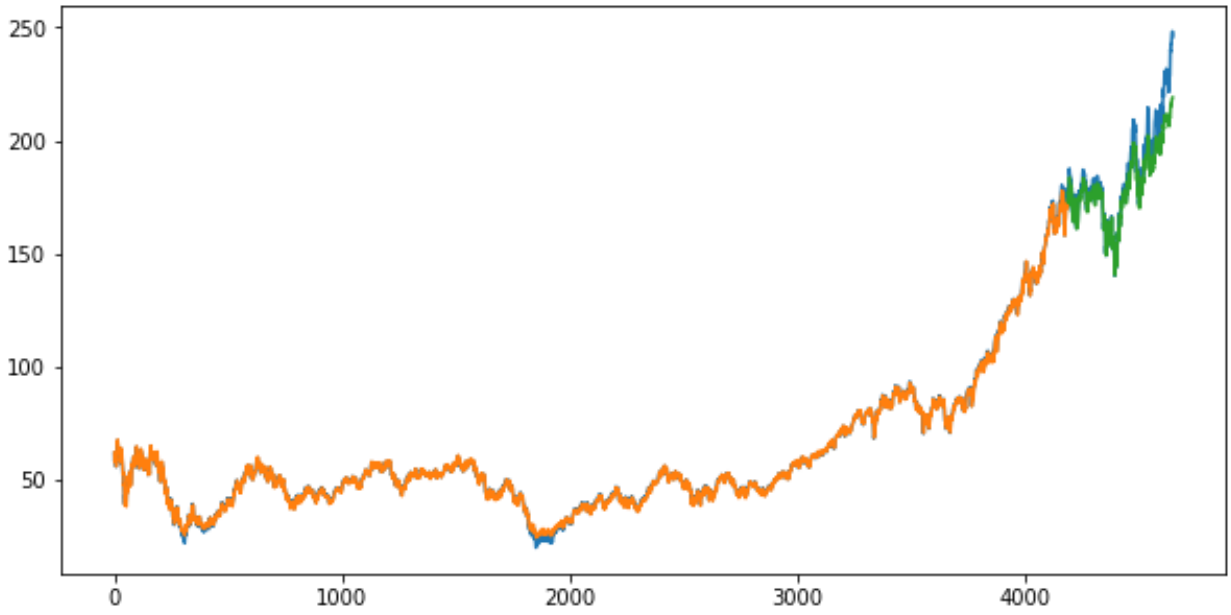
LSTM model applied on SPY ETF from 2000-01-03 to 2019-12-30.



Notes. The blue line is the historical price history of the SPY. The orange line is the performance of the LSTM model on training data. The green line is the out of sample forecast of the LSTM model, which would be the true out of sample forecast.

Figure B2

LSTM model applied on SOXX index from 2001-07-13 to 2019-12-30.



Notes. The blue line is the historical price history of the SOXX. The orange line is the performance of the LSTM model on training data. The green line is the out of sample forecast of the LSTM model, which would be the true out of sample forecast.

Appendix C: Instruments

Anaconda was used to process the data and develop the time series models. Anaconda is an open-source distribution of the Python and R programming languages for scientific computing. Within Anaconda, the Python programming languages were applied with Jupyter Notebooks, an integrated development environment. To further speed the creation of the time series models, Google's open-sourced Tensorflow package was used for machine learning purposes such as making deep neural networks for time series.

The CSV data points taken from Python and Yahoo Finance via Yfinance will have the headers Date, Open, High, Low, Close, Volume, and Adj Close. Sample data points from Yfinance below:

Date	Open	High	Low	Close	Adj Close	Volume
2001-07-13	72.300003	73.589996	70.580002	72.650002	61.596382	201100
2001-07-16	71.349998	71.550003	68.440002	68.440002	58.026913	60000
2001-07-17	67.949997	71.250000	67.900002	71.250000	60.409397	95700
2001-07-18	69.099998	69.610001	68.300003	68.300003	57.908218	100600
2001-07-19	72.400002	72.699997	71.199997	71.199997	60.366993	135000

Appendix D: IRB Approvals and Consent Form

Consent Form

Since this study did not use human participants, no consent forms were needed.

Statement of Consent

No statements of consent were required for this study.

Participant Bill of Rights

No participants were used in this study.

Appendix E: List of industry standard measures

- R2 is used to figure out how the predicted value compares to the actual value.

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

Where SSres is the total squared residuals from the expected values and SStot is the total squared deviations from the sample mean of the dependent variable. It shows how much of the variation in the dependent variable can be explained by the variation in the independent variable. A high R2 value shows that the model's variance is similar to that of the real values, while a low R2 value shows that the two values are not strongly related.

The most important thing to remember about R-squared is that it does not show if the model can accurately predict what will happen in the future. It shows whether or not the model fits well with the observed values and how well it fits. When the R2 is high, it means that the values that were seen and those that were expected are strongly related.

- When absolute error must be measured, Mean Absolute Error (MAE) is useful. It is simple to understand, but in the case of data with extreme values, it is inefficient. MAPE is also simple to understand and is used to compare different forecast models or datasets because it is a percentage value. MAPE has the same problem as MAE in that it is inefficient when data contains extreme values.

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

The MAE tells us, on average, how far off the forecast is likely to be. MAE = 0 means that the predicted values are correct and that the error statistics are in the same units as the predicted values. The better the model, the lower the MAE value. If the MAE value is zero, the forecast is correct. In other words, when comparing many models, the one with

the lowest MAE is seen as the best. But because MAE doesn't show how big or small the error is, it can be hard to tell the difference between big and small errors.

- Mean Squared Error (MSE)is beneficial when the spread of prediction values is significant and larger values must be punished. However, because it is a squared value, this metric is frequently difficult to comprehend.

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

Where y' is the value that was predicted and y is the real value. The number n shows how many values are in the test set as a whole. MSE is almost always good, and values that are lower are better. Due to the square term, this measure punishes big mistakes or outliers more than small mistakes. The better MSE is, the closer it is to 0. Even though it solves MAE and MAPE extreme value and zero problems, it may be bad in some situations. When there isn't much data, this statistic might overlook problems.

- When the spread is important and bigger values need to be penalized, Root Mean Squared Error (RMSE) is also useful. When compared to MSE, RMSE is easier to interpret because the RMSE number is on the same scale as the projected values.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$$

This statistic is always positive as well, with lower numbers showing better performance. The RMSE number and the projected value have the same unit, which is a benefit of this method. This makes it easier to understand compared to MSE. The RMSE can also be compared to the MAE to see if the forecast is off by a lot, but in a way that doesn't happen very often. The error size is more likely to change if the difference between RMSE and MAE is large. This statistic can hide problems with small amounts of data.

- When dealing with low-volume data, Weighted Mean Absolute Percentage Error (WMAPE) is also useful. WMAPE uses the weight (priority value) of each observation to help incorporate the priority.

$$\text{WMAPE} = \frac{\sum_{t=1}^{n} |A_t - F_t|}{\sum_{t=1}^{n} |A_t|}$$

The current data is shown by A, and the forecast is shown by F. This metric is better than MAPE because it doesn't have the problem of "infinite error."

The WMAPE number goes down as the model's performance goes up. This metric is useful for judging forecasting models when the amount of data is low and each observation has a different priority. Observations that are more important have a higher weight value. As the error in high-priority forecasts grows, so does the WMAPE number.

Appendix G: Python Code used for the current study

```
# -*- coding: utf-8 -*-

"""DBA LSTM Research SOXL.ipynb

"""


!pip install yfinance


# Commented out IPython magic to ensure Python compatibility.

#Python Machine Learning Cookbook - Second Edition

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# %matplotlib inline

import yfinance as yf

import datetime

from random import seed


seed(0)


# Gather data

start = datetime.datetime(2000, 1, 1)

end = datetime.datetime(2019, 12, 31)
```

```
df = yf.download("SOXX", start, end)

df = df.drop(columns=['Open', 'High', 'Low', 'Close', 'Volume'])

df = df.rename(columns={"Adj Close":'Close'})


#rename dataframe for downward code

Data = df

del df


df = yf.download("SOXX", start, end)

df


Data.shape


# Rescale data

from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()

DataScaled = scaler.fit_transform(Data)


# Split training and test data

np.random.seed(7)
```

```
TrainLen = int(len(DataScaled) * 0.90)
TestLen = len(DataScaled) - TrainLen
TrainData = DataScaled[0:TrainLen,:]
TestData = DataScaled[TrainLen:len(DataScaled),:]

print(len(TrainData), len(TestData))

# Construct tested output at timestep t+1
def DatasetCreation(dataset, TimeStep=1):
  DataX, DataY = [], []
  for i in range(len(dataset)-TimeStep-1):
    a = dataset[i:(i+TimeStep), 0]
    DataX.append(a)
    DataY.append(dataset[i+TimeStep, 0])
  return np.array(DataX), np.array(DataY)

# Network modeling prep
TimeStep = 1 #Same as MLP if = 1, should be something more far back...errors at different time steps, need to fix
TrainX, TrainY = DatasetCreation(TrainData, TimeStep)
TestX, TestY = DatasetCreation(TestData, TimeStep)

# Transform input into 3D form
```

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

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


# Import packages for model

from keras.models import Sequential

from keras.layers import LSTM, Dense


# Create sequential model

model = Sequential()

model.add(LSTM(500, input_shape=(1, TimeStep))) #256 w/ 5eopochs seem fine, worse with relu.  500 w/20 epocs is good


model.add(Dense(1, activation='sigmoid')) #maybe I can change this to linear later on

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae']) #metrics orginally accuracy

model.fit(TrainX, TrainY, epochs=40, batch_size=1, verbose=1)

model.summary()


# Evaluate model performance

score = model.evaluate(TrainX, TrainY, verbose=0)

print('Keras Model Loss = ', score[0])

print('Keras Model Accuracy = ', score[1])
```

```
# Make predictions
TrainPred = model.predict(TrainX)
TestPred = model.predict(TestX)

# Rescaled predictions to original asset values
TrainPred = scaler.inverse_transform(TrainPred)
TrainY = scaler.inverse_transform([TrainY])
TestPred = scaler.inverse_transform(TestPred)
TestY = scaler.inverse_transform([TestY])

# Verify predictions by graphing with proper time series shifting to align everything
TrainPredictPlot = np.empty_like(DataScaled)
TrainPredictPlot[:, :] = np.nan
TrainPredictPlot[1:len(TrainPred)+1, :] = TrainPred

TestPredictPlot = np.empty_like(DataScaled)
TestPredictPlot[:, :] = np.nan
TestPredictPlot[len(TrainPred)+(1*2)+1:len(DataScaled)-1, :] = TestPred

# Plot the data and predictions
plt.figure(figsize=(10,5))
plt.plot(scaler.inverse_transform(DataScaled))
```

```
plt.plot(TrainPredictPlot)

plt.plot(TestPredictPlot)

plt.show()
```

Appendix H: Seed Code

```
# -*- coding: utf-8 -*-

"""DBA LSTM Seed Code / Pilot Test SPY Research.ipynb


"""


!pip install yfinance


# Commented out IPython magic to ensure Python compatibility.
#Python Machine Learning Cookbook - Second Edition
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline
import yfinance as yf
import datetime
from random import seed


seed(0)


# Gather data
start = datetime.datetime(2000, 1, 1)
```

```
end = datetime.datetime(2019, 12, 31)

df = yf.download("SPY", start, end)

df


df = df.drop(columns=['Open', 'High', 'Low', 'Close', 'Volume'])

df = df.rename(columns={"Adj Close":'Close'})


#rename dataframe for downward code

Data = df

del df


# Rescale data

from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()

DataScaled = scaler.fit_transform(Data)


# Split training and test data

np.random.seed(7)


TrainLen = int(len(DataScaled) * 0.90)

TestLen = len(DataScaled) - TrainLen
```

```
TrainData = DataScaled[0:TrainLen,:]
TestData = DataScaled[TrainLen:len(DataScaled),:]

print(len(TrainData), len(TestData))

# Construct tested output at timestep t+1
def DatasetCreation(dataset, TimeStep=1):
  DataX, DataY = [], []
  for i in range(len(dataset)-TimeStep-1):
    a = dataset[i:(i+TimeStep), 0]
    DataX.append(a)
    DataY.append(dataset[i+TimeStep, 0])
  return np.array(DataX), np.array(DataY)

# Network modeling prep
TimeStep = 1 #Same as MLP if = 1, should be something more far back...errors at different time steps, need to fix
TrainX, TrainY = DatasetCreation(TrainData, TimeStep)
TestX, TestY = DatasetCreation(TestData, TimeStep)

# Transform input into 3D form
TrainX = np.reshape(TrainX, (TrainX.shape[0], 1, TrainX.shape[1]))
TestX = np.reshape(TestX, (TestX.shape[0], 1, TestX.shape[1]))
```

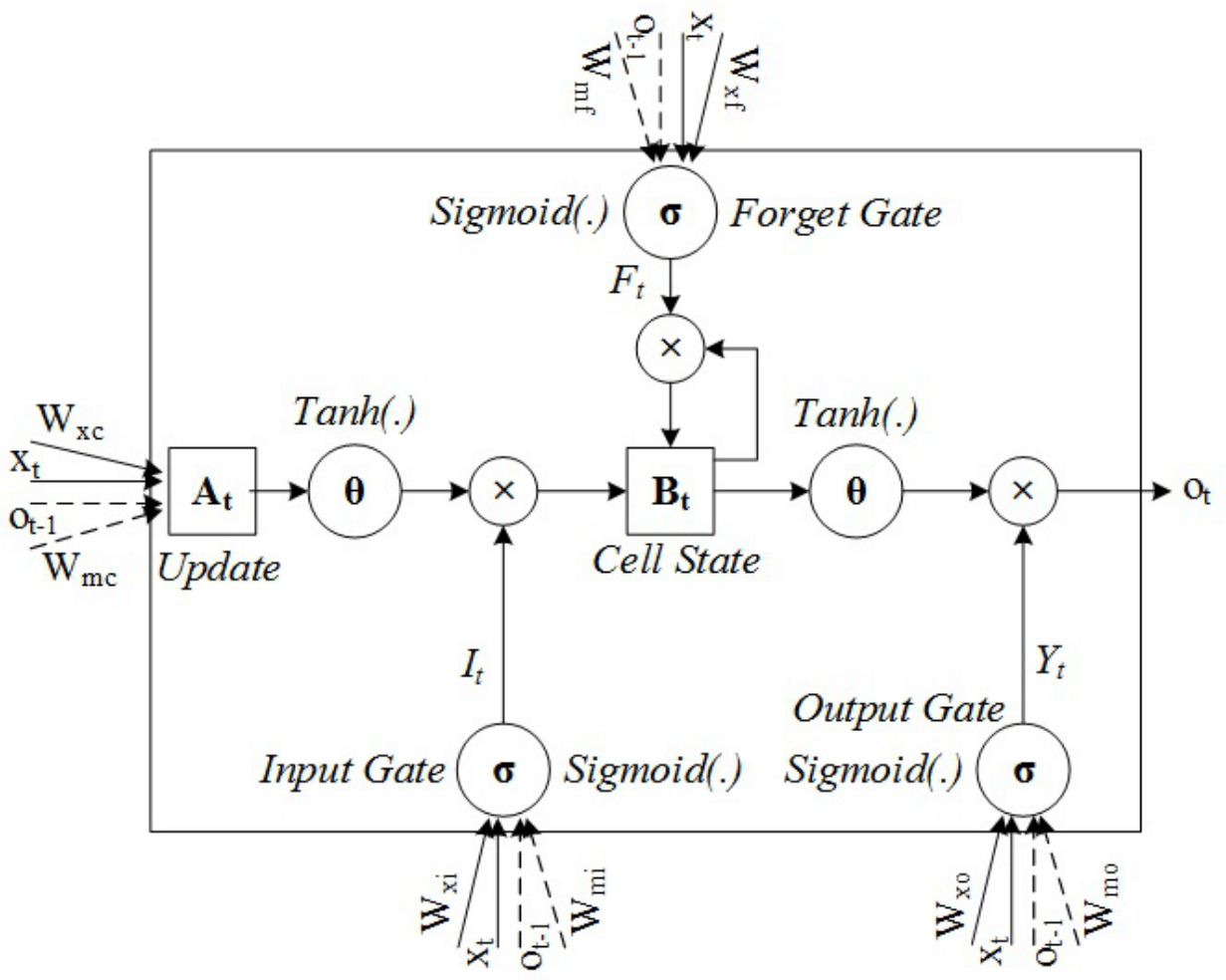
```
# Import packages for model
from keras.models import Sequential
from keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='mae',patience=2)

# Create sequential model
model = Sequential()
model.add(LSTM(500, input_shape=(1, TimeStep))) #256 w/ 5eopochs seem fine, worse with relu.  500 w/20 epocs is good

model.add(Dense(1, activation='sigmoid')) #maybe I can change this to linear later on
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae']) #metrics orginally accuracy
model.fit(TrainX, TrainY, epochs=40, batch_size=1, verbose=1, callbacks=[early_stop])
model.summary()

# Evaluate model performance
score = model.evaluate(TrainX, TrainY, verbose=0)
print('Keras Model Loss = ', score[0])
print('Keras Model Accuracy = ', score[1])
```

```
# Make predictions
TrainPred = model.predict(TrainX)
TestPred = model.predict(TestX)

# Rescaled predictions to original asset values
TrainPred = scaler.inverse_transform(TrainPred)
TrainY = scaler.inverse_transform([TrainY])
TestPred = scaler.inverse_transform(TestPred)
TestY = scaler.inverse_transform([TestY])

# Verify predictions by graphing with proper time series shifting to align everything
TrainPredictPlot = np.empty_like(DataScaled)
TrainPredictPlot[:, :] = np.nan
TrainPredictPlot[1:len(TrainPred)+1, :] = TrainPred

TestPredictPlot = np.empty_like(DataScaled)
TestPredictPlot[:, :] = np.nan
TestPredictPlot[len(TrainPred)+(1*2)+1:len(DataScaled)-1, :] = TestPred

# Plot the data and predictions
plt.figure(figsize=(10,5))
plt.plot(scaler.inverse_transform(DataScaled))
```

```
plt.plot(TrainPredictPlot)

plt.plot(TestPredictPlot)

plt.show()


plt.plot(TestPredictPlot)

plt.show()
```

Appendix I: LSTM Model

The LSTM (Long Short-Term Memory) network is a type of RNN (Recurrent Neural Network) that is often used to learn how to predict what will happen next in a sequence of data. Like other neural networks, LSTM has layers that help it learn and recognize patterns so it can do its job better. The basic way that LSTM works is to keep the information that is needed and get rid of the information that is not needed or useful for making predictions. The following are the parts of a simple LSTM network: forget gate, input gate, and output gate (Venna, et. Al, 2018).



The type of the simple LSTM changes as hidden layers and gates are added. Like in a BI LSTM network, it can be made up of two LSTM that pass information in either the same or opposite way.

Forget Gate

As we've already talked about, one of the main things that the LSTM does is remember and recognize the information that comes into the network and get rid of the information that the network doesn't need to learn data and make predictions. This part of the LSTM is made possible by this gate. It helps decide if information can move through the network's layers. It looks for two different kinds of information from the network: information from the previous layers and information from the presentation layer. The picture above shows a Forget gate circuit, where h and x are pieces of information. This information goes through the sigmoid function, which gets rid of the information that tends to get closer to zero.

Input Gate

By changing the state of the cell, the input gate helps decide how important the information is. Where the forget gate helps get rid of information from the network, the importance of the information is measured by the input gate, which helps the forget function get rid of information that isn't important and other layers learn the information that is important for making predictions. The information passes through the sigmoid and tanh functions. The sigmoid function decides how important each piece of information is, and the tanh function makes the network less biased.

Cell State

The information about weight gained goes through the cell state, and this layer figures out the cell state. In the cell state, the output of the forget gate is multiplied by the output of the input gate. The information that could be lost is multiplied by values that are close to zero. Here in the cell state, the input and output values are added together. This is done to try to keep the cell state up to date with information that is important to the network.

Output Gate

It is the last gate of the circuit that helps the sigmoid function figure out what the next hidden state of the network will be. The updated cell from the cell state is sent to the tanh function, and then the sigmoid function of the output state is used to multiply it. Which helps the information get to the hidden state. This is the last part of the circuit, and it helps the hidden state decide what information it should carry.

Appendix J: Detailed Testing and Modeling Procedures

The outside data and tools used are from open-sourced and public data. The Python code and models were from Google's open-sourced packages. On the other hand, the semiconductor data was publicly available stock data from Yahoo Finance.

The LSTM model tested in this study was a result of multiple iterations of different combinations of hyperparameters. In machine learning, a hyperparameter is a setting that the model can use to affect its performance. Parameters are chosen by the machine learning model itself, while users of the model can select the hyperparameters. One key hyperparameter chosen was the number of neurons in the LSTM network. The Google documentation has a default amount of neurons to be 4. Tested were neurons between 4 to 500, which were chosen arbitrarily. It is noteworthy to state that other studies chosen neurons in the 100s and justified the reason due to their own hyperparameter tuning. Another insight from hyperparameter turning found that early stopping to not be needed for improved forecast results.

The best setting was chosen from the out-of-sample forecast closest to the actual historical values. In machine learning, the portion of the dataset the model trained from does not hold much out of sample performance, while out of sample performance is the best metric for actual use cases.

Hypothesis Tests

First Hypothesis Test

Hypothesis testing utilizes Python. The following is a brief outline of the accuracy of the forecasts test:

H01: LSTM model do not produce accurate (<=50%) forecasts.

HA1: LSTM model do produce accurate (>50%) forecasts.

The hypothesis test involves comparing the accuracy of the forecast higher than random chance, 50%, for an up or down movement. Up or down movement is referencing a prediction higher or lower than the last historical price point. The 50% level was chosen for practical purposes. With a model that forecast greater than 50% accuracy, a trader can take financial derivatives that offer a ratio of 1 to 1 risk-reward set up. Meaning if the trader can have predict better than 50% accuracy, they can set up a trade that offers at one unit of risk for at least one unit

of reward for profit. Effectively, the trader would get odds similar to being the casino in a game of roulette. A lower than 50% accuracy will determine whether to reject the null hypothesis stating in LSTM being not an accurate forecasting model for SOXX. Rejecting the null hypothesis warrants the LSTM model producing accurate forecasts. Forecasting accuracy will be measured in predicting the same up or down movement. For example, if the predicted forecast was a gain of 0.37% and the actual result was a gain of 0.24%, then the model would have made an accurate forecast. While if the predicted forecast was a gain of 0.37% and the actual result was a loss of 0.01%, then the model would have made an inaccurate forecast.

Second Hypothesis Test

Hypothesis testing utilizes Python. The following is a brief outline of the absolute returns of the forecasting model against a buy and hold strategy:

H02: LSTM model do not generate higher absolute returns in comparison to buy and hold.

HA2: LSTM model do generate higher absolute returns in comparison to buy and hold.

The hypothesis test involves an active performance comparison against a benchmark strategy, buy and hold. The performance of the LSTM forecasts will determine whether to reject the null hypothesis stating no superior returns against buy and hold. Rejecting the null hypothesis warrants the LSTM model holding superior performance against buy and hold. The calculation for buy and hold returns will be the starting period of the forecast test set and the end of the forecasting test set. The training period is not counted, since the model has already seen those prices. The test set represents forecasting to real life conditions. The buy and hold return would be calculated the asset ending price divided by the asset beginning price minus 1 to get a percentage return. The forecasting model returns will be calculated as taking the end of day return on an accurate forecast, while having a loss of the end of day return on an inaccurate forecast. On the next day, the returns will be compounded until the end of the testing period and compared against the buy and hold returns.

Third Hypothesis Test

Hypothesis testing utilizes Python. The following is a brief outline of the sample size affecting the forecasting accuracy:

H03: Sample size does not affect LSTM forecasting accuracy (<= +/-10%).

HA3: Sample size does affect LSTM forecasting accuracy (> +/-10%).

The hypothesis test involves various sample sizes against forecasting accuracy. The accuracy from differing sample sizes will determine whether to reject the null hypothesis stating no sample size does not affect LSTM forecasting accuracy. Rejecting the null hypothesis warrants the sample size does affect LSTM forecasting accuracy. The study will use almost 20 years of pricing SOXX data to start, but will also test for 10 and 5 years of data. On a practical level, if less data can be used, then model calculations will be faster and save time.