RICE UNIVERSITY

Financial time series forecasting via RNNs and Wavelet Analysis

By

Michael Demone Jackson

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE

Kathy Ensor (Apr 22, 2022 10:08 CDT)

Katherine Ensor
Chair
Noah G. Harding Professor of Statistics

John Dobelman
Professor in the Practice in Statistics

C. Fred Higgs
John and Ann Doerr Professor of
Mechanical Engineering

HOUSTON, TEXAS

April 2022

ABSTRACT

Financial time series forecasting via RNNs and Wavelet Analysis

by

Michael Demone Jackson

Recent successes in both Artificial Neural Networks (ANN) and wavelets have placed these two methods in the spotlight of quantitative traders seeking the best tool to forecast financial time series. The Wavelet Neural Network (W-NN), a prediction model which combines wavelet-based denoising and ANN, has successfully combined the two strategies in order to make accurate predictions of financial time series. We explore how the most recent formulation of the W-NN model, with the Nonlinear Autoregressive Neural Network with Exogenous variables (NARX), is affected by the choice of wavelet thresholding technique when predicting daily returns of American index futures contracts. We explore how the choice of thresholding technique affects the profitability of two technical trading models based on daily return predictions from a NARX-based W-NN. The purpose of this research is twofold: it compares the effect of different wavelet thresholding techniques on a NARX-based W-NN's forecasting ability on 1-day returns of American index futures contracts and offers two easy-to-implement trading strategies.

In the second part of the thesis, we formulate a hybrid NARX-based seasonal predictive model, Seasonal Nonlinear Autoregressive Neural Network with Exogenous Variables (S-NARX), for end-of-day volume, where end-of-day volume is directly driven by the end of the day auctions. The S-NARX model will seek to take advantage of the information found in the data up until the auction time and high-frequency intraday trading volume's diurnal seasonal pattern to predict end-of-day volume. Volume is well known to be a leading indicator of price changes and the two metrics are simultaneously positively correlated. Algorithmic traders rely on accurate volume predictions to deploy algorithmic trading algorithms, especially when utilizing a Volume Weighted Average Price (VWAP) algorithm, that allows the execution of large orders with minimal slippage. Fundamental and quantitative investors are also interested in trading volume because it is a measure of trading intensity and an indicator of market liquidity. The S-NARX augments the NARX with the feature set from a seasonal ARMA(P,Q)[s] and offers quantitative traders a flexible machine learning model for forecasting time series with both longer dependencies and seasonality.

Finally, we develop an R package that provides the traditional NARX network, first introduced in (46), along with the novel seasonal version of the CoFES S-NARX that augments the NARX feature set with the features from an ARMA(P,0)[s] described in (73). The networks are built using the Keras, (4), framework in R and utilize the sequential model from this package.

Contents

Illustrations

Tables

Acknowledgements

I would like to thank my advisor, Kathy Ensor, for her support and guidance throughout this process. This has been an interesting 5+ years with several tail-end events. I cried on you during time series office hours and the first time I presented a paper. I appreciate your patience with me, I know I am a unique learner. I can honestly say that I would have given up after year two had you not stayed in contact with me during my time away from Rice. Thank you for pushing me and being patient as I worked at my pace. You are an amazing teacher and mentor. Thank you so much!

I would also like to thank Dr. Dobelman for his attention and helpful feedback. Thank you for agreeing to be on my committee. I worked with you on several courses and you were my first TA assignment. Thank you for your nurturing nature and dedication to financial statistics. It was a pleasure to watch you interact with students and lovingly guide them to their peak.

I would also like to thank Dr, Higgs for agreeing to be on my committee. We connected officially with AGEP STRIDES but we initially connected in the halls of Duncan. It was always good to be able to look across the room and connect with you in larger meetings when something interesting occurred. Thank you for the work that you do.

I would like to also thank Dr. Kyj and Vanguard for their support of my second project. Dr. Kyj, I appreciate your patience and timely industy and academic guidance as we progressed through the second project.

I would like to thank CoFES and its staff for all of their assistance.

I would like to thank Dr. Raath. It was a pleasure to work with you on my first research project and to study with you. You are a great friend.

I would also like to thank Yifan Zhang, it was a pleasure to work with you on the many projects that we worked on together from research to Dr. Dobelman's course that I audited. I learned a ton from you and I owe you an authentic African American dinner after graduation. Thank you for the lunches and for being an awesome friend.

I would like to thank the research assistants that helped with the projects: Alexander Gallegos, Gazi Fuad, Kelly Liu, Hanxuan Lin, and Cole Rabson. It was awesome to work with you all.

I would like to thank my parents, Brenda Joyce Jackson and Nathaniel Jackson Jr, for their support, guidance, and never-ending love.

I would like to thank Shelly for her support throughout this process. Sometimes a guy just needs to dance salsa and forget about statistical shortcomings. Thank you for sharing this journey with me.

I would like to thank Eric Lyons for being a mentor and providing fellowship during my time in Houston. You were my first official mentor and you introduced me to my first official mentee. I am forever indebted to you, the Collective Generational Mentoring Foundation, and Ujamaa Investment Group for providing me with a sense of community and brotherhood while I studied.

Last but not least, I would like to thank my close friends from my cohort:

Sara, Carly, Emma, and Neil. It was a pleasure to get to know you all and to study and learn statistics with you. Thank you for all of your support and friendship. Thank you for the meals, the drinks, the ball games, and the memories.

Chapter 1

Background

This chapter covers the background for this dissertation work.

1.1 Momentum Trading

Momentum trading consists of a set of popular trading strategies that attempt to detect and exploit changes in an asset's price. Some summarize the momentum strategy as 'buying winners and selling losers'. The momentum effect was discovered in the late 1980s, when traders realized that stocks with above-average recent returns outperformed stocks with below-average recent returns but with the same market risk (18). Momentum trading strategies are very popular and also range in sophistication. For instance, simple momentum strategies can be employed by anyone with access to past financial returns. More complex strategies can incorporate momentum technical indicators and prediction algorithms. Interestingly, finance literature suggests that returns over a short horizon exhibit autocorrelation and link momentum effect with longer periods, short to intermediate, of serial correlation (12).

Machine learning strategies, including the strategy employed in the first study, incorporate technical analysis by introducing technical indicators into

a feature set. Technical indicators, signals produced by calculations that involve volume, price, and/or open interest of an asset, are used by technical analysts to make decisions about different aspects of the investment process. There are three major types of technical indicators: trend, momentum, and sentiment. Trend technical indicators, the most popular category, attempt to quantify the direction, up, down, or sideways, of price movement. Momentum technical indicators attempt to quantify the price velocity or acceleration. Momentum is a leading indicator of change in trend. Sentiment technical indicators attempt to quantify an investor's attitude toward securities and attempt to highlight instances when a security is either oversold or overbought (18). The first study employs the following momentum technical indicators into our neural network's feature set: the Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), MACD Signal Line, MACD Histogram, Stochastic fast %K, Stochastic slow %K, Stochastic %D, and the Ultimate Oscillator.

Generally speaking, the Hurst exponent, first introduced in (35), is used in time series literature to both measure long memory and to gauge the predictability of a series. This method of detecting long memory is based on making estimates of the range that a variable changes over time and was first introduced in the context of long-range dependence in hydrology. According to the original paper, a Hurst Exponent (H) = .5 indicates a series where the current value does not depend on lagged values of the series. Mean reverting series have Hurst exponent's in the range of $0 < H < .5$ and the closer H is

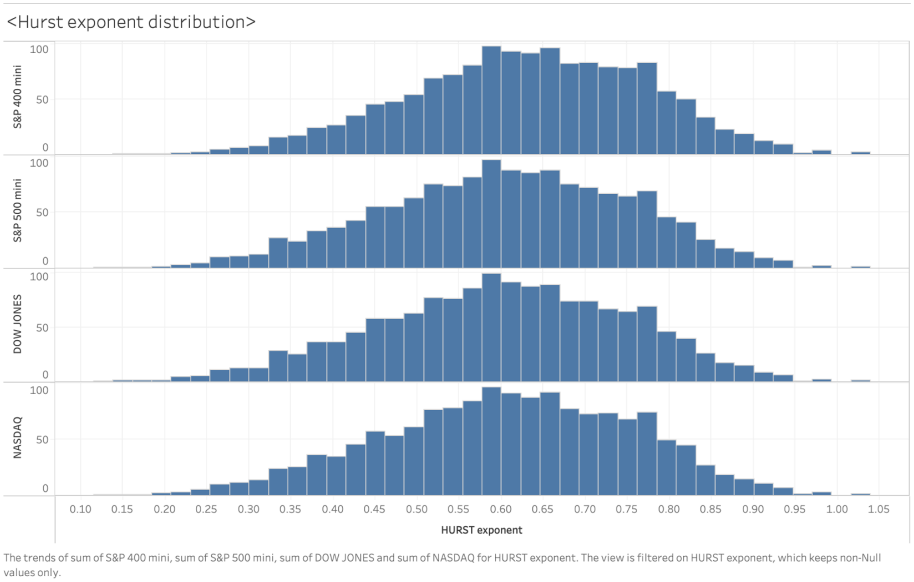


Figure 1.1 : Hurst exponent value distributions for the SP 400 Mini Futures contracts, SP 500 mini futures contracts, Dow Jones mini futures contracts, and NASDAQ mini futures contracts between 2011-2020.

to 0 the stronger the mean-reverting behavior. Hurst exponents in the range $.5 < H < 1$ display trend reinforcing behavior which is stronger the closer H is to 1. Peters,(61) and (62), introduced the Hurst exponent to financial time series via Fractal Markets Hypothesis (FMH). Since then, several methods have been developed to estimate the Hurst exponent of a time series. The second study employs the rescaled range analysis (R/S analysis) method for calculating the Hurst exponent of time series $X_t = X_1, X_2, ...$ with a sample size of 10,000. The Hurst Exponent in the first study was downloaded from Bloomberg with the original data set. The (R/S analysis) method for

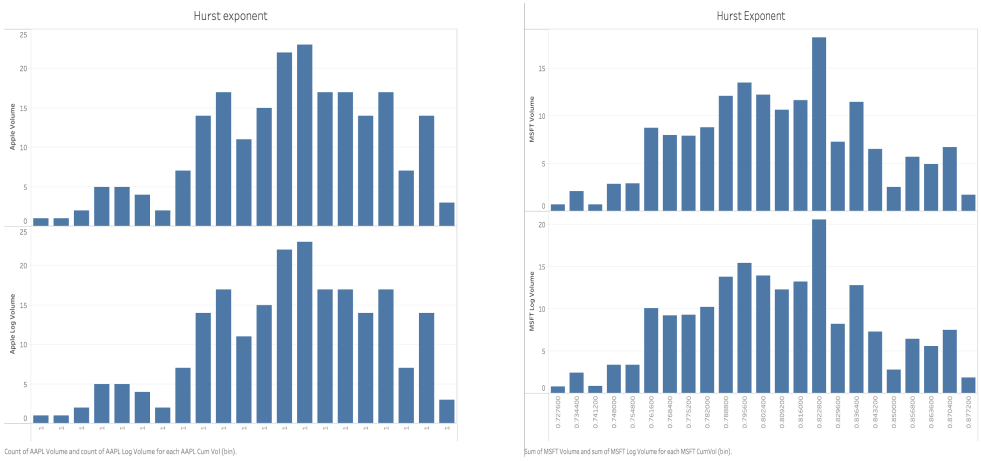


Figure 1.2 : Hurst exponent distribution for Volume(top), log volume (bottom), Apple (left), and Microsoft (right) using samples of size 10,000 between 9/1/2020 and 9/1/2021.

calculating the Hurst exponent is described in Algorithm 1.

Algorithm 1 R/S estimation of Hurst Exponent

Calculate mean value $m = \frac{1}{n}\sum_{i=1}^{n} X_i$

Calculate the mean adjust series $Y_t = X_t - M$

Calculate the cumulative deviate series $Z_t = \sum_{i=1}^{t}$, for $t = 1, 2, ..., n$

Calculate range Series $R_T = max(Z_1, Z_2, ..., Z_t) - min(Z_1, Z_2, ..., Z_t)$ for $t = 1, 2, ..., n$

Calculate Standard deviation series $S_t = \sqrt{\frac{1}{t}\sum_{i=1}^{t}(X_i - \mu)^2} t = 1, 2, ..., n$ where $\mu = \frac{1}{t}\sum_{i=1}^{t} X_i$

Calculate Rescaled range series $(R/S)_t = \frac{R_t}{S_t}$ for $t = 1, 2, ..., n$

According to Hurst, as time increases (R/S) scales by power-law, which suggests

$$(R/S)_t = d \cdot t^H \tag{1.1}$$

where d is a constant and H is the Hurst Exponent which is estimated by plotting log(R/S) versus log (t). The slope of the calculated regression line represents an approximation to the Hurst exponent (66). There are several empirical studies that suggest that time series with larger Hurst exponents are more predictable than time series with Hurst exponent closer to .5, [(42),(47),(64)]. All of the American index futures contract time series had Hurst values that averaged well above .5 for the time period of our experiment. The distribution of their Hurst exponent can be found in Figure 1.1. Similarly, in the second project, we attempt to predict transformations of high-frequency intraday volume and also use the Hurst exponent the detect long memory and gauge the predictability of the two series. As with the first project, these time series also have an average Hurst exponent larger than .5 over the entire data sample, see Figure 1.2.

Denoising is one of the most frequently used properties of wavelets analysis. Most wavelet denoising schemes are based on a thresholding technique applied to the wavelet coefficient that results from the wavelet transformation. A wavelet thresholding method can be distinguished by the choice of transformation, either discreet or continuous, the mother wavelet, which is strongly influenced by the transformation decision, and finally the threshold technique. Market noise can be thought of as the opposite of information

and consist of activities that distort the truth which leads to small insignificant price movements. Noise exists, to some extent, in all financial markets. Furthermore, machine learning-focused methods are less effective in the presence of noise.

RNNs are a class of neural networks that are designed to analyze time series and sequential data. RNNs are made of several sequential recurrent layers that form a directed or undirected graph along a temporal sequence, which enables the network to exhibit temporal dynamic behavior (11). However, analyzing time series with longer dependencies, 5-10 time steps (75), introduces issues with vanishing gradients for RNNs with gradient descent-based training methods. As a result, RNNs were designed to address this shortcoming. There are two popular RNNs, Long short-term memory (LSTM) and Gated Recurrent Unit (GRU), that are designed to mitigate the vanishing gradient issue and allow the network to learn longer dependencies. Of the two, the LSTM, first introduced in (30), has been the more popular of the two networks for analyzing financial time series with longer dependencies due to its superior performance of accurately modeling long and short-term dependencies in financial data (11). However, the LSTM doesn't completely eliminate the vanishing gradient issue and the network is known to be computationally expensive (23).

This thesis first proposes two successful momentum-based quantitative trading methods that either individual investors or large institutions can easily employ. The system deviates from the current literature by employ-

ing the Nonlinear autoregressive neural network with exogenous variables (NARX) instead of the field favorite LSTM. This choice is based on wanting to keep things simple. The memory gate mechanism employed by the LSTM is more difficult to explain than the autoregressive nature of the NARX. Similarly, the NARX is computationally more efficient and more suitable for individual investors with limited computational power. This system first denoises the American Index futures contracts via wavelet denoising techniques then feeds the denoised series into the NARX which then makes a series of one-day-ahead return predictions. Those predictions are used to make investment decisions for the next day with positive returns signaling a buy and negative returns giving a sell signal.

1.2 Intraday Trading Volume

Algorithmic trading involves using algorithms to systematically break larger orders into smaller orders to reduce transaction costs due to slippage. Often the two terms, Quantitative and Algorithmic trading, are used interchangeably but in this instance, this study refers to those defensive strategies. For example, percentage of volume (POV) is an algorithmic trading method that breaks down large orders based on a percentage of the volume traded. This strategy requires accurate volume predictions along with a user-defined participation ratio. Additionally, the volume-weighted average price (VWAP) algorithmic trading method attempts to use a market's volume profile to release smaller orders with the goal of executing as close to possible to the

volume-weighted average price. The VWAP is defined as:

$$\text{VWAP} = \frac{\sum \text{Price} \cdot \text{Volume}}{\sum \text{Volume}} \tag{1.2}$$

Accurate intraday volume predictions are critical to many aspects of the investment process, not just Algorithmic trading. Volume, the cumulative trading activity over a given period of time, is considered to be a technical indicator and can be used on its own to make an investment decision or it can be used to calculate other technical indicators. The daily seasonal patterns in intraday volume data are well documented. (59) highlights that high-frequency volume data may take an M-pattern, W-pattern, J-patter, U-pattern, or inverted U-pattern. In this data set, a majority of the securities displayed either the U pattern where wither the open or close had either slightly or drastically higher daily averages. The seasonal patterns were strongest in the non-transformed intraday volume data but it was still present in log trading volume as well.

The second project proposes an innovative method of predicting intraday high-frequency trading volume with the goal of improving volume-based algorithmic trading strategies. The method uses the novel CoFES S-NARX which augments the NARX feature set with the seasonal components from an ARMA(P,0)[s] model without adding any of the assumptions associated with the ARMA(P,Q)[s] model. Seasonality can be thought of as variations that occur at regular intervals of less than a year, like hourly or monthly. Seasonality is a well-covered topic in time series and there are several studies that suggest that seasonal analysis improves the prediction of time series

with strong seasonal patterns ((22),(50),(82)). This study uses minute level intraday data which results in a seasonality of 390, given there are 390 minutes in a trading day for U.S. equities. This can be verified in the PACF graphs of both Apple and Microsoft, where spikes are followed by exponential decay at intervals of 390, see Figure B.2 and Figure 3.1.

Volume is a major indicator of market activity; as a result, there are too many volume-based technical indicators to list them all. Volume technical indicators are of particular importance to technical analysis because volume can be used to confirm several hypotheses about the market. On Balance Volume (OBV), Volume Relative Strength Index (V.RSI), Money Flow Index (MFI), Chaikin Money Flow (CMF), Chaikin Accumulation / Distribution (chaikinAD), and Arms' Ease of Movement Value (EMC) are common volume technical indicators used by quants looking to take advantage of the signals in trading volume. Most of the time these technical indicators are used to determine some actionable insight into the state of the market but we want to see if they aid our models' ability to predict log volume for American technology stocks. Efficient market proponents would argue that historical price data is useless in predicting future trading volume levels (86). However, there are too many counterexamples that suggest that financial time series are in someways predictable [(41), (71)].

Chapter 2

Forecasting Daily Returns of American Index Future Contracts via Wavelets Denoising

2.1 Introduction

The ability to establish highly leveraged positions, ample liquidity, lower transaction cost, and volatility are just a few of the reasons why the U.S. Futures market is attractive to quantitative investors. In particular, U.S. index futures, futures contracts are written on a U.S. stock index like the S&P 500 or NASDAQ 100, are popular with quantitative investors because they are settled in cash which simplifies investment strategies by eliminating the risk of taking delivery of unwanted assets. Many would agree that the U.S. futures market is more efficient than the U.S. equity market because is it harder to trade in the futures market in a way that takes advantage of inside information. However, the U.S. futures market still exhibits momentum, a common market inefficiency [(55),(63)], and long memory in closing prices (78). As a result, momentum-based methods are popular quantitative trading strategies within the asset class (9), (53), (80). Many machine learning quantitative trading strategies incorporate momentum by including momentum technical indicators as predictors (10), (20), (83).

At the same time, changes in the market microstructure, the development of risk-factor exposure investment strategies, improvements in computing power, the increased availability of large data sets, and the initial success of pioneers are all partially responsible for machine learning techniques being used more frequently in quantitative trading (39). Individual investors have to employ more brainpower to remain competitive in an investing landscape that is starting to be dominated by machine learning techniques that often remove individual decision-making from the investment process. Like traditional stock market prediction, machine learning techniques can be broken down into either a fundamental analysis approach, using a company's fundamental information like earnings and dividends to attempt to predict stock prices, or a technical analysis approach, attempting to identify mispriced securities based on recurrent and predictable stock price patterns without considering a company's fundamentals (12).

Moreover, a major driver of this emergence of machine learning within financial time series analysis has been the success of machine learning techniques in predicting financial time series when compared to traditional methods like the ARIMA model and linear regression that rely on rigid assumptions like stationary or independence. Empirical studies suggest that a majority of real-world time series are not only non-stationary but are also nonlinear (76). Also, it has been well documented that a majority of financial time series deviate from normality via fat tails, which according to (81) is evidence of nonlinear dynamics. Additionally, as a result of high-frequency

trading, the amount of long-term dependencies within the American equity market times series, which is evidenced by Hurst exponent > 0.5, has increased steadily since 2005 (1). The Hurst exponent is used to measure the long memory in financial time series. Interestingly, by 2016 high frequency accounted for approximately 80% of foreign exchange futures volume and $66.\bar{6}\%$ of both interest rate futures and Treasury 10-year futures volumes (54). These facts suggest that methods that rely on the assumption of independence may be ignoring valuable information that U.S. futures market traders can use to make more accurate predictions. Deep Learning techniques, which allow practitioners to avoid all of the before mentioned assumptions, may be an appropriate alternative to forecasting U.S. index futures time series.

In particular, the W-NN, which first denoises a time series using a wavelet-based technique and then uses Artificial Neural networks to make a series of predictions, has been used to predict financial time series for over two decades. The W-NN model has been used with several different neural networks. For example, (84) used the W-NN model to predict an Exchange Traded Fund (ETF) based on the S&P 500 utilizing the Feed-forward Back Propagation Neural Network [FBNN]. However, the Long Short Term Memory (LSTM) neural network, designed to model both short-term and long-term nonlinear dependencies, became the dominating recurrent neural network in both financial time series prediction and within the W-NN when applied to financial time series. In fact, (71) highlights how LSTM networks

have dominated research on predicting financial time series using deep learning techniques between 2005-2019, being the network of choice in over 60% of the research. Similarly, a majority of the studies that used the W-NN model on financial data used the LSTM network as the NN (43), (67), and (89). Furthermore, the LSTM has also been a popular method to analyze the long memory patterns found in futures markets (32), (77),(87).

Recently, the Nonlinear Autoregressive Neural Network with Exogenous variables (NARX) has proven to be one of the best recurrent neural networks (RNN) for analyzing time series with persistent long memory patterns. Specifically, (23) found the NARX to be superior to the LSTM in terms of vanishing-gradient properties, improving performance with time series with long-term dependencies, and requiring fewer parameters and computation. Indeed the NARX has begun to appear in Financial Time Series forecasting research. For example, (16) successfully used the NARX to forecast Realized Volatility of the daily return of the Standard & Poor's index, whereas, (28) successfully employed the NARX to forecast the volatility of the crude oil price in Nigeria. Also, (38) successfully employed the NARX within the W-NN to build a trading strategy that predicted the price series of East Asian Futures.

Wavelet analysis has successfully been applied in several applications such as signal filtering and denoising, data compression, image processing, and pattern recognition (27). There are several reasons why wavelets are becoming more popular in financial time series analysis. Namely, wavelets

have the ability to sparsely represent a wide range of functions and are able to detect and represent localized features of a time series. Wavelets have the ability to analyze data in the time and frequency domain, overcoming the stationary limitation of the Fourier transform. Also, wavelets are more efficient than other methods in terms of computation speed and storage requirements (56).

Additionally, wavelet denoising applications have shown to be a successful technique to improve forecasts of financial time series. For instance,(31) used wavelet denoising, via the Haar wavelet along with a user-defined threshold, to improve prediction from a recurrent neural network (RNN) based on the artificial bee colony (ABC) algorithm (called ABC-RNN). This study forecast the Dow Jones Industrial Index, London FTSE-100 Index (FTSE), Tokyo Nikkei-225 Index (Nikkei), and Taiwan Stock Exchange Capitalization Weighted Stock Index (TAIEX). In addition, (88) used wavelet decomposition to improve the forecast from the LSTM on the Shanghai Stock index's next-day closing price between 2012-2017. Similarly, (44) used wavelet denoising to improve the forecast from LSTM that uses index data, volume, and technical indicators to predict the HSI, SSE, SZSE, TAIEX, NIKKEI, and KOSPI indexes between 2010-2016. Further, (21) used Wavelet denoising to improve the forecast from a Convolutional Neural Networks (CNN) on the price movements of EUR/USD between 1950–2016.

However, there has not been a consensus as to the most effective wavelet thresholding technique for improving the forecast of financial time series. A

recent study, (2) suggests that the three most popular wavelet thresholding techniques for financial time series are the universal hard, universal soft, and the empirical Bayesian thresholding technique. Further, (2) introduces two innovative wavelet thresholding techniques for analyzing financial time series, namely the Wave $L_2\mathbb{E}$ and the Wave $L_2\mathbb{E}_{\chi^2}$ which are based on the $L_2\mathbb{E}$ method, proposed by (70) and known for its robustness.

In this study, we explore how the novel Wave $L_2\mathbb{E}$ and the Wave $L_2\mathbb{E}_{\chi^2}$ thresholding techniques affect the forecasting ability of a NARX based W-NN and the profitability of our long/short and long-only momentum trading model based on the predicted daily returns for 4 U.S. index futures contracts. We compare the results of the two novel thresholding techniques against the 3 gold standard thresholding techniques for financial time series: the universal hard, universal soft, and empirical Bayesian. Also, trading profitability is compared against the traditional buy and hold strategy, which goes long the asset over the 10-year period, in the asset over the same time period and prediction success is measured with traditional classification metrics, accuracy, precision, and recall. Classification metrics are used because our trading algorithm will be based on the sign of the predicted return and we are interested in the W-NN ability to correctly predict the correct direction as opposed to the accuracy of the predicted return.

The remainder of this chapter is organized as follows: Section 2 describes the methods used in the future daily return prediction model used in this study. Section 3 presents the results and compares them to other wavelet

thresholding techniques. Section 4 summarizes the prediction and trading results from the 5 W-NN models on 4 American index future contracts.

2.2 Methods

2.2.1 RNN: NARX

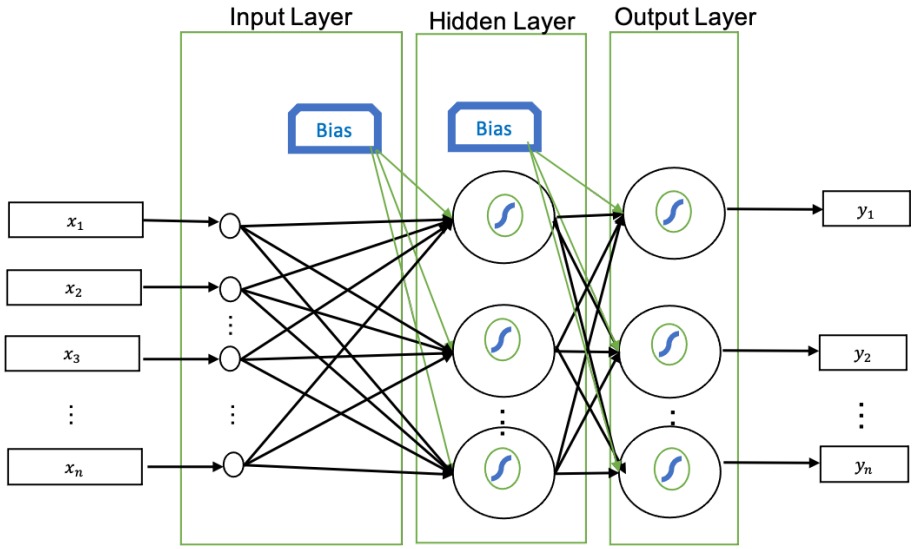


Figure 2.1 : Artificial Neural Network

Artificial Neural Networks (ANNs), powerful machine learning techniques that mimic learning mechanisms found in brains, have become popular within the quantitative trading community. ANN offers quantitative investors a set of powerful prediction tools and alternative methods to analyze the non-stationarity and non-linearity often found in financial time series. Similarly, ANN offers versatility which has allowed quantitative investors to

utilize ANNs in algorithmic trading, risk management, fraud detection, portfolio management, asset pricing, financial sentiment analysis, and behavioral finance (71). For example, (60) used a fusion approach involving ANN, Random Forest, and Support Vector Regression to predict Indian Stock market indices. Similarly, (90) used a hybrid ARIMA-ANN model to forecast the Canadian lynx and BP/USD exchange rate series. Additionally, (41) used a hybrid ARIMA-ANN-Fuzzy logic model to forecast US dollars/Iran Rials, Gold price, and Euro/Iran Rials. As well, (57) used Feed Forward Neural Networks to predict SPY ETF and 10 stocks from S&P 500.

ANNs include an *input layer*, one or more *hidden layers*, and an *output layer*. Adjacent layers are connected by a series of weights, which scale the input signal from the previous layer. Each layer has the potential to have multiple nodes and each node in the hidden and output layers has an activation function and a potential bias term which reduces variance and introduces flexibility to the neural network. Fitting an ANN involves using optimization methods to arrive at the optimal weights and bias for the system based on an appropriate loss function. ANN often uses backpropagation and gradient descent to systematically redistribute the error calculated in the output layer to the previous layers and to arrive at the optimal layer weights and bias values.

For example, Multilayer perceptrons (MLP) are a class of feed-forward ANNs that consist of multiple connected perceptrons. Perceptrons, also called McCulloch–Pitts neurons or linear threshold gates, are the oldest

and simplest form of neural networks. Perceptrons consist of a single input layer and the output node (25). MLPs have a minimum of three layers an input layer, an output layer, and one or more hidden layers. The non-input nodes typically employ nonlinear activation functions in an MLP. MLPs often employs the backpropagation algorithm to learn network weights and biases. MLPs can be used for either regression or classification problems depending on the function used in the output layer (25).

Recurrent Neural Networks (RNNs) are a class of ANNs that specialize in processing sequential data, which often have sequential dependencies. RNNs encode temporal information into feedback connections, which are capable of capturing the time-varying dynamics of the underlying system (11). Memory is introduced into RNN by adding a time window over the data via time delays. The unit time delay $H(z) = z^{-1}$, is the simplest form of memory, and Tapped Delay Lines (TDL), which consist of a series of unit time delays, are the simplest memory architecture. RNN's memory is one of the major distinctions between traditional ANN which assumes all inputs are independent. The learning process for RNN involves 'unfolding' the RNN, writing an equivalent acyclic graph representation, in order to have a closed-form for the network training. The transformed network can then be trained via traditional methods used on feed-forward neural networks. This process is called Backpropagation Through Time (BPTT). However, RNNs are known for having a short memory due to vanishing gradient issues that are introduced when attempting to learn longer memory relationships

with gradient-based learning algorithms. Different RNN models have been formulated, like the LSTM and Gated Recurrent Unit (GRU), to circumvent the short memory limitations in standard RNNs.

Notably, the Nonlinear autoregressive neural network with exogenous variables (NARX), first introduced in (46), is a class of RNN that was formulated to model long dependencies in sequential data. The NARX, which has a recurrent dynamic architecture with several hidden layers, was inspired by the nonlinear autoregressive with exogenous inputs, a discrete-time nonlinear model found in advanced non-linear time series applications (11). However, unlike traditional RNNs, the recurrence in the NARX model is given only by the feedback on the output, rather than from the entire internal state. Fortunately, (46) suggest that this design allows for the NARX to be implement via a MLP where the target y_t is regressed against d_y lagged values, $\{y_{(t-d_y)}, ..., y_{(t-1)}\}$, and d_x lagged values of an exogenous input signal $\{x_{(t-d_x)}, ..., x_{(t-1)}\}$.

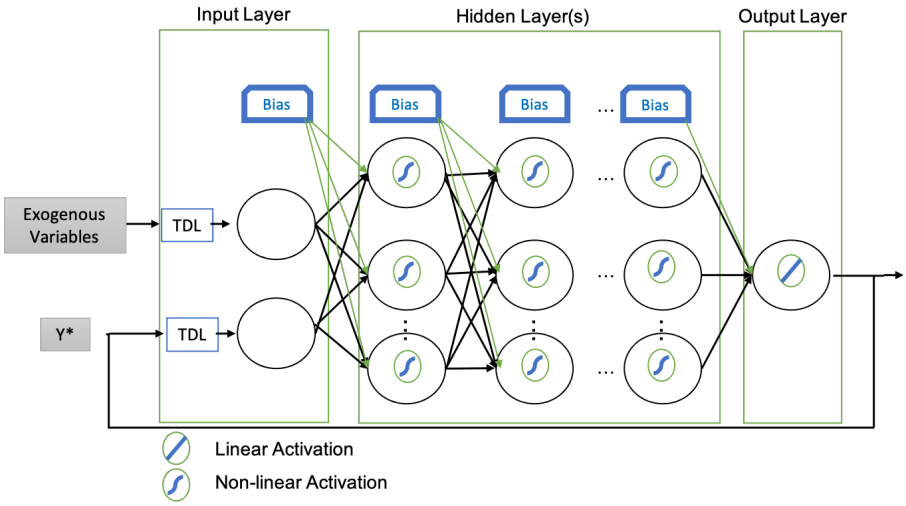
The NARX inputs, i_t, consist of two Tapped Delays Lines.

$$i_t = (x_{t-d_x}, ..., x_{t-1}, y_{t-d_y}, ..., y_{t-1}) \tag{2.1}$$

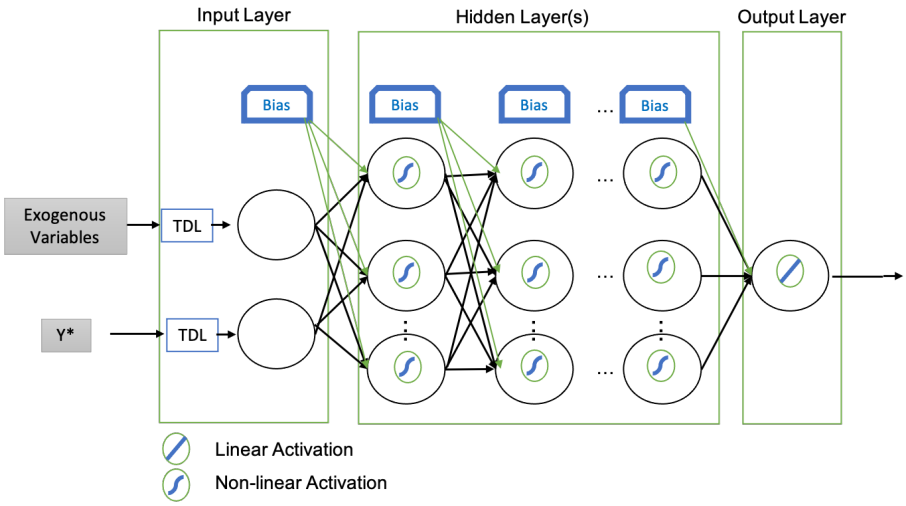
The NARX output equation is given by:

$$y_t = f(x_{t-d_x}, ..., x_{t-1}, x_t, y_{t-d_y}, ..., y_{t-l}, \Theta) \tag{2.2}$$

where $f(\cdot)$ is a nonlinear mapping function performed by a MLP, Θ are the parameters (weights & bias), d_x is the input delay, and d_y represent the



((a)) Operational mode



((b)) Training mode

Figure 2.2 : During training, the unit transfer is disconnected and y* is input directly into the network. After training, y* is removed and the output from the network y is connected to the input via a feedback loop.

output delay (11). The input to the NARX, i_t, has $d_x N_x + d_y N_y$ terms. Where N_x and N_y are the number of inputs and output variables.

The MLP used to model the NARX has a single input layer, $N_l \geq 1$ hidden nodes, and an output layer. The following equations dictate the output for the network.

$$h_1[t] = t(i_t, \theta_i) \tag{2.3}$$

$$h_l[t] = t(h_{l-1}[t-1], \theta_{hl}) \tag{2.4}$$

$$y[t] = l(h_{N_l}[t-1], \theta_0) \tag{2.5}$$

where $h_l[t] \in \mathbb{R}^{N_{h_l}}$ is the output of the l^{th} hidden later at time t, $l(\cdot)$ is the identity function, and $t(\cdot)$ is an activation function, which is either tahn or sigmoid for the NARX. The weights of the network are $\Theta = \{\theta_i, \theta_o, \theta_{h_1}, ..., \theta_{h_{N_l}}\}$, where $\theta_i = \{W_i^{h_1} \in \mathbb{R}^{d_x N_x + d_y N_y \times N_{h_1}} \ b_{h_1} \in \mathbb{R}^{N_{h_1}}\}$, $\theta_o = \{W^o_{h_{N_l}} \in \mathbb{R}^{N_l \times N_y} \ b_o \in \mathbb{R}^{N_y}\}$, and $\theta_{h_l} = \{W^{h_l}_{h_l - 1} \in \mathbb{R}^{N_{h_l - 1} \times N_{h_l}} \ b_{h_l} \in \mathbb{R}^{N_{h_l}}\}$.

During training the time series relative to the desired output y* is fed into the network along with the input time series x, see Figure 2.2. Then the output feedback, the recurrence that connects the output of the network with the input target node, is disconnected and the resulting network has a purely feed-forward architecture, which parameters can be trained using well-established backpropagation techniques(11). The NARX employs the

following loss function in the gradient descent.

$$L(y, y*; \Theta) = MSE(y, y*) + \lambda_2 ||\Theta||_2 \tag{2.6}$$

where MSE = Mean Square Error, λ_2 a hyperparameter that weights the importance of the L_2 regularization term in the loss function. The initial phase of λ_2 is transient, the first prediction of y is initially fed back into the network as input and disregarded. The NARX network has 5 hyperparameters: the input and output TDL, the number of hidden layers, the number of neurons in each layer, the regularization hyperparameter λ_2 in the loss function, and the learning rate (11).

2.2.2 Wavelet Denoising

Denoising is one of the most frequently used properties of wavelets analysis. Most wavelet denoising schemes are based on a thresholding technique applied to the wavelet coefficient that results from the wavelet transformation. A wavelet thresholding method can be distinguished by the choice of transformation, either discreet or continuous, the wavelet, which is strongly influenced by the transformation decision, and finally the threshold technique.

The traditional 3 step approach to wavelet denoising:

1. Apply either the **discrete wavelet transform (DWT)** or the **continuous wavelet transform (CWT)** to transform the original time series which results in a set of wavelet coefficients. Wavelet coeffi-

cients can be thought of as the information, or details contained in the original time series at different frequency components.

2. A threshold is applied to the coefficients which set all coefficients that are below the "threshold" equal to zero. The idea is to consider the wavelet coefficients that are smaller than a certain threshold to be noise. The thresholding process removes this noise from the original data set by setting small coefficients equal to 0.

3. The remaining coefficients are used to reconstruct the "denoised series" via an ***inverse wavelet transform***.

Universal Soft and Hard

The universal soft and universal hard thresholding techniques are two of the most popular discreet wavelet threshold techniques. The two thresholding techniques are based on methods of magnitude thresholding, which partitions the original signal into components and then reduces or removes certain wavelet coefficients in order to isolate a desired behavior from the signal. Unlike thresholding methods that depend on the scale, these two methods remove or reduce the smallest amplitudes coefficients without taking into consideration the scale. In the hard thresholding scheme, a threshold λ, is set and those coefficients below this threshold are considered to represent a random or noisy part of the signal. The hard threshold removes the entire coefficients if it falls below the threshold. In both cases, we used the

$$W_i^{hard} = \begin{cases} 0, & |W_i| < \lambda \\ W_i, & |W_i| \geq \lambda \end{cases} \tag{2.7}$$

The soft threshold on the other hand considered coefficients to be a mixture of signal and noise. In soft thresholding, all of the wavelet coefficients less than λ are set to zero and all of the coefficients above shrink towards zero by subtracting λ.

$$W_i^{Soft} = \begin{cases} 0, & |W_i| < \lambda \\ (W_i - \lambda), & W_i > \lambda \\ (\lambda - W_i), & W_i < -\lambda \end{cases} \tag{2.8}$$

In both of the formulations of the hard and soft we use one of the most popular thresholds for λ, the universal threshold $\lambda_u = (2 \cdot ln(N))^{\frac{1}{2}}\sigma$, where σ is the standard deviation of the noise within the signal and $(2 \cdot ln(N))^{\frac{1}{2}}$ is the expected maximum value of a white noise sequence of length N (Addison).

Empirical Bayes

The Empirical Bayes is another discreet wavelet transform-based thresholding technique where the threshold is estimated from the data. The wavelet coefficients, d_{jk}, are assumed to be independently distributed with prior $(1 - \pi_j)\delta(0) + \pi_j N(o, \tau_j^2)$, which is a mixture with an atom of probability at zero and a normal distribution with variance τ_j^2.

Suppose that Z=$(Z_1, ..., Z_n)$ are observations satisfying

$$Z_i = \mu_i + \epsilon_i \tag{2.9}$$

where $\epsilon_i \sim N(0, 1)$. μ_i is assumed to have an independent prior distribution given by

$$f_{prior}(\mu) = (1 - w)\delta_0 + w\gamma(\mu) \tag{2.10}$$

where γ, the nonzero part of the prior, is assumed to be a fixed unimodal symmetric density. Usually this density, γ , is normally distributed but can be replaced with distributions like double exponential. The marginal density of Z_i is assumed to have to form

$$(1 - w)\phi(z) + wg(z) \tag{2.11}$$

where g $= \gamma \star \phi$, $\star$ denotes convolution, and γ represents the standard normal density. The marginal maximum likelihood estimator $\hat{w}$ of w to be the maximizer of the marginal log-likelihood

$$l(w) = \sum_{i=1}^{n} log\{(1 - w)\phi(Z_i) + wg(Z_i)\} \tag{2.12}$$

This marginal log-likelihood has the following constraint on w, $t(w) \leq \sqrt{2 \cdot log(n)}$.

The basic approach to calculate the Bayesian threshold is to substitute $\hat{w}$ into the prior in order to estimate the μ_i. Suppose μ has prior described by equation 4 and we observe Z N(μ,1). If the posterior median is used then μ_i is estimated by $\hat{\mu}(Z_i; \hat{w})$ if the posterior mean is used then the estimate is

$\tilde{\mu}(Z_i; \hat{w})$. For fixed w<1, the function $\mu(z; w)$ will be monotonic function of z with threshold properties, in that there exist t(w)>0 such that $\hat{\mu}(z; w) = 0$ if and only if $|z| \leq t(w)$. The estimated weight $\hat{w}$ results in an estimated threshold $t(\hat{w}) = \hat{t}$. The posterior median threshold is defined to be the value of t(w) such

$$P(\mu > 0 | X = t(w)) = .5 \tag{2.13}$$

Under this formulation, t(w) is the largest observed value for which $\hat{\mu}(Z_i; \hat{w})$ will be zero (74).

Wave$\mathbf{L_2\mathbb{E}}$ & Wave$\mathbf{L_2\mathbb{E}_{\chi^2}}$

The Wave$L_2\mathbb{E}$ thresholding technique deviates from traditional wavelet thresholding techniques found in financial applications, which rely on the DWT, and instead utilize the CWT. Wave$L_2\mathbb{E}$'s choice of transformation allows this technique to employ the **Morlet Wavelet**, which is a popular wavelet in financial and economic applications. The Morlet wavelet, a complex wavelet, is characterized by the ability to provide information on both amplitude and phase (27). The Morlet Wavelet is also an analytic wavelet, which guarantees our ability to recover the original series x(t) with an inverse transform (2).

Wave$L_2\mathbb{E}$ thresholding is based on the $L_2\mathbb{E}$ method, known for its robustness, first proposed by (70) . The L_2E, a minimum distance estimator, is a substitute for traditional density estimators like histograms or kernel-based methods. In essence, (70) showed that for mixture distribution that the

L_2E does a good job at identifying outliers in large data sets. The Wave$L_2\mathbb{E}$ and Wave$L_2\mathbb{E}_{\chi^2}$ assume that outliers in the data are the true signal, with noise coefficients being close to zero in magnitude, and take advantage of the L_2E's ability to find groups of outliers in data sets. The steps to implement these two methods are as follows.

1. Apply the Morlet wavelet, a continuous wavelet transform (CWT), to the original time series and calculate the squared wavelet coefficients.

2. Apply the Wave$L_2\mathbb{E}$ and Wave$L_2\mathbb{E}_{\chi^2}$ at each time point solving for estimates of (σ_t and w_t) of the $L_2\mathbb{E}$ criterion.

$$L_2E(\omega_t, \sigma_t) = \frac{\omega_t^2}{(2\sigma_t\sqrt{\pi})^d} - \frac{2\omega_t}{n}\sum_{i=1}^{n}\phi(x_i|0, \sigma_t^2 I_d) \tag{2.14}$$

3. Threshold the wavelet coefficients based on the estimates where the λ_{WaveL_2E} = the largest of the smallest $\omega_t M$ wavelet coefficients, where M is the number of wavelet coefficients, pairs, or triples, depending upon the step chosen. The $\lambda_{WaveL_2E\chi^2}$= 95 critical value of the squared wavelet coefficient distribution. Both are hard thresholds where wavelet coefficients below the threshold are set to zero.

4. Calculate the inverse transform of the denoised time series or solve for the pure signal using the recovered estimates.

2.2.3 Research Methodology

Data

The sample data in this study was collected from Bloomberg L.P. and consists of historical open, high, low, and close (OHLC), volume, Hurst exponent, along with the 8 momentum-based technical indicators: Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), MACD Signal, Stochastic Fast %K, Stochastic Slow %K, Stochastic %D, and Ultimate Oscillator. Daily data were collected for S&P 400 mini continuous futures contracts, S&P 500 mini continuous futures contracts, NASDAQ mini continuous futures contracts, and Dow Jones mini continuous futures contracts between 1/1/2008 to 12/31/2020.

We looked at the Hurst exponent, introduced by (36), to measure the long-term memory in each of the index futures price series. The value of Hurst exponents ranges between 0 and 1 and effectively classifies a time series into 3 categories. If H=.05, the time series is categorized as random, $0 < H < .5$ indicates a persistent, mean-reverting, series, and $H > .5$ indicates a persistent series, which are trend reinforcing (64). The average Hurst exponent over this time period was .594 for the S&P 400 mini Futures Contract, .582 for the S&P 500 mini futures contract, .601 for the NASDAQ mini futures contract, and .598 for the Dow Jones mini futures contract over this time period. The distribution of the 4 futures contract's Hurst exponent is displayed in the Appendix. Missing data points are replaced

using the 5-day moving average.

Model Inputs

There are hundreds of momentum-based technical indicators that technical analysis can employ. Momentum-based technical indicators quantify the rate of change of price movement and are a leading indicator of a change in trend(18). Both (7) and (8) suggest that the Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), MACD Signal Line, MACD Histogram, Stochastic fast %K, Stochastic slow %K, Stochastic %D, and the Ultimate Oscillator are the most frequently used momentum technical indicators when forecasting financial time series. These momentum-based technical indicators are all derived from the future's trading volume, open, high, low, and close.

- MACD = 12 period average closing price - 26 period average closing price
- MACD Signal = 8 period average of MACD
- MACD Histogram = MACD - MACD Signal
- Stochastic Fast%K $= 100 \cdot \frac{C_t - \text{Lowest low over last 14 periods}}{\text{Highest high over last 14 periods} - \text{Lowest Low over last 14 periods}}$
- Stochastic Slow %K = 3 period average of Stochastic Fast %k
- Stochastic Slow %D = 3 period average of Stochastic Slow
- $RSI = 100 - \frac{100}{1 + \frac{\sum \text{Gains over past 14 periods}}{\sum \text{Losses over past 14 Periods}}}$

- Ultimate Oscillator $= 100 \cdot \frac{(A_7 \cdot 4)+(A_{14} \cdot 2)+(A_{28})}{4+2+1}$

where:

- Buying Pressure (BP) = close - Min(low, prior close)
- True Range (TR) = Max(high, prior close) - Min(low, prior close)
- $A_7 = \frac{\sum_{p=1}^{7} BP}{\sum_{p=1}^{7} TR}$
- $A_{14} = \frac{\sum_{p=1}^{14} BP}{\sum_{p=1}^{14} TR}$
- $A_{28} = \frac{\sum_{p=1}^{28} BP}{\sum_{p=1}^{28} TR}$

The Architecture of the Models

This study compares five different thresholding techniques which result in five different W-NN models. First, The model starts with 3 years of training data and then takes the 1-day return series of the close series, resulting in C_r. Next, each model denoises the training set's $OHLC_r$ series via their different wavelet thresholding techniques. Model 1 uses the universal hard threshold technique, model 2 uses the universal soft threshold technique, model 3 uses the empirical Bayesian threshold technique, model 4 uses the $Wave L_2\mathbb{E}$ threshold technique, and model 5 uses the $Wave L_2\mathbb{E}\chi^2$ thresholding technique. Next, the denoised OHL and the 8 technical indicators are grouped as the exogenous inputs, and the denoised C_r is used as the target input for the NARX neural network.

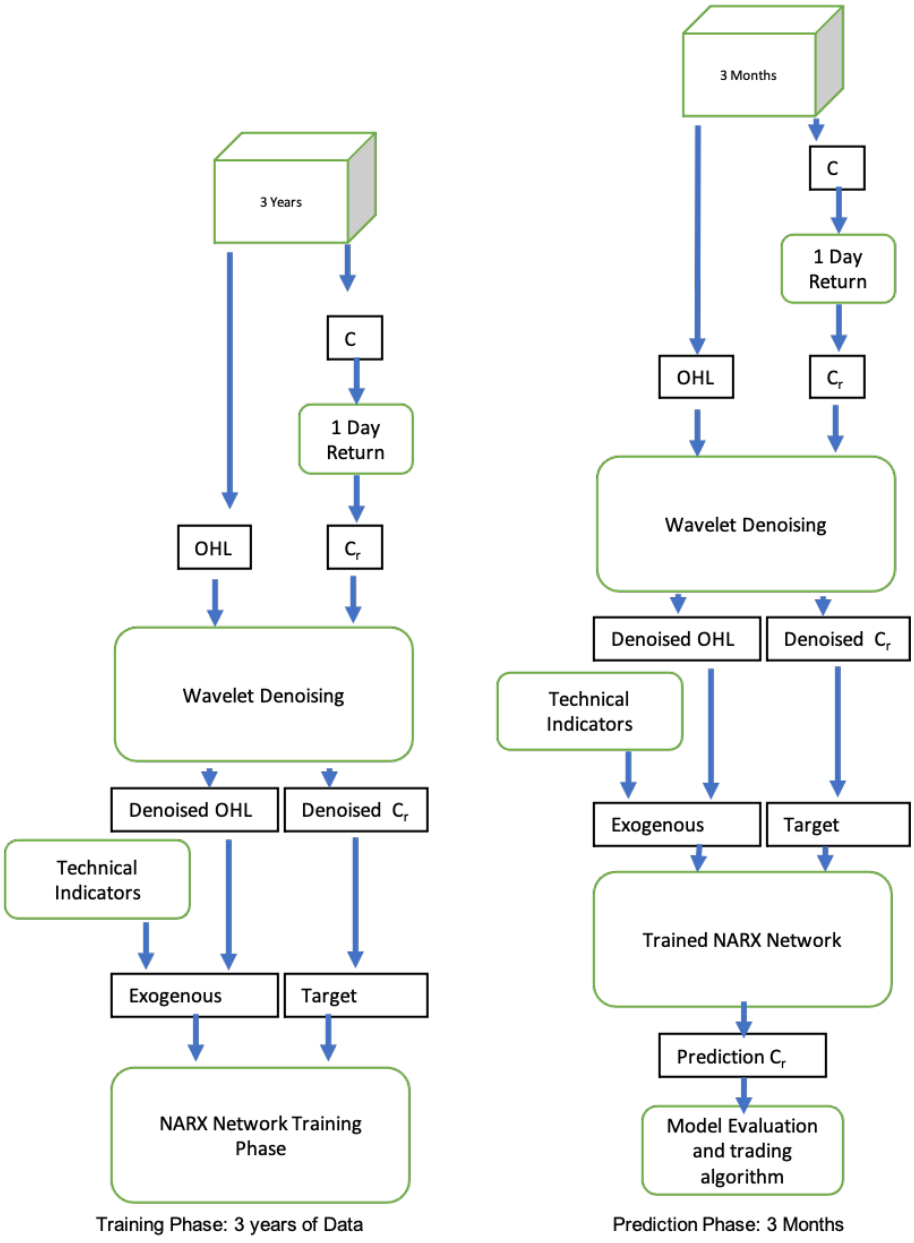


Figure 2.3 : Model architecture

Next, the model uses the trained network to make 3 months of 1 day ahead $\hat{C}_r$ predictions. The model takes the next 3 months of OHLC and calculates C_r. Each model next denoises 3 months of the $OHLC_r$ series via their different wavelet thresholding techniques. The denoising is done with a 3-month sliding window to avoid introducing forward-looking bias. The 3 months of denoised $OHLC_r$ and 8 technical indicators are used with the trained network to make 3 months of $\hat{C}_r$ predictions. At the end of the 3 month period, the window slides 3 months forward and the training process repeats. This process continues until the end of our 10-year backtesting window.

Last, the predictions are passed to the two trading algorithms where buy/sell signals are generated for the open and the position is held until close. The buy and sell signal for the first model, and the buy signal for the second are based on the sign of $\hat{C}_r$. Both algorithms long the open when the sign of the predicted 1-day return is positive and the long/short algorithm shorts the open when the sign of the predicted return is negative. Our model's architecture is represented in Figure 2.3.

Forecasting Performance

Accuracy, precision, and recall are used to compare the predictive accuracy of the resulting models. Accuracy is not a good measure if the classes are unequally distributed. Because each of the return series in this study has more positive 1-day returns than negative so we also include precision and

recall. The definition of accuracy is

$$Accuracy(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} 1_{\{\hat{y}_i = y_i\}} \tag{2.15}$$

where y_t is the target value and $\hat{y}$ is the predicted value. A higher accuracy value indicates better performance for the network. The definition of precision is

$$Precision = \frac{TP}{TP + FP} \tag{2.16}$$

where TP (true positives) is the number of points correctly assigned as positive and FP (false positive) is the number of points that were negative but classified as positive. The definition of recall is

$$Recall = \frac{TP}{TP + FN} \tag{2.17}$$

where FN (false negative) is the number of points that were positive but labeled negative. Precision attempts to quantify the proportion of positives classifications that were labeled correctly. Recall attempts to quantify the proportion of actual positives that were correctly identified.

Algorithm Profitability

This study compares the 10-year average annual return of the trading algorithm versus a basic buy and holds strategy over the same time period. The two trading algorithms are based on the sign of the predicted future price 1-day return. If the predicted 1-day return is positive then the long/short and long-only algorithm produces a buy signal and if the predicted 1-day

return is negative then only the long/short trading algorithm produces a sell signal.

$$\hat{C}_r > 0 : \ buy$$

$$\hat{C}_r < 0 : \ sell$$

where $\hat{C}_r$ is the predicted value for the next day's return. The cumulative profit or loss is calculated on a yearly basis. The cumulative returns of this trading strategy are calculated by multiplying the 1 plus returns from all of the buy and sell signals 1-day returns and then subtracting 1 from the product.

$$Return = (\prod_{t:r_t^l \in R^L} (1 + r_t^l) * \prod_{t:r_t^s \in R^S} (1 + r_t^s)) - 1 \tag{2.18}$$

where r_t^l is the 1-day return from a long signal on day t, R^L is the set of all long returns days, r_t^s is the 1-day return from a short signal on day t, and R^S is the set of all short returns.

2.3 Results

The study measured the forecast ability using accuracy, precision, and recall and measure profitability via the average 10-year annual returns of the two trading algorithms. The results are compared from the five different wavelet thresholding techniques: the novel WaveL$_2E$ and WaveL$_2E_{\chi^2}$, the universal hard, universal soft, and empirical Bayesian. The long/short algorithm allows both long and short positions and the long-only algorithm ignores sale signals and only employs long positions. Each model's NARX consisted of

1 hidden layer with 5 nodes, 6 delays, and was trained on 3 years of data. Performance is based on the highest accuracy, precision, recall, and highest 10-year average annual returns.

2.3.1 S&P 400 mini futures contracts

Table 2.1 : S&P 400 mini futures contract 10-year average accuracy, precision, and recall

Model	**Accuracy**	**Precision**	**Recall**
Wave$L_2\mathbb{E}$	0.535	0.548	0.877
Wave$L_2\mathbb{E}_{\chi^2}$	0.533	0.545	0.912*
Universal Hard	0.516	0.538	0.824
Universal Soft	0.543*	0.551*	0.904
EBayes	0.517	0.540	0.816

The S&P 400 mini futures contracts results show that all of the thresholding techniques had a 10-year average accuracy rate above .5. From the results presented in Table 2.1, we see that the universal soft threshold outperforms the other models with the highest 10-year average accuracy, .543, where a higher accuracy rate indicates a more accurate forecasting ability. However, the WaveL_2E and Wave$L_2E_{\chi^2}$ are not far away with a 10-year average accuracy of .535 and .533 respectively. The Wave$L_2E_{\chi^2}$has the second-highest 10-year recall average, .548, which is behind the universal softś 10-year recall

average of .551. The WaveL_2E has the highest average 10-year precision, .912, which is followed by the universal soft and the WaveL_2E, with 10 years average precision of .904 and .877 respectively.

Table 2.2 : S&P 400 futures contract yearly returns long/short

Model	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**CAGR**
Wave$L_2\mathbb{E}$	6.57	15.00	37.88	0.93	−0.23	−1.14	6.97	−9.33	19.48	49.47*	**12.56**
Wave$L_2\mathbb{E}_{\chi^2}$	−23.44	9.79	18.96	6.89	2.44	20.28	12.54	−14.04	20.47	47.70	**10.16**
Universal Hard	−15.85	29.75*	38.66*	6.53	−4.41	22.65*	11.80	−21.08	−0.35	−15.03	**5.27**
Universal Soft	6.27	16.68	21.85	1.60	25.65*	18.11	18.78*	−3.53	27.16*	15.75	**14.83***
EBayes	7.01*	6.49	15.42	20.87*	15.29	6.47	0.34	−0.1 *	19.37	−36.6	**5.46**
Buy& Hold	−3.09	16.05	31.56	8.15	−3.80	19.06	14.66	−12.63	24.22	11.56	**10.57**

For the long/short strategy, the universal soft threshold had the highest 10 year average annual return, 14.83. The WaveL_2E, with a 10 year average annual return of 12.56, was the only other score to also beat buy and hold over the same period.

The universal soft threshold also leads the long only strategy with a 10 year average annual return of 12.7 with the both WaveL_2E and the Wave$L_2E_{\chi^2}$10 year annual return, of 11.62 and 10.58 respectively, being the only other thresholding techniques that beat buy and hold over the same 10 year period.

Table 2.3 : S&P 400 futures contract yearly returns long only

Model	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	CAGR
WaveL$_2\mathbb{E}$	2.48	15.69	34.84	4.63	−1.93	8.67	10.79	−10.92	21.97	29.95	11.62
WaveL$_2\mathbb{E}_{\chi^2}$	−13.66	13.01	25.13	7.62	−0.65	19.73	13.70	−13.15	22.53	31.56*	10.58
Universal Hard	−9.41	23.00*	35.50*	7.41	−3.81	21.21*	13.37	−16.86	11.45	−1.00	8.09
Universal Soft	1.56	16.40	26.84	4.90	10.46*	18.63	16.74*	−8.14	25.86*	13.70	12.70*
EBayes	3.27*	11.56	23.42	14.45*	5.50	12.94	7.33	−6.44 *	22.03	−15.23	7.88
Buy & Hold	−3.09	16.05	31.56	8.15	−3.80	19.06	14.66	−12.63	24.22	11.56	10.57

2.3.2 S&P 500 mini futures contracts

Table 2.4 : S&P 500 mini futures contract 10 year average accuracy, precision, and recall

Model	Accuracy	Precision	Recall
WaveL$_2\mathbb{E}$	0.538	0.556	0.879
WaveL$_2\mathbb{E}_{\chi^2}$	0.535	0.552	0.916*
Universal Hard	0.519	0.548	0.828
Universal Soft	0.542*	0.557*	0.909
EBayes	0.524	0.55*	0.815

Similar to the S&P 400 results, the S&P 500 mini futures also resulted in 10 year accuracy rates above .5. The universal soft thresholding technique had the highest accuracy, .542, followed by WaveL$_2E$ and WaveL$_2E_{\chi^2}$ with

Table 2.5 : S&P 500 mini yearly returns long/short

Model	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**CAGR**
$\text{WaveL}_2\mathbb{E}$	11.76	15.40	30.74	8.31	−0.98	−5.79	11.58	−4.21	18.23	40.76*	12.58
$\text{WaveL}_2\mathbb{E}_{\chi^2}$	−21.06	6.93	19.70	10.42	3.63	10.33	14.95	−11.13	15.28	36.67	8.57
Universal Hard	−10.23	21.48*	33.72*	11.55	0.78	15.53*	12.30	−16.23	1.54	12.29	8.27
Universal Soft	8.10	12.09	24.77	9.00	26.73*	9.04	21.90*	3.46	28.46	16.36	15.99*
EBayes	12.59*	9.87	16.44	23.55*	10.49	8.34	10.69	9.27*	19.09	−6.95	11.34
Buy & Hold	−0.04	13.37	29.65	11.49	−0.83	9.86	19.66	−6.38	28.97*	16.02	12.18

.538 and .535 respectively. The universal soft has the highest 10 year average precision, .557, followed by WaveL_2E and $\text{WaveL}_2E_{\chi^2}$, .556 and .552 respectively.

Table 2.6 : S&P 500 mini yearly returns long only

Model	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**CAGR**
$\text{WaveL}_2\mathbb{E}$	6.25	14.48	30.29	9.99	−0.79	1.87	15.57	−5.23	23.59	28.21*	12.42
$\text{WaveL}_2\mathbb{E}_{\chi^2}$	−11.00	10.20	24.60	11.06	1.46	10.13	17.33	−8.54	22.07	27.51	10.48
Universal Hard	−5.05	17.56*	31.96*	11.58	0.28	12.89*	15.99	−11.35	14.55	15.29	10.37
Universal Soft	4.01	12.76	27.30	10.28	12.61*	9.47	20.80*	−1.53	28.85	16.28	14.08*
EBayes	7.08*	11.93	23.00	17.45*	4.84	9.32	15.12	1.29*	24.13	4.37	11.85
Buy& Hold	−0.04	13.37	29.65	11.49	−0.83	9.86	19.66	−6.38	28.97*	16.02	12.18

For the long/short strategy, the universal soft resulted in the highest 10 year annual average return, 15.99. Again, the result from the WaveL_2E, 12.58, is the only other strategy to beat buy and hold. For the long only

strategy, the universal soft also resulted in the highest 10 year annual average return, 14.08. The WaveL_2E thresholding technique had the only other 10 year annual average return, 12.42, which is higher than buy and hold over the same time period.

2.3.3 Dow Jones mini futures contracts

Table 2.7 : Dow Jones mini futures contract 10-year average accuracy, precision, and recall

Model	**Accuracy**	**Precision**	**Recall**
Wave$L_2\mathbb{E}$	0.541	0.554	0.877
Wave$L_2\mathbb{E}_{\chi^2}$	0.537	0.549	0.914*
Universal Hard	0.529	0.550	0.821
Universal Soft	0.547*	0.556*	0.906
EBayes	0.536	0.556*	0.809

The Dow Jones mini future contract also resulted in 10 year average accuracy rates above .5 for all thresholding types. The Dow futures 10 year average accuracy results were also lead by the universal soft thresholding technique, .547, closely followed by the WaveL_2E and Wave$L_2E_{\chi^2}$, .541 and .537 respectively. The universal soft lead the long/short strategy with a 12.61 ten year average annual return. Both the WaveL_2E and the empirical Bayesian thresholding technique 10 year average return, 10.95 and 10.89

Table 2.8 : Dow Jones mini futures contract yearly returns long/short

Model	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**CAGR**
WaveL$_2\mathbb{E}$	11.23	8.19	26.01	5.72	−2.11	7.43	19.06	−5.08	16.69	21.75*	10.89
WaveL$_2\mathbb{E}_{\chi^2}$	−6.54	5.67	22.85	7.65	−0.03	12.96	20.77	−7.14	18.22	18.22	9.26
Universal Hard	−0.84	10.74*	27.95*	7.12	0.91	15.73*	22.60	−10.19	11.99	10.17	9.62
Universal Soft	8.87	7.64	25.53	7.77	11.41*	11.48	26.26*	−1.87	21.91	7.10	12.61*
EBayes	12.83*	8.69	21.88	14.34*	3.86	10.81	22.32	0.30*	19.10	−4.64	10.95
Buy & Hold	5.53	7.22	26.61	7.63	−2.32	13.72	25.43	−5.93	22.52*	6.98	10.74

respectively, beat buy and hold over the same time period. The results are similar for the long only strategy with the universal soft leading with an 14.5 ten year average annual return followed by the WaveL$_2E$ and the empirical Bayesian, 11.03 and 11.01 respectively.

Table 2.9 : Dow Jones mini futures contract yearly returns long only

Model	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**CAGR**
WaveL$_2\mathbb{E}$	16.36	9.01	25.25	3.67	−2.08	1.25	12.96	−4.36	10.90	37.30*	**11.03**
WaveL$_2\mathbb{E}_{\chi^2}$	−17.45	4.00	19.14	7.48	2.12	12.16	16.19	−8.89	13.85	27.20	**7.58**
Universal Hard	−7.21	14.04*	28.84*	6.49	3.62	17.33*	19.72	−14.44	2.19	10.98	**8.15**
Universal Soft	12.20	8.03	24.26	7.82	26.06*	9.24	27.05*	2.24	21.05	7.10	**14.50***
EBayes	18.89*	9.66	17.11	21.32*	10.08	7.58	19.24	6.60*	15.43	−15.77	**11.01**
Buy & Hold	5.53	7.22	26.61	7.63	−2.32	13.72	25.43	−5.93	22.52*	6.98	**10.74**

Table 2.10 : NASDAQ mini futures contract 10 year average accuracy, precision, and recall

Model	**Accuracy**	**Precision**	**Recall**
WaveL$_2\mathbb{E}$	0.540	0.562	0.894
WaveL$_2\mathbb{E}_{\chi^2}$	0.545	0.560	0.949*
Universal Hard	0.541	0.566	0.857
Universal Soft	0.559*	0.567*	0.947
EBayes	0.533	0.565	0.825

2.3.4 NASDAQ mini futures contracts

The NASDAQ mini futures had the worst performance in terms of the trading algorithms. None of the models resulted in average 10 year annual returns that that were higher than buy and hold over the same period, which was 20.29. Both the long/short and the long only strategy were lead again by the Universal Soft thresholding technique with 10 year annual average returns of 19.26 and 19.40 respectively. The WaveL$_2E$ had the second highest 10 year average annual return for both the long/short and long-only strategies, 12.43 and 15.98 respectively. However, all of the average 10 year accuracy values are above .5, which is similar to the previous results. Universal soft had the highest accuracy rate, .559, but the WaveL$_2E$ and Universal Hard results were not far behind, with accuracy rates of .545 and .541

respectively. The Universal soft, universal hard, and empirical Bayesian threshold had the top three precision scores, .567, .566, .567 respectively. The Wave$L_2E_{\chi^2}$ had the highest recall score, .949, followed closely by the Universal Soft and the WaveL_2E, with a precision score of .947 and .894 respectively.

Table 2.11 : NASDAQ mini futures contract yearly returns long/short

Model	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**CAGR**
Wave$L_2\mathbb{E}$	8.57*	24.83*	14.38	−3.16	3.17	5.84	24.94	7.86*	36.03	1.84	**12.43**
Wave$L_2\mathbb{E}_{\chi^2}$	−0.08	10.96	23.66	13.74	9.37	1.39	16.36	0.93	20.60	−3.52	**9.34**
Universal Hard	−1.10	15.87	20.50	7.81	6.31	4.91	6.75	−7.84	30.59	38.81	**12.26**
Universal Soft	6.43	19.05	21.73	14.92	21.90*	5.89	18.88	1.66	35.06	47.09*	**19.26**
EBayes	−31.67	10.93	7.79	5.37	16.64	0.74	17.98	6.48	51.02*	28.79	**11.41**
Buy & Hold	2.64	16.74	34.97*	18.11*	8.39	6.02*	31.76*	−1.18	38.20	47.22*	**20.29***

Table 2.12 : NASDAQ mini futures contract yearly returns long only

Model	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	**CAGR**
Wave$L_2\mathbb{E}$	6.98*	20.77*	22.56	7.22	6.39	6.49*	26.53	4.64*	34.90	23.36	**15.98**
Wave$L_2\mathbb{E}_{\chi^2}$	2.66	13.83	27.20	15.67	9.50	4.26	22.23	1.17	27.18	20.68	**14.44**
Universal Hard	2.15	16.29	25.62	12.71	7.96	6.02	17.43	−3.22	32.18	41.85	**15.90**
Universal Soft	5.91	17.88	26.24	16.26	15.76	6.51	23.50	1.54	34.41	45.99	**19.40**
EBayes	−13.10	13.82	19.27	11.48	13.13*	3.94	23.05	3.95	42.39*	36.84	**15.47**
PBuy & Hold	2.64	16.74	34.97*	18.11*	8.39	6.02	31.76*	−1.18	38.20	47.22*	**20.29***

2.4 Conclusion

Innovations in machine learning and statistics have further facilitated the mass adoption of the ML techniques in trading. With all of the options for methods available it is often difficult to decide which method works best in different scenarios. This study investigated how the choice of a wavelet thresholding technique affected the performance of a well established prediction model, the Wavelet Neural Network, when predicting non-stationary, financial return series. This research also investigated the effect on profitability when the results from the prediction model is used in a trading strategy. The strategy first denoised the futures closing price's return series along with the original OHL series. The denoised OHL series were then combined the 8 momentum focused technical indicators as the inputs to the NARX network with the denoised close return series as the target. The NARX is trained on 3 years of data and makes 3 months of one day ahead return predictions. The sliding window is then moved 3 months forward and the process is repeated until the end data stream. The predictions are fed into a long/short and a long only trading algorithm that either take long/short positions on the open based on the sign of the predicted return and then closes out the position at the end of the trading day. The long only strategy ignores short signals and just take long positions similar to several mutual funds, which have more trading restrictions than hedge funds.

The success of the trading system and the accuracy of the prediction model, W-NN, is compared using the WaveL_2E , Wave$L_2E_{\chi^2}$, universal

soft, universal hard, or empirical Bayesian thresholding technique on S&P 400 mini future contracts, S&P 500 mini future contracts, Dow Jones mini futures contracts, and NASDAQ mini future contracts from (2011-2020). The results of the two trading algorithms, the long/short and the long-only strategy, are also compared against buy and hold over the same time period. Although the universal soft consistently had the highest 10-year average annual return for both the long/short and long-only strategies, the WaveL_2E threshold technique was also able to outperform buy and hold over the same time period for both trading strategies with the S&P 500 mini futures, S&P 400 mini futures, and the Dow Jones mini futures contracts. Both the novel WaveL_2E and the Wave$L_2E_{\chi^2}$ results were also very close to the leader's results in accuracy and precision while the Wave$L_2E_{\chi^2}$ outperformed in recall for all of the futures contracts. On average the long-only strategy had higher returns than the long/short strategy which is because the fact all strategies had higher average true positive rates when than true negative rates. The results seem consistent with the prediction results from previous studies (91) and (24) also attempt to use classification machine learning methods on daily returns.

SUPPLEMENTARY MATERIAL

Forecasting Code The code used to fit the NARX Model and make predictions from the trained networks (.R file)

Back Test Code The code that implements the trading algorithm and

checks the accuracy, precision, recall, and profitability. (.R file)

S&P 500 Raw Data The open, high, low, close, volume, and 8 technical indicators for S&P 500 mini futures contract from (2005-2021). (.cvs file)

S&P 400 Raw Data The open, high, low, close, volume, and 8 technical indicators for S&P 400 mini futures contract from (2005-to 2021). (.cvs file)

Dow Jones Raw Data The open, high, low, close, volume, and 8 technical indicators for Dow Jones mini futures contract from (2005-2021). (.cvs file)

NASDAQ Raw Data The open, high, low, close, volume, and 8 technical indicators for NASDAQ mini futures contract from (2005-2021). (.cvs file)

Chapter 3

Forecasting intraday volume via CoFES S-NARX

3.1 Introduction

Given the importance of trading volume to market analysis, there is no surprise that there have been several studies focused on predicting intraday volume. Intraday volume data is known to be heteroskedastic, non-normally distributed, and to exhibit strong autocorrelation and diurnal seasonality. These statistical characteristics vary based on several other features, like the type of financial asset and geographic region, introducing more complexity to the task of modeling intraday volume. A large portion of the literature on intraday volume forecasting falls into either traditional time series approaches, Bayesian approaches or machine learning approaches. Other noteworthy approaches include those (68) that used a functional analysis approach to develop a Time-Varying Coefficient Model to predict the intraday volume of Forex Futures traded on the Chicago Mercantile Exchange. Another study of note uses the model employed in FactSet's Trading Solutions, (19) which incorporates events when predicting intraday Volume via a coarse-grained Hawkes model. There were also two spline approaches to

forecasting intraday time series, (79) using discount weighted regression and discount weighted splines to model intraday time series that contain both intraweek and intraday seasonal cycles. Further, (37) implemented a dynamic cubic spline model, Spline-DCS, to forecast high-frequency trade volume for equities and foreign currency exchange rates.

A bulk of the literature on predicting intraday volume originates from time series methods, which is logical given that intraday volume data sets are in fact time series. For example, (17) used a state space approach to predict intraday trading volume making use of the Kalman Filter for likelihood evaluation coupled with the EM algorithm for parameter estimation. For extensions to multiple state-space models see (49), and (17). In the latter paper, cross-validation is used to determine the best number of states. Also, there were traditional basic time series models used to predict intraday volume, such as (58) who used moving average, exponentially weighted moving average, and ensemble approaches that combined classic time series models. For example, (69) and (52) propose 4 and 5-component autoregressive-moving-average (ARMA) models, respectively, to model intraday volume. The most effective time-series approach appears to be the Component Multiplicative Error Model, a form of the GARCH Model, which was first introduced in (26). The component GARCH model is extended in (14) by introducing a new loss function for the evaluation of proportion forecasts. A paper within the machine learning literature, (45), on the subject of intraday volume prediction, suggests that the Component Multiplicative Error model is the most

effective model and uses this model as its benchmark.

There were also Bayesian time series approaches used to predict intraday volume but this literature is not as voluminous as the traditional time series literature. Updating the end-of-day volume forecast over the course of the day as new data/information arrives fits nicely with the Bayesian framework. Bayesian inference to model the intraday traded volume of Polish stocks was the focus in (33) which the linear Autoregressive conditional volume (ACV) model was combined with a Weibull distribution for the error term for intraday volume. This work was extended, (34), to include an exponential and generalized gamma distribution for the error term of the intraday volume. Finally in the Bayesian time series literature, (15) bring forward the mixed autoregressive conditional duration (Mixed ACD) model, to model daily duration time series, where duration is defined as the waiting time between market events.

Similarly, numerous machine learning approaches have been employed to forecast intraday volume. The backpropagation neural network is the machine learning method of choice for (40) where they forecast monthly futures trading volume at the Winnipeg Commodity Exchange (WCE). The dynamic support vector machine (DSVM) is presented in (48) with a focus on forecasting intraday volume percentages of gold futures and S&P 500 futures. Combining machine learning and time series, (45) used the LSTM network in conjunction with SVM and AR models to forecast the log volume of the S&P 500 ETF (SPY). Similarly, two regression methods, namely

support vector regression (SVR) and partial least squares (PLS) were used to forecast the daily volume of Bovespa traded stocks using high-frequency predictors (5).

Recently, the Nonlinear Autoregressive Neural Network with Exogenous variables (NARX) has proven to be a suitable alternative to the LSTM for analyzing financial time series. Specifically, (23) found the NARX to be superior to the LSTM in terms of vanishing-gradient properties, improving performance with time series with long-term dependencies, and requiring fewer parameters and less computation. Indeed the NARX has begun to appear in financial time series forecasting research. For example, (16) successfully used the NARX to forecast Realized Volatility of the daily return of the Standard & Poor's index, whereas, (28) successfully employed the NARX to forecast the volatility of the crude oil price in Nigeria. Also, (38) successfully employed the NARX within the W-NN to build a trading strategy that predicted the price series of East Asian Futures.

With the success of the NARX model, there is no surprise that there have been several versions of this RNN formulated to handle different scenarios. For example, (13) formulates the Neural NARXs (NNARXs), which uses Feed-Forward Neural Networks (FFNNs) as regression functions for the Nonlinear Autoregressive exogenous (NARX) model. Similarly, (65) proposed a NARX based dual-stage attention-based recurrent neural network (DA-RNN) where in the first stage, an input attention mechanism is introduced to adaptively extract relevant driving series (a.k.a., input features)

at each time step by referring to the previous encoder hidden state and in the second stage, a temporal attention mechanism is employed to select relevant encoder hidden states across all time steps. Also, (6) formulated an ELMAN-NARX hybrid to analyze and predict chaotic time series. However, there hasn't been any consensus on which NARX model to use to deal with non-linear time series with strong seasonality. For example, (29) formulated a "seasonal-NARX" where separate NARX models are fit for each of the four seasons of the year and compared against a model over the entire year. However, this method is limited by seasonal patterns that do not fit in one of the four seasons of the year. Four different NARX models are already computationally expensive when compared to traditional seasonal models. In this formulation, the number of required models grows linearly with the number of seasons modeled. So, for monthly seasonality, this formulation would require 12 different NARX models and approximately 390 separate NARX models for a minute-by-minute diurnal seasonal pattern.

Similarly, (85) introduced a slightly more sophisticated model with the SARIMA-NARX to predict Scarlet fever incidence using data between January 2004 and July 2018. The model first uses a SARIMA model to analyze/predict the seasonal patterns within a time series and then uses the standard errors from this model, which in theory contains the non-linear portion of the signal, as inputs for a NARX model to make a prediction on nonlinear information found in scarlet fever data. The two predictions, from the SARIMA and the NARX, are then combined to give the overall

SARIMA-NARX prediction. This method assumes that the seasonal pattern is linear and assumes there will be little to no seasonal information remaining within the residual errors for the NARX model to analyze which indicated that this model may fail to analyze any non-linear seasonal information.

Likewise, (51), maybe the most sophisticated method to date, uses the inputs from the novel Seasonal Component Autoregressive (SCAR) model as inputs for the NARX models to create a seasonal version (SCARX) to make a day-ahead electricity price forecast. First, the algorithm decomposes the original log-price series and the exogenous variables series into a long-term seasonal component and a stochastic component with short-term periodicities. Then, computes persistent forecast of long-term seasonal components independently for each of the 24 of the next day. They calibrate the NARX model and compute the forecast of the stochastic component with short-term periodicities for the 24 hours of the next day. Further, they combine the persistent forecast and NARX forecast. Finally, they take the exponents of the log-price forecasts from step 3 in order to convert them into price forecasts. (51)

In this paper, we introduce an ARMA(P,0)[s] inspired seasonal formulation of the NARX, the CoFES S-NARX, and use it to forecast the high frequency, minute by minute, log(trading volume) of American technology stocks found on the NASDAQ 100. This study focuses on the model's ability to forecast the last 15 minutes of the trading day given the heightened importance this period plays in volume-based algorithmic trading strategies.

This study shows that the model captures the diurnal periodic information, the autocorrelation, and the nonlinearity of the high-frequency intraday volume data of American technology stocks on the NASDAQ 100. This study adds volume technical indicators as exogenous variables that improve the model's ability to predict the different transformations of volume. Many machine learning quantitative strategies incorporate technical analysis by including technical indicators in the feature set (10), (20), (83).

Volume is a major indicator of market activity, as a result, there are too many volume-based technical indicators to list them all. Volume technical indicators are of particular importance to technical analysis because volume can be used to confirm several hypotheses about the market. On Balance Volume (OBV), Volume Relative Strength Index (V.RSI), Money Flow Index (MFI), Chaikin Money Flow (CMF), Chaikin Accumulation / Distribution (chaikinAD), Arms' Ease of Movement Value (EMC), and volume-based Simple Moving Averages (SMAs) are common volume technical indicators used by quants looking to take advantage of the signals in trading volume. Most of the time these technical indicators are used to determine some actionable insight into the state of the market but we want to see if they aid our models' ability to predict log volume for American technology stocks. Efficient market proponents would argue that historical price data is useless in predicting future trading volume levels (86). However, there are too many counterexamples that suggest that financial time series are somewhat predictable [(41), (71)].

The remainder of this paper is organized as follows: Section 2 gives an initial examination of the volume and log(volume) data sets. Section 3 describes the methods used in the high-frequency intraday trading volume prediction model used in this study. Section 4 presents the results and compares them to other machine learning-based prediction techniques. Section 5 summarizes the prediction results on select securities from the NASDAQ 100.

3.2 Data Analysis

First, we present some of the empirical properties of the high-frequency intraday volume and log volume data used in this study. We choose to study Apple and Microsoft, which are American technology stocks found in the QQQ, a NASDAQ 100 ETF, to ensure a healthy trading volume. The data sets consisted of regularly spaced, minute level, intra-daily open, high, low, close, and volume (OHLCV) data points for Apple and Microsoft stocks between 9/1/2020 to 9/1/2021. The data set contained two half days around the Thanksgiving and Christmas holidays which were included in the analysis.

The daily seasonal patterns in intraday volume data are well documented. For example, (59) highlights that high-frequency volume data may take an M-pattern, W-pattern, J-patter, U-pattern, or inverted U-pattern. To investigate the intraday patterns in our data sets, the average and standard deviation is calculated for hourly trading volume and log trading volume for

		H1	H2	H3	H4	H5	H6	H7
Volume	AAPL	462,328	240,864	173,440	144,438	131,591	141,838	232,661
		(481,953)	(124,662)	(106,841)	(98,445)	(104,588)	(102,358)	(210,886)
	MSFT	96,205	50,952	37,071	30,268	27,272	29,347	60,876
		(179,655)	(33,694)	(26,011)	(23,100)	(22,922)	(20,640)	(73,892)
Log Vol	AAPL	12.886	12.306	11.951	11.738	11.627	11.705	12.137
		(0.553)	(0.493)	(0.540)	(0.580)	(0.606)	(0.607)	(0.678)
	MSFT	11.177	10.685	10.360	10.125	10.03	10.115	10.687
		(0.634)	(0.538)	(0.553)	(0.603)	(0.589)	(0.569)	(0.741)

Table 3.1 : The average hourly trading volume and log volume over the entire sample period September 2021 to September 2022. H1 9:30-10:00; H2 10:00-11:00; H3 11:00-12:00; H5 1:00-2:00: H6 2:00-3::00; H7 3:00-4:00. Standard deviations are reported in brackets().

the entire sample and are displayed in Table 3.1. Average hourly volume was one of the initial prediction methods used for predicting intraday volume and serves as the base case in many studies in the field. Table 3.1 displays the U-shaped diurnal patterns from both Apple and Microsoft's hourly trading volume. Both securities have higher trading volumes at the open compared to the end of the day. It should also be noted that the standard deviation of the trading volume of both tickers also exhibits a U-shaped pattern with spikes at the beginning and end of the trading day, see Figure B.3. Similarly, log volume, also exhibit the U-shaped daily pattern for both securities.

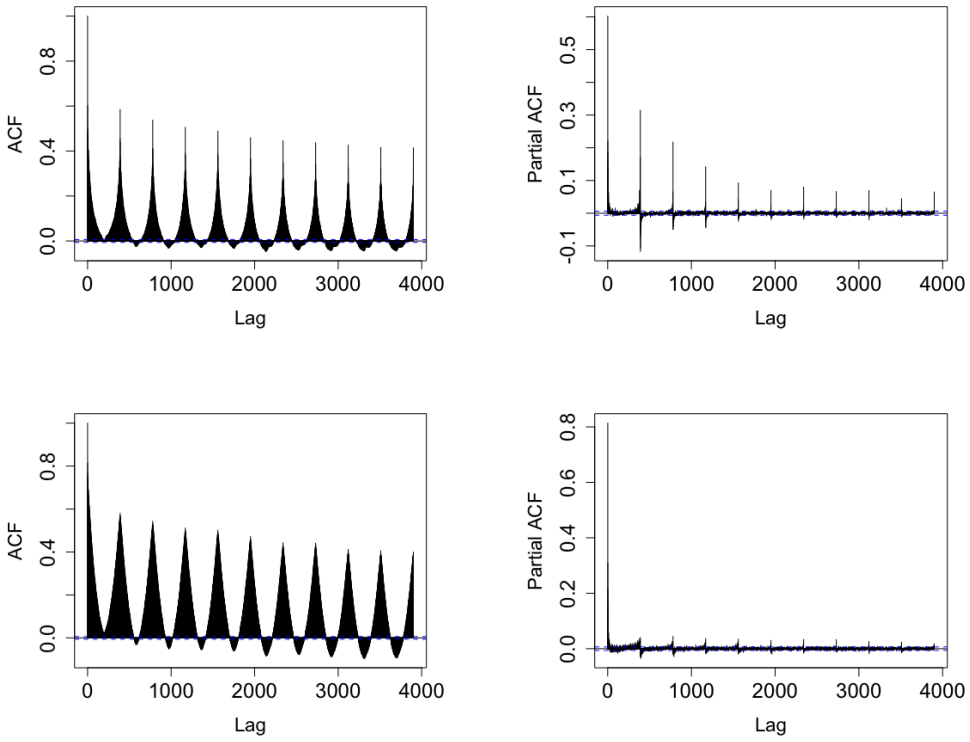


Figure 3.1 : Apple acf (left), pacf (right), of volume (top) and log volume(bottom) binned at the 1-minute intervals.

To further investigated the seasonality of the target data sets, the autocorrelation function (ACF) and partial autocorrelation function (PACF) are calculated for the entire sample for volume and log volume for both Apple and Microsoft. The ACF/PACF analysis for Apple can be found in Figure 3.1 and Figure B.2 contains the analysis for Microsoft. Both appear to have exponentially decaying spikes around multiples of 390. According to traditional time series analysis, this is a sign of daily seasonality and would

normally point to an ARMA(P,Q)[390] model given that neither the ACF nor PACF values cut off after a certain lag and rather tail off at lags that are multiples of 390 minutes. For example, AR(P)[s] models have PACF values that cut off after lag Ps and MA(Q)[s] models have ACF values that cut off after lag Qs. This daily pattern is most obvious in the ACF/PACF graphs of 1-minute volume for both securities but the pattern also appears in the acf/pacf graph of log 1-minute volume. Given the seasonal pattern in the ACF/PACF and the apparent seasonality visible in the graphs of average hourly trading volume and log trading volume, it seems appropriate to employ methods suited to analyze seasonality within time series.

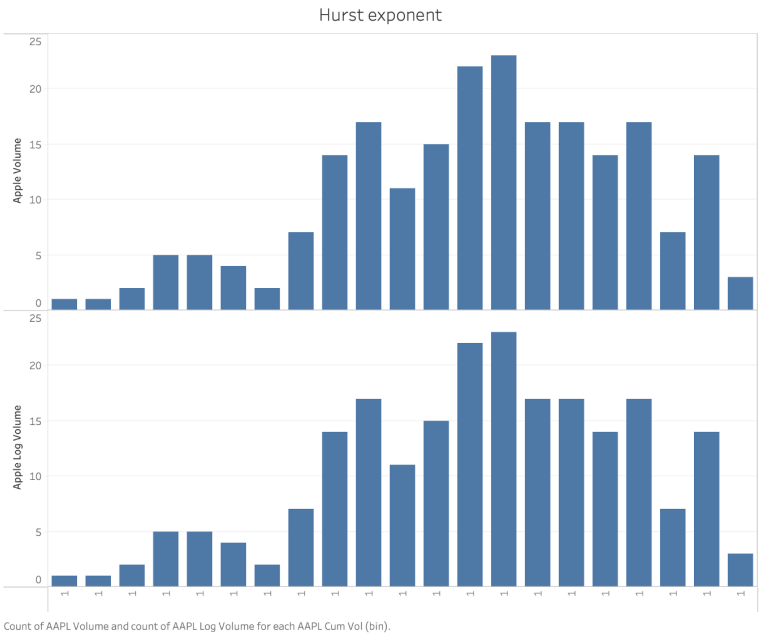


Figure 3.2 : Hurst exponent distribution for Apple using samples of size 10,000.

Similarly, the ACF and Hurst Exponent of both securities were calculated to investigate long memory within the volume and log volume time series. The sample autocorrelation functions, which are not large but tend to persist for a long time, exhibit what traditionally time series analysts refer to as long memory (73). This pattern is visible in the ACF of both securities, Figure 3.1 and Figure B.2, and is strongest for volume but also present for log volume. Similarly, the Hurst exponent, which is used as a measure of the long memory in a time series, was also calculated for both securities and for all 3 time series using the 'hurstexp()' function in the 'pracma' R package, which returns a re-scaled range analysis based method, a corrected re-scaled range analysis based method, an empirical method, a corrected empirical method, and a theoretical method. The Hurst exponent, introduced by (36), ranges between 0 and 1 and effectively classifies a-time series into 3 categories. If H=.05, the time series is categorized as random, $0 < H < .5$ indicates a persistent, mean-reverting, series, and $H > .5$ indicates a persistent series, which are trend reinforcing (64). The Hurst exponent analysis is reported in Figure 3.2 and B.4. The distributions of Hurst exponent values are all larger than .5 for volume and log volume for both Apple and Microsoft. Given both of these facts, it seems apparent that neural networks that handle long memory, like the LSTM and NARX, would be appropriate to predict these financial time series.

3.3 Methods

The CoFES Seasonal NARX augments the Non-Linear Autoregressive Neural Network with Exogenous Variables (NARX), first introduced in (46), which is a class of RNN that was formulated to model long dependencies in sequential data. The NARX, which has a recurrent dynamic architecture with several hidden layers, was inspired by the nonlinear autoregressive with exogenous inputs, a discrete-time nonlinear model found in advanced nonlinear time series applications (11). However, unlike traditional RNNs, the recurrence in the NARX model only occurs in the feedback on the output, rather than from the hidden states, see Figure 3.3. Fortunately, (46) suggest that this design allows for the NARX to be implement via a multilayer perceptron (MLP) where the target $y[t]$ is regressed against d_y lagged values, $\{y[t-d_y], ..., y[t-1]\}$, and d_x lagged values of values of an exogenous input signal $\{x[t-d_x], ..., x[t-1]\}$.

The NARX inputs, i_t, consist of two Tapped Delays Lines.

$$i_t = (x_{t-d_x}, ..., x_{t-1}, y_{t-d_y}, ..., y_{t-1}) \tag{3.1}$$

The NARX output equation is given by:

$$y_t = f(x_{t-d_x}, ..., x_{t-1}, x_t, y_{t-d_y}, ..., y_{t-l}, \Theta) \tag{3.2}$$

where $f(\cdot)$ is a nonlinear mapping function performed by a MLP, Θ are the parameters (weights & bias), d_x is the input delay, and d_y represent the output delay (11). The input to the NARX, i_t, has $d_x N_x + d_y N_y$terms.

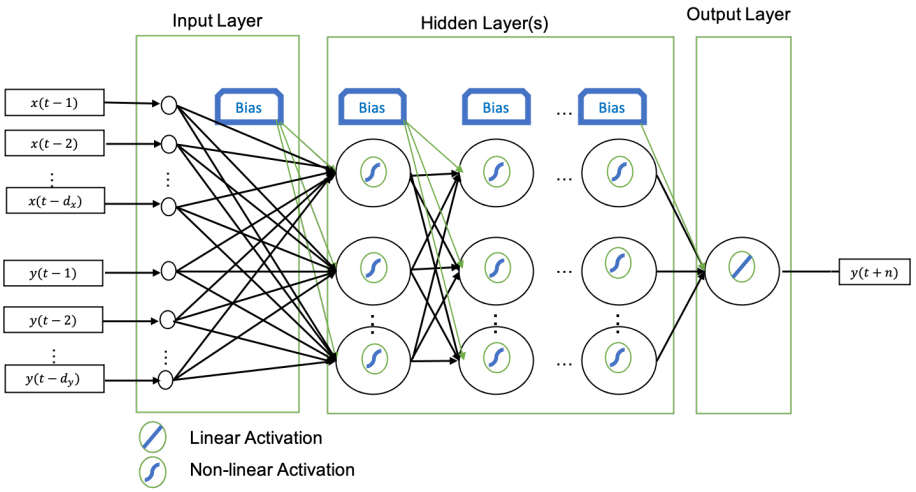


Figure 3.3 : NARX

Where N_x and N_y are the number of inputs and output variables.

The MLP used to model the NARX has a single input layer, $N_l \geq 1$ hidden layers each with its own number of hidden nodes $(h_1, ..., h_l)$, and a single output layer. The following equations dictate the output for the network.

$$h_1[t] = t(i_t, \theta_i) \tag{3.3}$$

$$h_l[t] = t(h_{l-1}[t-1], \theta_{hl}) \tag{3.4}$$

$$y_t = l(h_{N_l}[t-1], \theta_0) \tag{3.5}$$

where $h_l[t] \in \mathbb{R}^{N_{h_l}}$ is the output of the l^{th} hidden later at time t, l$(\cdot)$ is the identity function, and t$(\cdot)$ is an activation function, which is ei-

ther tahn or sigmoid for the NARX. The weights of the network are $\Theta = \{\theta_i, \theta_o, \theta_{h_1}, ..., \theta_{h_{N_l}}\}$, where $\theta_i = \{W_i^{h_1} \in \mathbb{R}^{(d_x N_x + d_y N_y) \times N_{h_1}} \; b_{h_1} \in \mathbb{R}^{N_{h_1}}\}$, $\theta_o = \{W_{h_{N_l}}^o \in \mathbb{R}^{N_l \times N_y} \; b_o \in \mathbb{R}^{N_y}\}$, and $\theta_{h_l} = \{W_{h_l-1}^{h_l} \in \mathbb{R}^{N_{(h_l-1)} \times N_{h_l}} \; b_{h_l} \in \mathbb{R}^{N_{h_l}}\}$ for $l = 1, .., N_l$.

During training, the time series relative to the desired output y* is fed into the network along with the input time series x. Then the output feedback, the recurrence that connects the output of the network with the input target node, is disconnected and the resulting network has a purely feed-forward architecture, in which parameters can be trained using well-established backpropagation techniques(11). The NARX employs the following loss function in the gradient descent.

$$L(y, y*; \Theta) = MSE(y, y*) + \lambda_2 ||\Theta||_2 \tag{3.6}$$

where MSE = Mean Square Error, λ_2 a hyperparameter that weights the importance of the L_2 regularization term in the loss function. The initial phase of λ_2 is transient, the first prediction of y is initially fed back into the network as input and disregarded. The NARX network has 5 hyperparameters: the input and output TDL, the number of hidden layers, the number of neurons in each layer, the regularization hyperparameter λ_2 in the loss function, and the learning rate (11).

The CoFES S-NARX extends the NARX by introducing a seasonally lagged version of the target series, y_{t-d_s}, to the feature set of the network without adding any of the rigid assumptions, like linearity and stationarity, associated with the ARMA(P,Q)[s] model, see Figure 3.4. The season-

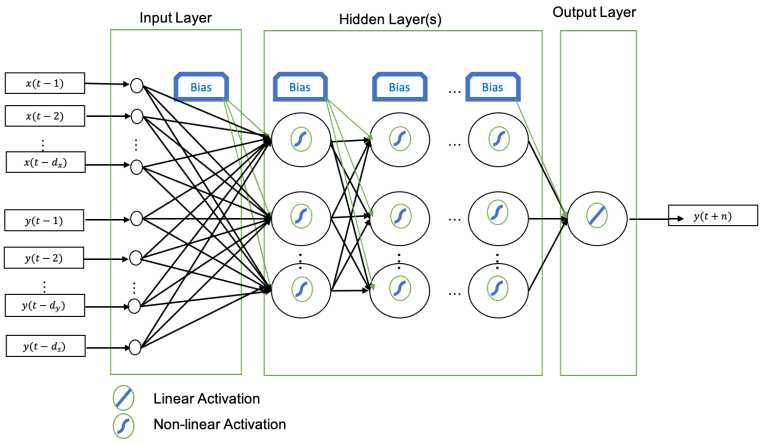


Figure 3.4 : CoFES S-NARX when P=1

ally lagged version of the original series represents the key feature in the ARMA(1,0)[s], which is a pure seasonal autoregressive version of (72)'s seasonal autoregressive moving average model ARMA(P,Q)[s] given by:

$$\Phi_P(B^S)y_t = \Theta_Q(B^S)w_t \tag{3.7}$$

where $\Phi_P(B^S)$ is the **seasonal autoregressive back shift operators** given by

$$\Phi_P(B^S) = 1 - \Phi_1 B^S - \Phi_2 B^{(2S)} - ... - \Phi_p B^{(PS)} \tag{3.8}$$

and $\Theta_Q(B^S)$ is the **seasonal moving average operator** given by

$$\Theta_Q(B^S) = 1 + \Theta_1 B^S - \Theta_2 B^{(2S)} - ... - \Theta_p B^{(QS)} \tag{3.9}$$

where B is the **backshift operator** defined by

$$B^n x_t = x_{t-n} \tag{3.10}$$

ARMA(P,0)[s] model is are given by

$$(1 - \Phi_1 B^S - \Phi_2 B^{(2S)} - ... - \Phi_p B^{(PS)}) y_t = w_t \tag{3.11}$$

Which simplifies to

$$y_t = \Phi_1 y_{t-s} + \Phi_2 y_{t-2s} + ... + \Phi_p y_{t-ps} + w_t \tag{3.12}$$

The CoFES S-NARX augments the NARX feature set with the features from ARMA(P,0)[s] model because this model's feature set represents pure seasonal inputs unlike other multiplicative seasonal time series methods in (72) that result in additional nonseasonal terms due to the difference and seasonal difference components. This study uses P=1. ARMA(1,0)[s] is given by:

$$y_t = \Phi_1 y_{t-s} + w_t \tag{3.13}$$

The study attempt to forecast time t+n given information at time t, so the ARMA(1,0)[s] becomes

$$y_{t+n} = \Phi_1 y_{t+n-s} + w_{t+n} \text{ for } s \geq 0 \tag{3.14}$$

The CoFES S-NARX output equation is given by:

$$y[t+n] = f(x[t-d_x], ..., x[t-1], x[t], y[t+n-d_s], y[t-d_y], ..., y[t-l], \Theta) \tag{3.15}$$

where $f(\cdot)$ is a nonlinear mapping function performed by a MLP, Θ are the parameters (weights & bias), d_x is the input delay, d_y represent the

output delay, d_s represent the seasonal lag and $d_s \geq 0$, and n is the forecast window which describes the number of time steps forward that the network predicts. This augments the NARX inputs i_t, has $d_x N_x + (d_y + P)N_y$ terms which changes the weights of the input layer to $\theta_i = \{W_i^{h_1} \in \mathbb{R}^{(d_x N_x + (d_y+P)N_y) \times N_{h_1}}\ b_{h_1} \in \mathbb{R}^{N_{h_1}}\}$. A higher order ARMA(P,0)[s] is possible but machine learning algorithms attempt to balance the number of features with the potential problem of over fitting and we must be careful not to add too many variables to a the feature set. We investigate the results from P=1,2 and $d_s = 390$, which was motivated by the diurnal seasonal pattern in high frequency intraday volume data and the fact that the data was reported at the minute level.

3.4 Research Methodology

The exogenous variables consisted of open, high, low, and close price data and seven traditional volume-based technical indicators, On Balance Volume (OBV), Volume Relative Strength Index (V.RSI), Money Flow Index (MFI), Chaikin Money Flow (CMF), Chaikin Accumulation / Distribution (chaikinAD), Arms' Ease of Movement Value (EMC), and a 10 period simple moving average (SMA) of log volume. The models make minute-by-minute log(volume) predictions with a prediction window that varies from n=[1,5,10,15] minutes in the future. This study also considers variations that include a quantitative variable for minute of the trading day(**m**), a minute of the hour (**M**), and hour of the day (**H**) in the feature set of the

NARX based models, which are designated (HmM). Forecasting accuracy is measured by mean absolute percentage error (MAPE), mean absolute error (MAE), and root mean square error (RMSE). The CoFES S-NARX and the NARX are programmed in R and are based on the sequential model found in the Keras package.

Data

The sample data in this study was supplied by Vanguard L.P. and consists of high frequency open, high, low, and close (OHLC), volume, VWAP, and a number of trades. Minute level data was collected for all 100 companies in the NASDAQ 100 between 09/01/2020 to 09/01/2021. Missing data points are replaced using the 5-minute moving average. The technical indicators were calculated using the TTR package in R. The models are trained on 63 days of minute level data, or 24,750 data points, and then makes 15 days or 5850-minute level predictions into the future. The training phase uses 20% of the data, or 4950 data points, for validation and the metrics from validation are used to select hyperparameters during hyperparameter tuning. Hyperparameter tuning was completed using the Tfruns package in R and all neural networks were built and executed using Keras in R. This study tested a total of six strategies to predict the log(volume) of Apple and Microsoft shares over a 15 trading day window.

Model Inputs

There are several volume technical indicators to choose from given the fact that there is useful information on the level of volume and how the level has changed recently. This study employs the technical indicator *On Balance Volume (OBV)*, a measure of an asset's buying/selling pressure, Volume Relative Strength Index (V.RSI), a measure of price trend change, Money Flow Index (MFI), an oscillator that attempts to quantify when an asset is either overbought or oversold, Chaikin Money Flow (CMF), which quantifies money flow volume over a given period, Chaikin Accumulation / Distribution (chaikinAD), which quantifies the cumulative flow of money in and out of an asset, Arms' Ease of Movement Value (EMC), which quantifies how easily a price can increase or decrease based on the relationship between price and volume, and a 10 period simple moving average (SMA) of log volume(18).

$$OBV = \frac{c - p}{|c - p|} \times V$$

where c is the current period's closing price, p is the previous period's closing price, and v is the current period's volume.

$$V.RSI = 100 - \frac{100}{1 + \frac{\sum \text{Gains over past 14 periods}}{\sum \text{Losses over past 14 Periods}}}$$

.

$$\text{Money Flow Index } = 100 - \frac{100}{1 + \frac{\text{14 Period Positive Money Flow}}{\text{14 Period Negative Money Flow}}}$$

where Raw Money Flow = Volume * $\frac{high+low+close}{3}$. If price increases, Raw Money Flow is positive and it is added to Positive Money Flow and if price

decreases, Raw money Flow is added to Negative Money Flow.

$$\text{CMF} = \frac{\text{21-day EMA of MFV}}{\text{21-day EMA of Volume}}$$

where Money Flow Volume (MFV) = $\frac{\text{(Close - Low)-(High - Close)}}{\text{High-Low}}$*Volume for the Period

$$\text{chaikinAD} = \text{Volume}*\frac{(c-l)-(h-c)}{h-l}$$

where c= close, h=high, and l=low.

$$\text{EMC} = \frac{(h+l)}{2} = \frac{(h_p+l_p)}{2}/\frac{v}{(h-l)}$$

where h=current period's high, l=current periods low, h_p previous period's high, l_p=previous period's low, and v=current periods volume. The 10 period simple moving average (SMA_k) of log volume log(v) at period k is given by

$$SMA_k = \frac{1}{n}\sum_{k=-n+1}^{k} log(v_i)$$

where n is the number of periods included in the moving average. This study also adds 3 quantitative variables, ***HR*** represents the hour of the trading day [9-15], **MIN** represents the minute of the hour [0-59], and ***min*** represents the minute of the trading day [1-390], to the feature set to attempt to improve the networks ability to model seasonality.

The Architecture of the Models

This study compares the predictive performance of the CoFES S-NARX, CoFES S-NARX-(HmM), the NARX, NARX-(HmM), LSTM, and LSTM-AR-HR, the best model from(45), on log(volume). Each network in this

study contained 3 hidden layers with 100 nodes in the first hidden layer, 50 nodes in the second hidden layer, and 25 nodes in the last hidden layer. Models are trained on 3 months of daily high-frequency data and then make 3 weeks of predictions for n=1,5,10,15 minutes into the future. The training window is then slid 1 day and the process repeats 100 times to get a good idea of the model's predictability within the year's worth of daily data. The S-NARX_P2 and S_NARX-HmM_P2 use P=2 in the S-NARX framework and introduce two seasonally lagged to the S-NARX feature set, s and 2s.

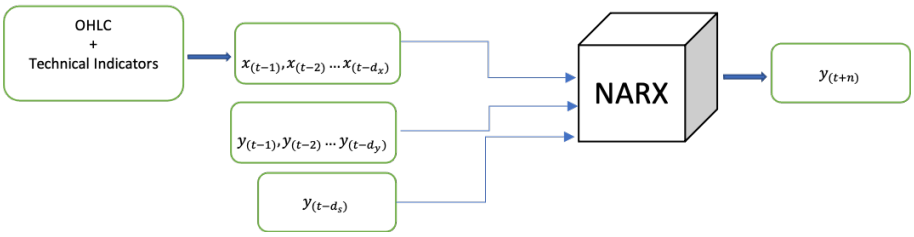


Figure 3.5 : CoFES S-NARX model configuration

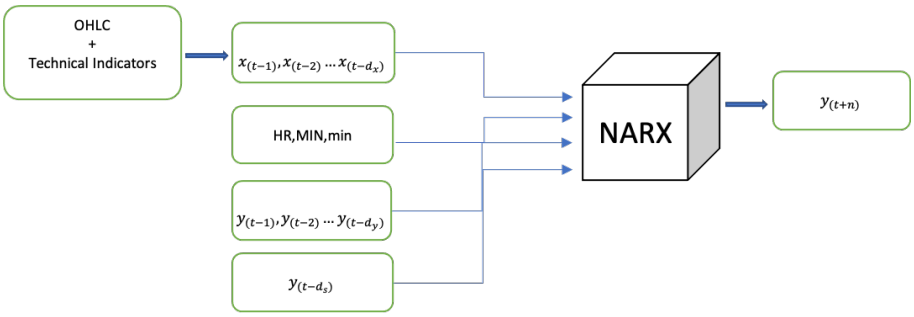


Figure 3.6 : CoFES S-NARX-HmM models configuration

The CoFES S-NARX models were fit with d_s=390, and the value d_x and

d_y were both determine via hyperparameter tuning for each model, where d_x and d_y ranged between [1-3] and [5-10] respectively. The feature set consists of the open, high, low, close, OBV, MFI, ,CMF, (chaikinAD),(EMC),(SMA), $\{y_{t-1}, y_{t-2}, ..., y_{t-d_y}\}$ and y_{t-d_s}. The NARX models were trained on the exact same parameters and feature set as the CoFES S-NARX excluding y_{t-d_s}. The CoFES S-NARX-(HmM) and NARX-(HmM) augment the feature set of the CoFES S-NARX and the NARX with HR, MIN, and min variables calculated from the intraday data. The S-NARX and S_NARX-HmM models are represented in Figure 3.5 and Figure 3.6 respectively. The NARX and NARX-HmM models are represented in Figure 3.7 and Figure 3.8 respectively.

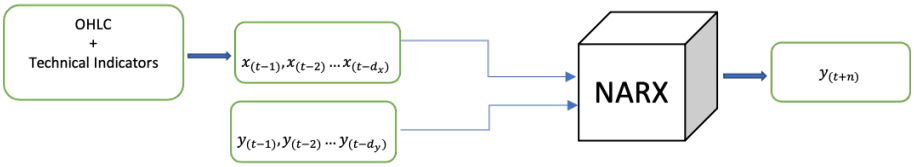


Figure 3.7 : NARX models configuration

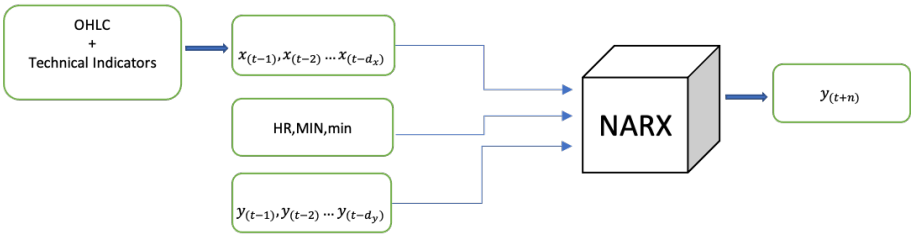


Figure 3.8 : NARX-HmM model configuration

LSTM models were trained with the following feature set: open, high, low, close, OBV, MFI, CMF, (chaikinAD),(EMC), (SMA) and were given a 5-minute window which was equal to the maximum d_x value used in the CoFES S-NARX and NARX models. The LSTM-AR-HR augments the feature set of the LSTM with the HR variable and the results from separate AR(1) models fit on the target and price variables. The LSTM and LSTM-AR-HR models are represented in Figure 3.9 and Figure 3.10 respectively.

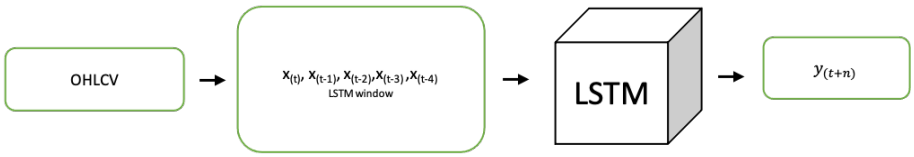


Figure 3.9 : LSTM model configuration

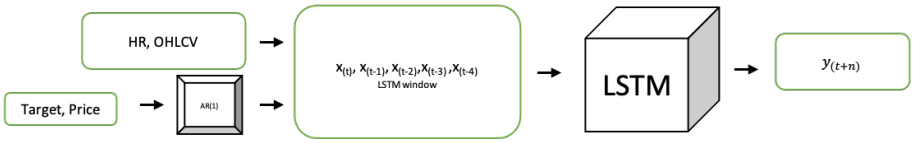


Figure 3.10 : LSTM-AR-HR model configuration

Forecast Accuracy

To focus analysis on the last 15 minutes of trading, which is of great importance to algorithmic traders, the MAPE, MAE, and RMSE are also calculated for just this subset of the forecast results for just the last 15 minutes

of each trading day in this study's prediction window. We will refer to these calculations as MAPE15, MAE15, and RMSE15 in our results section. MAPE15 is defined as

$$\text{MAPE15}(\hat{y_{15}}, y_{15}) = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_{15} - \hat{y_{15}}|}{y_{15}} \tag{3.16}$$

MAE15 is defined as

$$\text{MAE15}(\hat{y_{15}}, y_{15}) = \frac{1}{n}\sum_{i=1}^{n}|y_{15} - \hat{y_{15}}| \tag{3.17}$$

RMSE15 is defined as

$$\text{RMSE15}(\hat{y_{15}}, y_{15}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{15} - \hat{y_{15}})^2} \tag{3.18}$$

where y_{15} represents the actual values in the last 15 minutes of the trading day and $\hat{y_{15}}$ the predicted values in the last 15 minutes of the trading day.

3.5 Experimental Results

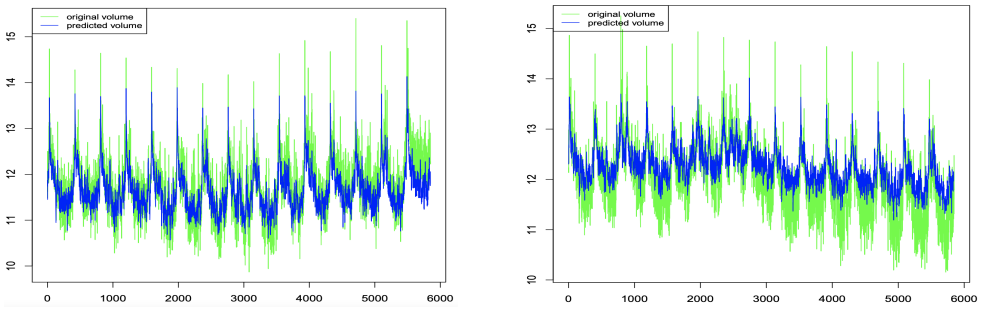


Figure 3.11 : A result from the S-NARX (left) and from the S-NARX-HmM (right) when predicting Apple's log trading volume.

The study measured the model's forecast ability using MAPE15, MAE15, and RMSE15. The results are compared for the six different RNN-based techniques: CoFES S-NARX, CoFES S-NARX-HmM, LSTM, LSTM-AR-HR, NARX, and NARX-HmM. Each model's network consisted of 3 hidden layers(100,50,25). The models are trained on training data set, 63 days of minute level intraday data, and make 15 days worth of n={1,5,10,15} minutes predictions into the future from the testing data set. Performance is judged based on the lowest average MAPE15, MAE15, and RMSE15.

3.5.1 Apple

	n=1			n=5			n=10			n=15		
Model	MAPE15	MAE15	RMSE15	MAPE15	MAE15	RMSE15	MAPE15	MAE15	RMSE15	MAPE15	MAE15	RMSE15
LSTM	6.29%	0.837	1.057	7.40%	0.989	1.152	8.04%	1.075	1.202	7.63%	1.019	1.162
LSTM_AR_HR	6.23%	0.829	1.049	7.39%	0.988	1.151	8.08%	1.081	1.207	7.66%	1.022	1.165
NARXHmM	4.71%	0.617	0.761	5.47%	0.726	0.876	5.87%	0.782	0.946	5.49%	0.732	0.932
S-NARX-HmM	4.47%	0.585	0.723	5.39%	0.715	0.861	5.92%	0.789	0.955	5.61%	0.744	0.936
S-NARX-HmM_P2	4.35%	0.568	0.708	5.86%	0.776	0.921	5.71%	0.762	0.935	5.45%*	0.726*	0.931*
S-NARX	4.24%	0.553	0.686	5.59%	0.741	0.886	5.59%*	0.746*	0.912*	5.76%	0.767	0.964
NARX	4.15%*	0.540*	0.675*	5.16%*	0.685*	0.837*	6.42%	0.857	1.022	5.68%	0.757	0.957

Table 3.2 : Average MAPE15, MAE15, RMSE15 results from 100 training runs for n={1,5,10,15} for Apple's log volume predictions over the last 15 minutes of the trading day.

Overall, the NARX-based models outperformed the LSTM-based model over the last 15 minutes of the trading day in terms of the average MAPE15,

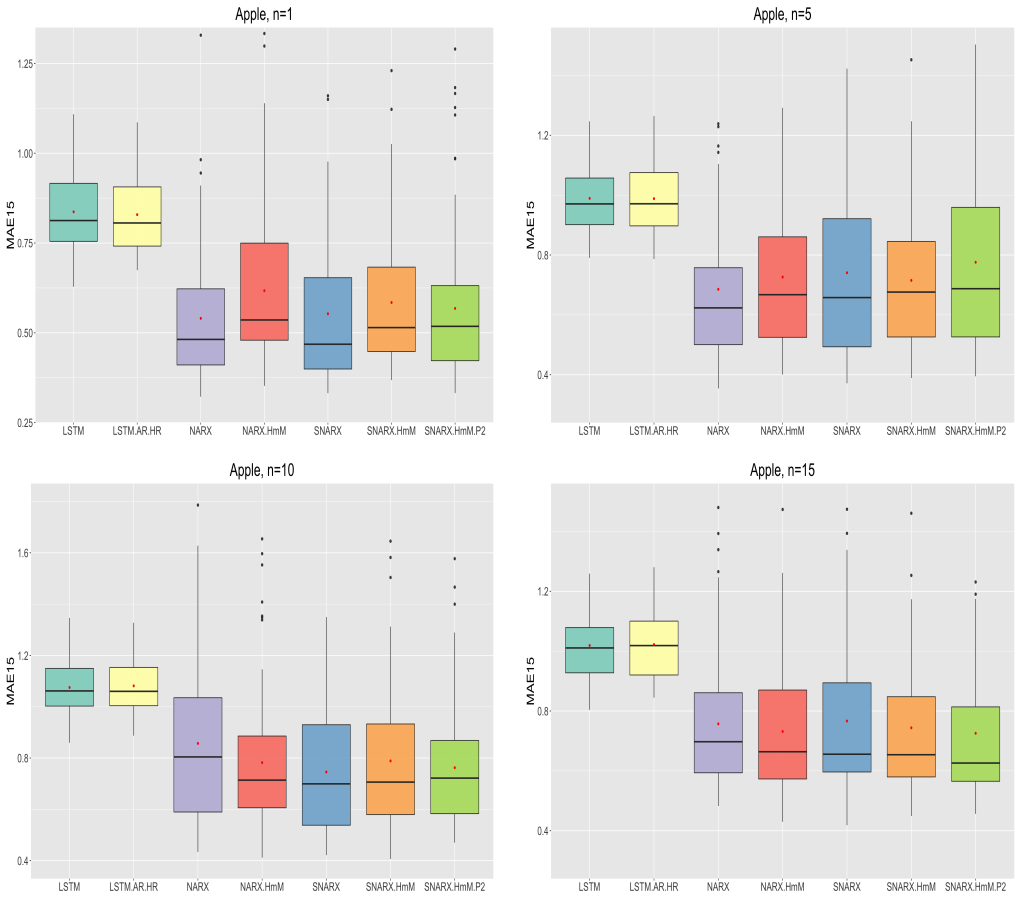


Figure 3.12 : MAE15 results for Apple's log (vol) prediction, n = 1(upper left), n = 5(upper right), n = 10 (lower left), and n = 15 (lower right).

MAE15, and RMSE15. Both LSTM models, LSTM and LSTM-HR-AR, when predicting log volume for Apple resulted in lines that look approximately horizontal, see Figure 3.13. On the other hand, the NARX-based models successfully modeled the seasonal pattern in the data, see Figure B.5. The NARX model had the lowest average metrics for n=1 and n=5. The

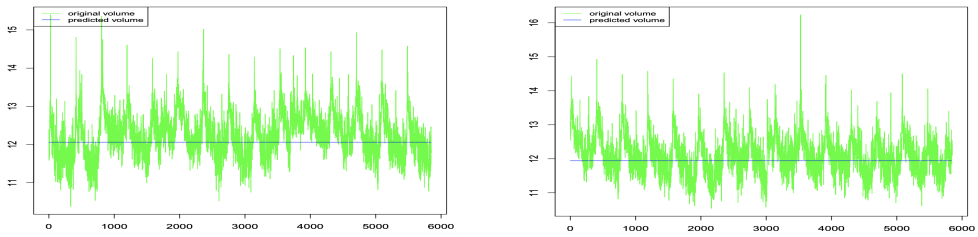


Figure 3.13 : The result from the LSTM (left) and LSTM-AR-HR (right) model's prediction of AAPL's log intraday trading volume.

S-NARX had the lowest metrics for n=10 and the S-NARX-HmM_P2 had the lowest metrics for n=15. Overall the NARX, S-NARX, and S-NARX-HmM_P2 seem to consistently have the lowest average MAPE15, MAE15, and RMSE15 for minute-by-minute prediction of Apple's log trading volume. The NARX-based model, NARX and S-NARX, appear to outperform the LSTM at all values of n, and as n increases the level of outperformance increases as well. From the box plots of Apples MAE15 scores for n=1,5,10,15, see Figure 3.12, it appears that as n increases the S-NARX based models have lower variability, as evidenced by the range and interquartile range, than the NARX based models. This is more apparent in the Microsoft results.

3.5.2 Microsoft

Similar to Apple's results, the NARX based models outperform the LSTM based models when predicting intraday volume. The LSTM models were not able to pick up on the diurnal pattern in the Microsoft log volume data

	n=1			n=5			n=10			n=15		
Model	MAPE15	MAE15	RMSE15	MAPE15	MAE15	RMSE15	MAPE15	MAE15	RMSE15	MAPE15	MAE15	RMSE15
LSTM_AR_HR	9.36%	1.129	1.369	10.39%	1.254	1.439	10.25%	1.237	1.433	8.50%	1.018	1.267
LSTM	9.35%	1.129	1.369	10.41%	1.257	1.442	10.24%	1.237	1.432	8.54%	1.022	1.271
NARX_	7.28%	0.852	0.988	7.50%	0.894	1.062	8.95%	1.067	1.284	7.57%	0.886	1.139
S-NARX	6.64%	0.780	0.917	7.71%	0.922	1.089	8.10%	0.970	1.194	7.56%	0.886	1.144
S-NARX_P2	6.63%	0.779	0.913	7.79%	0.928	1.095	8.49%	1.008	1.212	7.65%	0.897	1.155
S-NARX-HmM_P2	6.13%	0.719*	0.859	7.10%*	0.844*	1.011*	7.60%*	0.899*	1.099*	7.35%	0.853	1.087*
S-NARX-HmM	6.11%	0.719*	0.857*	7.51%	0.890	1.055	7.72%	0.914	1.115	7.12%*	0.836*	1.088
NARXHmM	6.10%*	0.720	0.865	7.82%	0.932	1.099	8.59%	1.025	1.239	8.06%	0.941	1.193

Table 3.3 : Average MAPE15, MAE15, RMSE15 results from 100 training runs for n={1,5,10,15} for Microsoft's log daily trading volume over the last 15 minutes of the trading day.

and instead predicted close to the same value for the entire 15 day, minute-by-minute, prediction window. For n=1, the NARX-HmM had the lowest MAPE15 and the S-NARX-HmM had the lowest MAE15 and RMSE15. For n=5, the S-NARX-HmM_P2 had the lowest metrics with S-NARX-HmM not far behind. For n=10, the S-NARX-HmM_P2 had the lowest metrics again with the S-NARX-HmM not far behind. For n=15, the S-NARX-HmM had the lowest MAPE15 and S-NARX-HmM_P2 had the lowest MAE15 AND RMSE15. The S-NARX-HmM and S-NARX-HmM_P2 appear to consistently have the lowest metrics for the tasks of predicting the last 15 minutes of Microsoft's log trading volume. The box plots from the MAE15 results for Microsoft, see Figure 3.14, suggest that like Apple the S-NARX based mod-

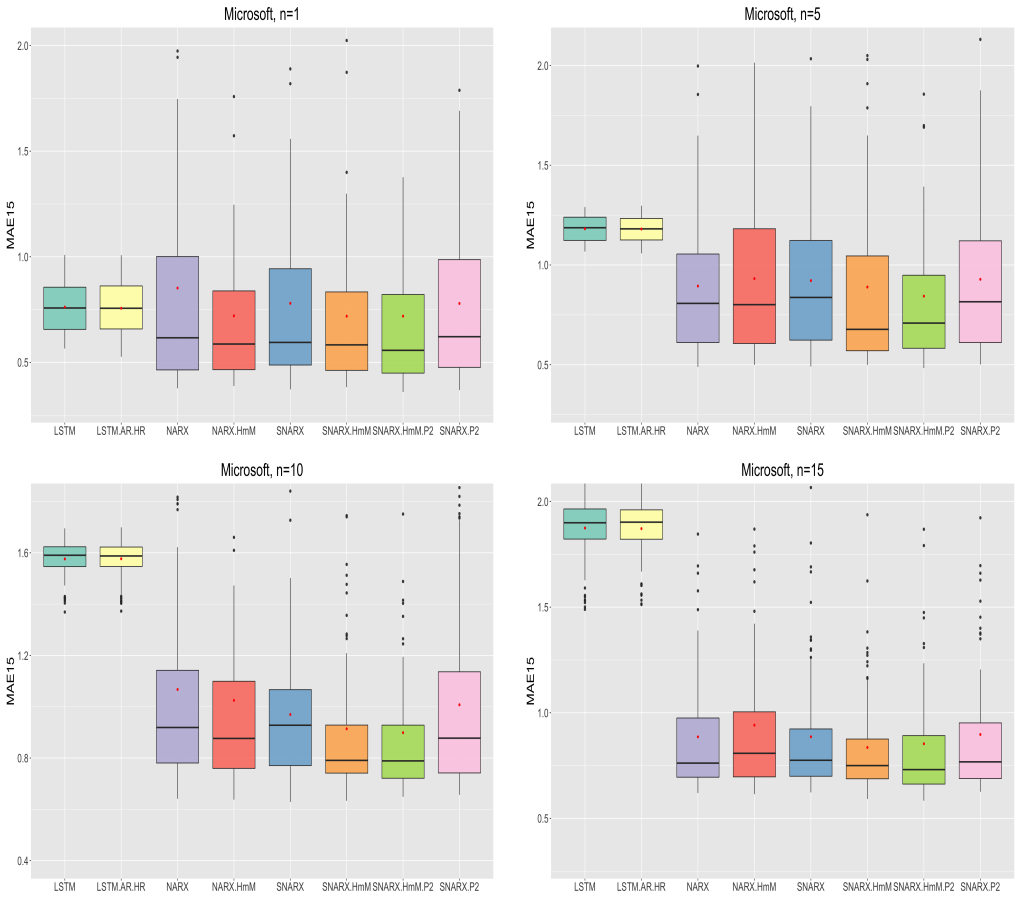


Figure 3.14 : MAE15 results for Microsoft's log (vol) prediction, n = 1(upper left), n = 5(upper right), n = 10 (lower left), and n = 15 (lower right) .

els appear to have lover variability in the prediction results as n increases which separates it from the NARX and NARX-HmM. This suggests that the prediction results from the S-NARX-based models may be more consistent than the NARX-based models as n increase.

3.6 Conclusion

In this study, we investigate the predictive performance of the CoFES S-NARX network and an altered version of this model, the S-NARX-HmM, that includes three time-related quantitative trend variables, on intraday log volume from two of the most heavily traded American technology stocks. The CoFES S-NARX's results are compared against LSTM, the state of the art time series recurrent neural network, the LSTM-AR-HR which combines with the results from an AR(1) and a quantitative variable for the hour of the day to the LSTM feature set, the NARX, and the NARX-HmM. The study trains the networks on 3 months of data using an 80/20 testing validation split and then makes 15 days, or 3 weeks, of n=1,5,10,15 minute(s) ahead predictions of log volume. Then the window slides one day and the process is repeated 100 times. The mean is then taken from the 3 prediction performance metrics MAPE15, MAE15, and RMSE15.

Predicting intraday volume is important to a variety of trading scenarios yet surprisingly there are not many studies done on predicting intraday volume using machine learning techniques. This makes it is difficult to find apple to apple comparisons to determine if these results are on par with the field. In particular, we focused on our model's ability to predict the transformation of intraday trading volume over the last 15 minutes of the trading day. For example, (45) used the LSTM to predict intraday volume but binned the data into 10-minute bins and used the past five 10 minute bins to predict the log change in volume over the next 10-minute bin, where

the bin volume was calculated as the sum of the volume over the 10-minute window. Our strategy is more comparable to other high-frequency strategies that also produce minute-level predictions of intraday volume using machine learning methods. From the results, the CoFES S-NARX seems to be a strong alternative for individuals looking to use machine learning-based predictions of time series with seasonality. The CoFES S-NARX didn't completely separate itself from the NARX models when comparing the average metrics from prediction of log volume, but it does a more consistent job, resulting in less variability in the predictions as n increases. Ultimately, this is a desirable property given these are minute by minute predictions and larger prediction windows will increases the number of applications with which this strategy can be paired.

Chapter 4

Conclusion

In this work, we developed two momentum-based quantitative trading algorithms that first use wavelet techniques to denoises the OHLC price series and then use the NARX to make a series of 1 day ahead predictions for the return of American Index future contracts. The results of this prediction are fed into a trading algorithm that goes long the open if the sign of the predicted return is positive and takes the opposite position if the sign of the predicted return is negative. Given the choice of the neural network and the accessibility of the wavelet denoising techniques in R, these two strategies can be implemented by someone in the industry or a retail investor with limited computational power. We have also formulated a seasonal version of the NARX based on the ARMA(P,0)[s] that is also straightforward enough for retail investors to understand and employ. The LSTM will forever be the best in the show for time series with longer dependencies, but the network is computationally expensive and the NARX and CoFES S-NARX are alternatives for instances where the underlying dynamics of the data are not modeled well with the LSTM, like intraday volume at the minute level. The S-NARX-based models were able to capture the "U" diurnal pattern in high-frequency trading volume data and produced accurate predictions

for the future. Similarly, the NARX and S-NARX models did well in the last 15 minutes of the trading day. The S-NARX-based models separated themselves from the NARX-based models as n increased and were able to provide less variability in the predictions and lower average metrics.

Still many open questions remain. In particular, can the two trading algorithms be generalized to other asset classes or investment horizons like intraday or monthly return predictions? How will the algorithm perform with a larger network as this study limited the analysis to a single hidden layer with 5 nodes? The paper compared wavelet techniques, so the networks were not individually optimized to allow appropriate comparison between methods. How can optimizing the network, with appropriate hyperparameter tuning, improve the performance of these strategies? From the second study, can splines improve predictions of seasonal intraday data in this high-frequency framework? How will these methods perform in simulation or paper trading in terms of their ability to improve VWAP algorithmic trading strategies? How well do these methods work on small and mid-cap stocks that have more volatility and lower average intraday trading volume?

Chapter 5

R-Package

5.1 Introductions

Finally, we develop an R package, the CoFESSNARX. We recommend referencing the CoFES S-NARX R package that is available at `https://github.com/demonejackson/CofesSNARX` after the thesis is submitted. The package provides the traditional NARX network, first introduced in (46), along with the novel seasonal version the CoFES S-NARX that augments the NARX feature set with the features from an ARMA(P,0)[s] described in (73). The networks are built using the Keras, (4), framework in R and utilizes the sequential model from this package.

5.2 *CoFESNARX* Function

```
#'  CoFESNARX
#'
#'  \code{CoFESNARX} returns a combiled NARX model via Keras
#'
#'  This function takes target variable series, exogenous variables series,
#'  and the number of desired lags for each group respectively and then returns the
#'  requested lag series for each in a single data set.  Ydelay/Xdelay are vectors
#'  that hold the consecutive delays required for the target variable series
#'  (ex 1:3 or 4:5)
```

```
#'
#' @param y,x numeric vectors for the target and exogenous variables
#' @param dim numeric vector for the dimension of the desired network
#' @param dr_o drop out rate used between hidden layers
#' @param KL2 the L2 kernel_regularizerregularization factor for hidden
#' layer kernel regularization
#' @param BL2 the L2 bias_regularizer regularization factor for hidden layer
#' bias regularization
#' @param act activation function used, if not specified 'relu' used
#' @return returns the target combined with the original and lagged exogenous
#' variables in one database.
#'
#'
#'


################################################################################
CoFESNARX <-  function(x, y, dim, KL2 = .01, BL2 = .01,  dr_o = 0, act = 'relu'){
  # This function determine the size of the model and builds the same model with
  # different number of requested nodes.  Dim should be entered using c() i.e.
  # c(10,5) for a two hidden layers with 10 nodes in the first layer and 5
  # nodes in the second.
  model <- keras_model_sequential()
  if (length(dim)==1) {
    model %>%
      layer_dense(units = dim[1], activation = act,
                  kernel_regularizer= regularizer_l2(l = KL2),
                  bias_regularizer = regularizer_l2(l = BL2),
                  input_shape = ncol(x)) %>%
      layer_dropout(rate = dr_o) %>%
      layer_dense(units = 1, activation = 'linear',
                  kernel_regularizer= regularizer_l2(l = KL2))

  } else if  (length(dim)==2) {
    model %>%
```

```
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))
} else if  (length(dim)==3) {
  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))
} else if  (length(dim)==4) {
  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
```

```
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[4], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))

} else if  (length(dim)==5) {
  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[4], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[5], activation = act,
```

```
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))
} else if  (length(dim)==6) {
  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[4], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[5], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[6], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))
```

```
} else if  (length(dim)==7) {
  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[4], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[5], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[6], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[7], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))

} else if  (length(dim)==8) {
```

```
model %>%
  layer_dense(units = dim[1], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2),
              input_shape = ncol(x)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[2], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[3], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[4], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[5], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[6], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[7], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[8], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = 1, activation = 'linear',
```

```
                kernel_regularizer= regularizer_l2(l = KL2))

} else if  (length(dim)==9) {
  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[4], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[5], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[6], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[7], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[8], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
```

```
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[9], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))

} else if  (length(dim)==10) {
  model %>%


    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[4], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[5], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[6], activation = act,
```

```
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[7], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[8], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[9], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[10], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))
} else if  (length(dim)==11) {
  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
```

```
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[4], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[5], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[6], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[7], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[8], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[9], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[10], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[11], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = 1, activation = 'linear',
            kernel_regularizer= regularizer_l2(l = KL2))
```

```
} else if  (length(dim)==12) {
  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation =
                  act, kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[4], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[5], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[6], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[7], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[8], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
```

```
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[9], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[10], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[11], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[12], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))


} else if  (length(dim)==13) {
  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
```

```
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[4], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[5], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[6], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[7], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[8], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[9], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[10], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[11], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[12], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
```

```
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[13], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))

} else if  (length(dim)==14) {

  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[4], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[5], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[6], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
```

```
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[7], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[8], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[9], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[10], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[11], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[12], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[13], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[14], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = 1, activation = 'linear',
```

```
                kernel_regularizer= regularizer_l2(l = KL2))

} else if  (length(dim)==15) {
  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[4], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[5], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[6], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[7], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[8], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
```

```
            bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[9], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[10], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[11], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[12], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[13], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[14], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[15], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
            kernel_regularizer= regularizer_l2(l = KL2))

} else if  (length(dim)==16) {
```

```
model %>%
  layer_dense(units = dim[1], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2),
              input_shape = ncol(x)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[2], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[3], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[4], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[5], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[6], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[7], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[8], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[9], activation = act,
```

```
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[10], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[11], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[12], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[13], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[14], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[15], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[16], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))

} else if  (length(dim)==17) {
```

```
model %>%
  layer_dense(units = dim[1], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2),
              input_shape = ncol(x)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[2], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[3], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[4], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[5], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[6], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[7], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
  layer_dense(units = dim[8], activation = act,
              kernel_regularizer= regularizer_l2(l = KL2),
              bias_regularizer = regularizer_l2(l = BL2)) %>%
  layer_dropout(rate = dr_o) %>%
```

```
layer_dense(units = dim[9], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[10], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[11], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[12], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[13], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[14], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[15], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[16], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[17], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
```

```
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))

} else if  (length(dim)==18) {

  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[4], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[5], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[6], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[7], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
```

```
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[8], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[9], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[10], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[11], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[12], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[13], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[14], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[15], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[16], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
```

```
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[17], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[18], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))


} else if  (length(dim)==19) {

  model %>%

    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape = ncol(x)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[2], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[3], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[4], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
```

```
layer_dense(units = dim[5], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[6], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[7], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[8], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[9], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[10], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[11], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[12], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[13], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
```

```
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[14], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[15], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[16], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[17], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[18], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = dim[19], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
    layer_dropout(rate = dr_o) %>%
    layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2))


} else if (length(dim)==20) {
  model %>%
    layer_dense(units = dim[1], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2),
                input_shape =  ncol(x) ) %>%
```

```
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[2], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[3], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[4], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[5], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[6], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[7], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[8], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[9], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[10], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
```

```
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[11], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[12], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[13], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[14], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[15], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[16], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[17], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[18], activation = act,
            kernel_regularizer= regularizer_l2(l = KL2),
            bias_regularizer = regularizer_l2(l = BL2)) %>%
layer_dropout(rate = dr_o) %>%
layer_dense(units = dim[19], activation = act,
```

```
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
      layer_dropout(rate = dr_o) %>%
      layer_dense(units = dim[20], activation = act,
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2)) %>%
      layer_dropout(rate = dr_o) %>%
      layer_dense(units = 1, activation = 'linear',
                kernel_regularizer= regularizer_l2(l = KL2),
                bias_regularizer = regularizer_l2(l = BL2))
  } else {
    print("Your specification is too large")}
  summary(model)
  return(model)
}
```

5.3 CoFESNARXdata

CoFESNARXdata and CoFESSNARXdata are utility functions that tranform your target and feature data into the appropriate lags for the NARX and S-NARX usage respectively. The user can set the xdelay/ydelay which represent the AR order for the feature set for the target and exogenous variables. The user may also set n using the delay parameter which sets how many time steps in the future the network will predict. CoFESSNARXdata prepares your data for analysis with the S-NARX and allows the user to also set s as either an integer or a vector for seasonal lags and P is set by the length of s. For example, c(12,24) would represent P=2 and s=12.

```
#'  CoFESNARXdata
```

```
#'
#'  \code{CoFESNARXdata} returns the lead/lag series necessary to run the CoFESNARX
#'
#'  This function transforms the target and exegenous series variables for the
#'  CoFESNARX model.  variables series, and the number of desired lags for each
#'  group respectively and then returns the requested lag series for each in a
#'  single  data set.  Ydelay/Xdelay are vectors that holds the consecutive
#'  delays required for the target variable series (ex 1:3 or 4:5).
#'
#' @param yy numeric vectors containing target data
#' @param xx numeric vectors containing feature data
#' @param ydelay integer representing the desired AR order for y
#' @param xdelay integer representing the desired AR order for x
#' @param delay integers representing n, the size of the prediction
#' window into the future
#' @return returns the target combined with the original and lagged exegenous
#' variables in one database.
#'
#'
#'
CoFESNARXdata <-function(yy, xx, ydelay, xdelay, delay=1, date=T){
  # create a sequence up to the number of delays for the target
  lags <- seq(ydelay)
  # create column names for the lags create in the previous line.
  lag_names <- paste("lag", formatC(lags, width = nchar(max(lags)), flag = "0")
                     , sep = "_")
  # A function to create target.
  lag_functions <- stats::setNames(paste("dplyr::lag(.,␣", lags, ")"), lag_names)
  # A function to create lead series for target.
  lead_functions <- stats::setNames(paste("dplyr::lead(.,␣", delay, ")"), 'target')

  ifelse(date,
         # Apply lag function to the target series and create lags
         y<-yy %>% dplyr::mutate_at(vars( colnames(yy)[2] )
                                    , funs_(c(lead_functions,lag_functions))),
```

```
        y<-yy %>% dplyr::mutate_at(vars( colnames(yy)[1] )
                                   , funs_(c(lead_functions,lag_functions))))

  # Remove NA's
  y<-stats::na.omit(y)
  # create a sequence up to the number of delays for the exegenous series
  lagsx <- seq(xdelay+1)
  # create column names for the lags create in the previous line.
  lag_namesx <- paste("lag", formatC(lagsx, width = nchar(max(lagsx))
                                     , flag = "0"), sep = "_")
  # A function to create lags and adds their names.
  lag_functionsx <- stats::setNames(paste("dplyr::lag(.,␣", lagsx, ")"), lag_namesx)
  # Apply lag function to the target series and create lags
  x<-xx %>% dplyr::mutate_at(vars(colnames(xx)[2:ncol(xx)] ), funs_(lag_functionsx))
  # Remove NA's
  x<-stats::na.omit(x)
  # Merge X and Y series
  abc<-merge(y, x)
  # remove NA's
  abc <- stats::na.omit(abc)
  # return boty X and Y
  return(abc)
}
#'  CoFESSNARXdata
#'
#'  \code{CoFESSNARXdata} returns the lead/lag series necessary to run the CoFESSNARX
#'
#'  This function transforms the target and exegenous series variables for the
#'  CoFESSNARX model.  Variables series, and the number of desired lags for each
#'  group respectively and then returns the requested lag series arlong with the
#'  seasonally lagged verson of teh original series for each in a single  data set.
#'
#' @param yy numeric vectors containing target data
#' @param xx numeric vectors containing feature data
#' @param ydelay integer representing the desired AR order for y
```

```
#' @param xdelay integer representing the desired AR order for x
#' @param delay integers representing n, the size of the prediction window
#' into the future
#' @param s integers representing the seasonality desired.  Can be expressed as a
#' list for P>1.
#' @param date boolean value to indicate if your data includes dates
#' @return returns the target combined with the seasonally lagged target variable,
#'  and the original and lagged exogenous variables in one database.
#'
#'
##############################################################################
CoFESSNARXdata <-function(yy, xx, ydelay, xdelay, delay=1, s=NULL, date=T){
  # If s=Null, then ignore the seasonal portion and boils down to prepareNARXdata.
  if(is.null(s)){
    # Traditional series lags.
    lags <- seq(ydelay)
  }
  else {
    # Add S to the end of the traditional series.
    lags <- c(seq(ydelay),s)
  }
  # create column names for the lags create in the previous line.
  lag_names <- paste("lag", formatC(lags, width = nchar(max(lags)), flag = "0")
                     , sep = "_")
  # A function to create target.
  lag_functions <- stats::setNames(paste("dplyr::lag(.,␣", lags, ")"), lag_names)
  # A function to create lead series for target.
  lead_functions <- stats::setNames(paste("dplyr::lead(.,␣", delay, ")"), 'target')

  ifelse(date,
  # Apply lag function to the target series and create lags
  y<-yy %>% dplyr::mutate_at(vars( colnames(yy)[2] )
                             , funs_(c(lead_functions,lag_functions))),
  y<-yy %>% dplyr::mutate_at(vars( colnames(yy)[1] )
                             , funs_(c(lead_functions,lag_functions))))
```

```
  # Remove NA's
  y<-stats::na.omit(y)
  # create a sequence up to the number of delays for the exegenous series
  lagsx <- seq(xdelay+1)
  # create column names for the lags create in the previous line.
  lag_namesx <- paste("lag", formatC(lagsx, width = nchar(max(lagsx))
                                     , flag = "0"),
                      sep = "_")
  # A function to create lags and adds their names.
  lag_functionsx <- stats::setNames(paste("dplyr::lag(.,␣", lagsx, ")"), lag_namesx)
  # Apply lag function to the target series and create lags
  x<-xx %>% dplyr::mutate_at(vars(colnames(xx)[2:ncol(xx)] ), funs_(lag_functionsx))
  # Remove NA's
  x <- stats::na.omit(x)
  # Merge X and Y series
  abc <- merge(y, x)
  # remove NA's
  abc <- stats::na.omit(abc)
  # return both X and Y
  return(abc)
}
```

Appendix A

A.0.1 R-Code: Prediction

This code snippet represents the prediction model where we denoise the data and then feed it into the NARX.

```
# This file is used to process the training data set and generate the
# predicted close price.
################################################################################
library("R.matlab")
library("lubridate")
library("CoFESWave")
library("EbayesThresh")
library("waveslim")
library("wmtsa")
library("phonTools")
library("tidyverse")
library("ggplot2")
library("xts")

### Repeat Entire Process Loop
pdf(NULL)
i<-5
Matlab$startServer(port = 9999)
matlab <- Matlab(port = 9999)
setOption(matlab, "readResult/interval", 900000000000) # Default is 1 second
setOption(matlab, "readResult/maxTries", 90000000000 * (60 / 10))
```

```
##############################################################################
## check if the matlab connection is open
isOpen <- open(matlab)

# Set name variables to aid in downloading data
ch <-c( 's&p500_rawdata.csv','nasdaq_rawdata.csv','dowjones2_rawdata.csv',
        'sp4002_rawdata.csv', 'crude.csv')
nam <-c( 's&p500','nasdaq','dowjones','400mini', 'crudeoil')
##############################################################################
#### 0. Load data

data.raw <- read.csv(file = ch[i])
chr_secrity_name <- nam[i]

## Get Column Names
vec_colname <- colnames(data.raw)

## Get Number of rows and colunns of data.raw
nrow_raw <- nrow(data.raw)
ncol_raw <- ncol(data.raw)

## date processing
# 1 Change Format of date
# 2 Add Year Column
# 3 Add Quarter Column
# 4 Add Month Column
# 5 Add yyyyqq column to the Data

data.raw <- data.raw %>%
  mutate(Date =as.Date(Date,format = "%m/%d/%Y"),
         yr = lubridate::year(Date),
         qtr = lubridate::quarter(Date),
         mo = lubridate::month(Date)) %>%
  mutate(yyyyqq = yr * 100 + qtr)
```

```
# Process the number of quarters

#########################################################################
# 1. Set parameters and prepare data

## 1.1. Basic parameters
N = 62; # number of last data of the target series to be tested by trained net
M = 739; # quantity of first part and second part of data

L = 40; # the period to retrain before getting signal
FF = 204; # this is to bring the testing period F records back to validate
          # future prediction more accurate

qfrom = 1;# this is the quantity of multiperiods we are analysing
#qto   = nqtr_raw; #calculated above
n_rebuild   = 10; # number of time to rebuild the whole thing

## Net related variables
delay      = 5;
hid        = 6;
repeat_net = 1; # number of times trains the network
vratio     = 10;
tratio     = 10; # ratio of validation and test

#########################################################################
## 1.2. Miscellaneous items
draw  = 0; # 125, desired performance of trained net on last N number of
                # data on target
processstep = 0;
spread      = 0; # spread is set according to the broker policy

#########################################################################
## 1.3. Matlab setup

## Set Variables in matlab
```

```
setVariable(matlab,
            repeat_net = repeat_net,
            delay  = delay,    hid    = hid,
            vratio = vratio,   tratio = tratio)
################################################################################
## 1.4. Basic processing
# Takes a slice of original data Set (3627 X 27) and add return data.
data.original <- data.raw %>%
  mutate( ropen   = open ,
          rhigh   = high ,
          rlow    = low ,
          rclose  = close/ lag(close) - 1,
          rRSI    = RSI ,
          rMACD   = MACD ,
          rSIG    = SIG ,
          rFS     = FS ,
          rSS     = SS ,
          rS      = S ,
          rULT    = ULT ,
          rVol    = Vol )  %>%
  mutate(rVol = replace(rVol, is.infinite(rVol), NA),
         rVol = replace(rVol, is.na(rVol), 0),
         rFS = replace(rFS, is.infinite(rFS), NA),
         rFS = replace(rFS, is.na(rFS), 0),
         rULT = replace(rULT, is.infinite(rULT), NA),
         rULT = replace(rULT, is.na(rULT), 0)

  )
data.original<-data.original %>% tidyr::fill(ropen, .direction="downup")
data.original<-data.original %>% tidyr::fill(rhigh, .direction="downup")
data.original<-data.original %>% tidyr::fill(rlow, .direction="downup")
data.original<-data.original %>% tidyr::fill(rclose, .direction="downup")
data.original<-data.original %>% tidyr::fill(rRSI, .direction="downup")
data.original<-data.original %>% tidyr::fill(rMACD, .direction="downup")
data.original<-data.original %>% tidyr::fill(rSIG, .direction="downup")
```

```
data.original<-data.original %>% tidyr::fill(rFS, .direction="downup")
data.original<-data.original %>% tidyr::fill(rSS, .direction="downup")
data.original<-data.original %>% tidyr::fill(rS, .direction="downup")
data.original<-data.original %>% tidyr::fill(rULT, .direction="downup")
data.original<-data.original %>% tidyr::fill(rVol, .direction="downup")

################################################################################
# Create Vectors of yyyyqq and calculate lengths of vector
Vec_yyyyqq <- sort(unique(data.original$yyyyqq))

# drop the 1st and last quarter.
Vec_yyyyqq <- Vec_yyyyqq[2:(length(Vec_yyyyqq)-1)]

# count # of qtrs
n_qtr <- length(Vec_yyyyqq)

# final step in preparation
df.result <- data.original %>%
  select(Date, yyyyqq) %>%
  mutate(Close_pd_L2E  = NA,
         Close_pd_chi  = NA,
         Close_pd_soft = NA,
         Close_pd_hard = NA,
         Close_pd_bye  = NA
         )

#### 2. main ####================================================================

q_start = 13
for (q in c(q_start:n_qtr)) {
#### 2.1. Preparation for quarter training: loop q ####
# Isolate the quarter and year in question
  loop.qtr <- Vec_yyyyqq[q]

# Isolate data for the  quarter and year in question
```

```
  data.q <- data.original %>%
    filter(yyyyqq == loop.qtr) %>%
    mutate(index_day = 1:n())

# Number of rows in the Quarter Data Frame.
  loop.nday <- nrow(data.q)

# (747 x 28)
  data.qtrain  <- data.original %>% filter(is.element(yyyyqq,
                                                      Vec_yyyyqq[(q-12):(q-1)]))

#(125 x 28)
  data.qretrain <- data.original %>% filter(is.element(yyyyqq,
                                                       Vec_yyyyqq[(q-1):q]) )

#### 2.2. Denoise for quarter training: loop q ###################################
  df.qtrain.L2E <- data.qtrain
  df.qtrain.chi <- data.qtrain
  df.qtrain.soft <- data.qtrain
  df.qtrain.hard <- data.qtrain
  df.qtrain.byes <- data.qtrain

#### 2.2.1. L2E Threshold ########################################################
  temp.open  <- CoFESWave::WaveL2E(data.qtrain$ropen,  base_plot = FALSE)
  temp.high  <- CoFESWave::WaveL2E(data.qtrain$rhigh,  base_plot = FALSE)
  temp.low   <- CoFESWave::WaveL2E(data.qtrain$rlow,   base_plot = FALSE)
  temp.close <- CoFESWave::WaveL2E(data.qtrain$rclose, base_plot = FALSE)
  temp.RSI   <- data.qtrain$rRSI
  temp.MACD  <- data.qtrain$rMACD
  temp.SIG   <- data.qtrain$rSIG
  temp.FS    <- data.qtrain$rFS
  temp.SS    <- data.qtrain$rSS
  temp.S     <- data.qtrain$rS
  temp.ULT   <- data.qtrain$rULT
  temp.Vol   <- data.qtrain$rVol
```

```
  df.qtrain.L2E$ropen  <- temp.open$recon_L2E$series$x.r
  df.qtrain.L2E$rhigh  <- temp.high$recon_L2E$series$x.r
  df.qtrain.L2E$rlow   <- temp.low$recon_L2E$series$x.r
  df.qtrain.L2E$rclose <- temp.close$recon_L2E$series$x.r
  df.qtrain.L2E$rRSI   <- temp.RSI
  df.qtrain.L2E$rMACD  <- temp.MACD
  df.qtrain.L2E$rSIG   <- temp.SIG
  df.qtrain.L2E$rFS    <- temp.FS
  df.qtrain.L2E$rSS    <- temp.SS
  df.qtrain.L2E$rS     <- temp.S
  df.qtrain.L2E$rULT   <- temp.ULT
  df.qtrain.L2E$rVol   <- temp.Vol

#### 2.2.2. L2E Chi-Squared Threshold#############################
  df.qtrain.chi$ropen  <- temp.open$recon_Chi_square$series$x.r
  df.qtrain.chi$rhigh  <- temp.high$recon_Chi_square$series$x.r
  df.qtrain.chi$rlow   <- temp.low$recon_Chi_square$series$x.r
  df.qtrain.chi$rclose <- temp.close$recon_Chi_square$series$x.r
  df.qtrain.chi$rRSI   <- temp.RSI
  df.qtrain.chi$rMACD  <- temp.MACD
  df.qtrain.chi$rSIG   <- temp.SIG
  df.qtrain.chi$rFS    <- temp.FS
  df.qtrain.chi$rSS    <- temp.SS
  df.qtrain.chi$rS     <- temp.S
  df.qtrain.chi$rULT   <- temp.ULT
  df.qtrain.chi$rVol   <- temp.Vol

#### 2.2.3. Universal Hard Thershold ############################################
  df.qtrain.hard$ropen <- wavShrink(data.qtrain$ropen,
                                    wavelet="s8",
                                    shrink.fun="hard",
                                    thresh.fun="universal",
                                    threshold=NULL,
                                    thresh.scale=1,
```

```
                                    xform="dwt",
                                    noise.variance=-1.0,
                                    reflect=TRUE)
df.qtrain.hard$rhigh <- wavShrink(data.qtrain$rhigh,
                                    wavelet="s8",
                                    shrink.fun="hard",
                                    thresh.fun="universal",
                                    threshold=NULL,
                                    thresh.scale=1,
                                    xform="dwt",
                                    noise.variance=-1.0,
                                    reflect=TRUE)
df.qtrain.hard$rlow <-  wavShrink(data.qtrain$rlow,
                                    wavelet="s8",
                                    shrink.fun="hard",
                                    thresh.fun="universal",
                                    threshold=NULL,
                                    thresh.scale=1,
                                    xform="dwt",
                                    noise.variance=-1.0,
                                    reflect=TRUE)
df.qtrain.hard$rclose <- wavShrink(data.qtrain$rclose, wavelet="s8",
shrink.fun="hard", thresh.fun="universal", threshold=NULL,thresh.scale=1,
xform="dwt", noise.variance=-1.0, reflect=TRUE)

df.qtrain.hard$rRSI  <- data.qtrain$rRSI
df.qtrain.hard$rMACD <- data.qtrain$rMACD
df.qtrain.hard$rSIG  <- data.qtrain$rSIG
df.qtrain.hard$rFS   <- data.qtrain$rFS
df.qtrain.hard$rSS   <- data.qtrain$rSS
df.qtrain.hard$rS    <- data.qtrain$rS
df.qtrain.hard$rULT  <- data.qtrain$rULT
df.qtrain.hard$rVol  <- data.qtrain$rVol
```

```
#### 2.2.4. Universal Soft Threshold ##########################################
df.qtrain.soft$ropen <- wavShrink(data.qtrain$ropen, wavelet="s8",
shrink.fun="soft",thresh.fun="universal", threshold=NULL,thresh.scale=1,
xform="dwt", noise.variance=-1.0, reflect=TRUE)
df.qtrain.soft$rhigh <- wavShrink(data.qtrain$rhigh, wavelet="s8",
shrink.fun="soft", thresh.fun="universal", threshold=NULL,
thresh.scale=1,xform="dwt", noise.variance=-1.0, reflect=TRUE)
df.qtrain.soft$rlow <-  wavShrink(data.qtrain$rlow, wavelet="s8",
shrink.fun="soft",thresh.fun="universal", threshold=NULL,
thresh.scale=1, xform="dwt", noise.variance=-1.0, reflect=TRUE)
df.qtrain.soft$rclose <- wavShrink(data.qtrain$rclose, wavelet="s8",
shrink.fun="soft", thresh.fun="universal", threshold=NULL,thresh.scale=1,
xform="dwt", noise.variance=-1.0, reflect=TRUE)

df.qtrain.soft$rRSI  <- data.qtrain$rRSI
df.qtrain.soft$rMACD <- data.qtrain$rMACD
df.qtrain.soft$rSIG  <- data.qtrain$rSIG
df.qtrain.soft$rFS   <- data.qtrain$rFS
df.qtrain.soft$rSS   <- data.qtrain$rSS
df.qtrain.soft$rS    <- data.qtrain$rS
df.qtrain.soft$rULT  <- data.qtrain$rULT
df.qtrain.soft$rVol  <- data.qtrain$rVol

#### 2.2.5. Emperical Bayesian Threshold   ####################################
#Wavelet Transform
a =1024-length(data.qtrain$ropen)
b=1+a
z = zeros(1024-length(data.qtrain$ropen))
bayes.dwto <- dwt( c(z,data.qtrain$ropen),  wf="la8")
#Emperical Bayesian Threshold
bayes_impo <- ebayesthresh.wavelet(bayes.dwto)
#nverse Transform
df.qtrain.byes$ropen <- idwt(bayes_impo)[b:1024]

#Wavelet Transform
```

```
  bayes.dwth <- dwt(c(z,data.qtrain$rhigh),  wf="la8")
  #Emperical Bayesian Threshold
  bayes_imph <- ebayesthresh.wavelet(bayes.dwth)
  #nverse Transform
  df.qtrain.byes$rhigh <- idwt(bayes_imph)[b:1024]

  #Wavelet Transform
  bayes.dwtl <- dwt(c(z,data.qtrain$rlow),  wf="la8")
  #Emperical Bayesian Threshold
  bayes_impl <- ebayesthresh.wavelet(bayes.dwtl)
  #nverse Transform
  df.qtrain.byes$rlow <- idwt(bayes_impl)[b:1024]

  #Wavelet Transform
  bayes.dwtcl <- dwt(c(z,data.qtrain$rclose),  wf="la8")
  #Emperical Bayesian Threshold
  bayes_impcl <- ebayesthresh.wavelet(bayes.dwtcl)
  #nverse Transform
  df.qtrain.byes$rclose <- idwt(bayes_impcl)[b:1024]


  df.qtrain.byes$rRSI <- data.qtrain$rRSI
  df.qtrain.byes$rMACD <- data.qtrain$rMACD
  df.qtrain.byes$rSIG <- data.qtrain$rSIG
  df.qtrain.byes$rFS <- data.qtrain$rFS
  df.qtrain.byes$rSS <- data.qtrain$rSS
  df.qtrain.byes$rS <- data.qtrain$rS
  df.qtrain.byes$rULT <- data.qtrain$rULT
  df.qtrain.byes$rVol <- data.qtrain$rVol

#############################################################################
#### 2.3. training ####

#### 2.3.1. Rearrangement before training ####
  # Seperate Target and Exogenous Variables:
```

```
  df.target.L2E <- df.qtrain.L2E %>% select(rclose)
  df.input.L2E  <- df.qtrain.L2E %>% select(ropen, rhigh, rlow, rRSI,
  rMACD, rSIG, rFS, rSS, rS, rULT, rVol)

  df.target.chi <- df.qtrain.chi %>% select(rclose)
  df.input.chi  <- df.qtrain.chi %>% select(ropen, rhigh, rlow, rRSI,
  rMACD, rSIG, rFS, rSS, rS, rULT, rVol)

  df.target.hard <- df.qtrain.hard %>% select(rclose)
  df.input.hard  <- df.qtrain.hard %>% select(ropen, rhigh, rlow, rRSI,
  rMACD, rSIG, rFS, rSS, rS, rULT, rVol)

  df.target.soft <- df.qtrain.soft %>% select(rclose)
  df.input.soft  <- df.qtrain.soft %>% select(ropen, rhigh, rlow, rRSI,
  rMACD, rSIG, rFS, rSS, rS, rULT, rVol)

  df.target.byes <- df.qtrain.byes %>% select(rclose)
  df.input.byes  <- df.qtrain.byes %>% select(ropen, rhigh, rlow,
  rRSI, rMACD, rSIG, rFS, rSS, rS, rULT, rVol)

  # Create  Matrix of NA's (61 x 10)
  matrix.rclose.pd.L2E  <- matrix(NA, nrow = loop.nday, ncol = n_rebuild)
  matrix.rclose.pd.chi  <- matrix(NA, nrow = loop.nday, ncol = n_rebuild)
  matrix.rclose.pd.hard <- matrix(NA, nrow = loop.nday, ncol = n_rebuild)
  matrix.rclose.pd.soft <- matrix(NA, nrow = loop.nday, ncol = n_rebuild)
  matrix.rclose.pd.byes <- matrix(NA, nrow = loop.nday, ncol = n_rebuild)
  ##############################################################################
  # From 1 to 10
  for (i in c(1:n_rebuild)) {
  # for (i in c(1:1)) {
    #### 2.3.2. training for quarter training: loop q ####
    for (ii in c(1:repeat_net)) {
      # Set variables
setVariable(matlab, ii = ii, input_L2E  = as.matrix(df.input.L2E),
input_chi  = as.matrix(df.input.chi), input_hard = as.matrix(df.input.hard),
```

```
input_soft = as.matrix(df.input.soft),input_byes = as.matrix(df.input.byes),
target_L2E  = as.matrix(df.target.L2E),target_chi  = as.matrix(df.target.chi),
target_hard = as.matrix(df.target.hard),target_soft = as.matrix(df.target.soft),
target_byes = as.matrix(df.target.byes))

## train
temp.expression = paste0("[net_L2E_", ii, ",performance] = WNN_network(input_L2E,
target_L2E(:, 1), delay, hid, vratio, tratio);","[net_chi_",
ii, ",performance] = WNN_network(input_chi,target_chi(:, 1),
delay, hid, vratio, tratio);","[net_hard_", ii,
",performance] = WNN_network(input_hard, target_hard(:, 1),
delay, hid, vratio, tratio);","[net_soft_", ii, ",
performance] = WNN_network(input_soft, target_soft(:, 1), delay,
hid, vratio, tratio);","[net_byes_", ii, ",
performance] = WNN_network(input_byes, target_byes(:, 1),
delay, hid, vratio, tratio);"
)

evaluate(matlab, temp.expression)
}

################################################################################
    #### 2.4. Preparation for day training: loop d ####
    # for 1 to 61 (vary according to quarters)
    for (d in c(1:loop.nday)) {
      ind_d <- which(data.qretrain$Date == data.q$Date[d]) -1

      ## adjust for the last day (the day we are forecasting)
      data.qretrain.d <- data.qretrain %>%
        dplyr::slice(c((ind_d-L):ind_d)+1) %>%
        mutate(ind_retrain = 1:n(),
               max_ind = max(ind_retrain, na.rm = TRUE),
               signal_last = (ind_retrain == (max_ind-1)),
               signal_pd = (ind_retrain == max_ind)) %>%
        mutate(rhigh  = replace(rhigh,  signal_pd,   ropen[signal_pd  ]),
```

```
                rlow   = replace(rlow,   signal_pd,   ropen[signal_pd  ]),
                rclose = replace(rclose, signal_pd,   ropen[signal_pd  ]),
                rRSI   = replace(rRSI,   signal_pd,   rRSI[ signal_last]),
                rMACD  = replace(rMACD,  signal_pd,   rMACD[signal_last]),
                rSIG   = replace(rSIG,   signal_pd,   rSIG[ signal_last]),
                rFS    = replace(rFS,    signal_pd,   rFS[  signal_last]),
                rSS    = replace(rSS,    signal_pd,   rSS[  signal_last]),
                rS     = replace(rS,     signal_pd,   rS[   signal_last]),
                rULT   = replace(rULT,   signal_pd,   rULT[ signal_last]),
                rVol   = replace(rVol,   signal_pd,   rVol[ signal_last])
                )
nday_retrain <- nrow(data.qretrain.d)

#### 2.5. Denoise for day training: loop d ####
# we can add more denoising variables later
      df.qretrain.L2E <- data.qretrain.d
      df.qretrain.chi <- data.qretrain.d
      df.qretrain.soft <- data.qretrain.d
      df.qretrain.hard <- data.qretrain.d
      df.qretrain.byes <- data.qretrain.d

#### 2.5.1. L2E ######################################################################
      temp.open  <- CoFESWave::WaveL2E(data.qretrain.d$ropen, base_plot = FALSE)
      temp.high  <- CoFESWave::WaveL2E(data.qretrain.d$rhigh, base_plot = FALSE)
      temp.low   <- CoFESWave::WaveL2E(data.qretrain.d$rlow, base_plot = FALSE)
      temp.close <- CoFESWave::WaveL2E(data.qretrain.d$rclose, base_plot = FALSE)
      temp.RSI   <- data.qretrain.d$rRSI
      temp.MACD  <- data.qretrain.d$rMACD
      temp.SIG   <- data.qretrain.d$rSIG
      temp.FS    <- data.qretrain.d$rFS
      temp.SS    <- data.qretrain.d$rSS
      temp.S     <- data.qretrain.d$rS
      temp.ULT   <- data.qretrain.d$rULT
      temp.Vol   <- data.qretrain.d$rVol
```

```
df.qretrain.L2E$ropen  <- temp.open$recon_L2E$series$x.r
df.qretrain.L2E$rhigh  <- temp.high$recon_L2E$series$x.r
df.qretrain.L2E$rlow   <- temp.low$recon_L2E$series$x.r
df.qretrain.L2E$rclose <- temp.close$recon_L2E$series$x.r
df.qretrain.L2E$rRSI   <- temp.RSI
df.qretrain.L2E$rMACD  <- temp.MACD
df.qretrain.L2E$rSIG   <- temp.SIG
df.qretrain.L2E$rFS    <- temp.FS
df.qretrain.L2E$rSS    <- temp.SS
df.qretrain.L2E$rS     <- temp.S
df.qretrain.L2E$rULT   <- temp.ULT
df.qretrain.L2E$rVol   <- temp.Vol

#### 2.5.2. L2E Chi Squared Threshold ##################################
df.qretrain.chi$ropen  <- temp.open$recon_Chi_square$series$x.r
df.qretrain.chi$rhigh  <- temp.high$recon_Chi_square$series$x.r
df.qretrain.chi$rlow   <- temp.low$recon_Chi_square$series$x.r
df.qretrain.chi$rclose <- temp.close$recon_Chi_square$series$x.r
df.qretrain.chi$rRSI   <- temp.RSI
df.qretrain.chi$rMACD  <- temp.MACD
df.qretrain.chi$rSIG   <- temp.SIG
df.qretrain.chi$rFS    <- temp.FS
df.qretrain.chi$rSS    <- temp.SS
df.qretrain.chi$rS     <- temp.S
df.qretrain.chi$rULT   <- temp.ULT
df.qretrain.chi$rVol   <- temp.Vol

#### 2.5.3. Universal Hard Thershold ###################################
df.qretrain.hard$ropen <- wavShrink(data.qretrain.d$ropen,
wavelet="s8", shrink.fun="hard", thresh.fun="universal",
threshold=NULL,thresh.scale=1,xform="dwt", noise.variance=-1.0,
reflect=TRUE)
df.qretrain.hard$rhigh <- wavShrink(data.qretrain.d$rhigh,
wavelet="s8", shrink.fun="hard",thresh.fun="universal",
threshold=NULL,thresh.scale=1, xform="dwt", noise.variance=-1.0,
```

```
reflect=TRUE)
df.qretrain.hard$rlow <-  wavShrink(data.qretrain.d$rlow,
wavelet="s8", shrink.fun="hard",thresh.fun="universal",
threshold=NULL,thresh.scale=1,xform="dwt", noise.variance=-1.0,
reflect=TRUE)
df.qretrain.hard$rclose <- wavShrink(data.qretrain.d$rclose,
wavelet="s8", shrink.fun="hard", thresh.fun="universal",
threshold=NULL,thresh.scale=1,xform="dwt", noise.variance=-1.0,
reflect=TRUE)

df.qretrain.hard$rRSI  <-  data.qretrain.d$rRSI
df.qretrain.hard$rMACD <-  data.qretrain.d$rMACD
df.qretrain.hard$rSIG  <-  data.qretrain.d$rSIG
df.qretrain.hard$rFS   <-  data.qretrain.d$rFS
df.qretrain.hard$rSS   <-  data.qretrain.d$rSS
df.qretrain.hard$rS    <-  data.qretrain.d$rS
df.qretrain.hard$rULT  <-  data.qretrain.d$rULT
df.qretrain.hard$rVol <-   data.qretrain.d$rVol

#### 2.5.4. Universal Soft Threshold ####################################
df.qretrain.soft$ropen <- wavShrink(data.qretrain.d$ropen,
wavelet="s8", shrink.fun="soft", thresh.fun="universal",
threshold=NULL,thresh.scale=1, xform="dwt",
noise.variance=-1.0, reflect=TRUE)
df.qretrain.soft$rhigh <- wavShrink(data.qretrain.d$rhigh,
wavelet="s8", shrink.fun="soft", thresh.fun="universal",
threshold=NULL,thresh.scale=1, xform="dwt",
noise.variance=-1.0, reflect=TRUE)
df.qretrain.soft$rlow <-  wavShrink(data.qretrain.d$rlow,
wavelet="s8", shrink.fun="soft", thresh.fun="universal",
threshold=NULL,thresh.scale=1, xform="dwt",
noise.variance=-1.0, reflect=TRUE)
df.qretrain.soft$rclose <- wavShrink(data.qretrain.d$rclose,
wavelet="s8", shrink.fun="soft",thresh.fun="universal",
threshold=NULL,thresh.scale=1,xform="dwt",
```

```
noise.variance=-1.0, reflect=TRUE)

df.qretrain.soft$rRSI  <-  data.qretrain.d$rRSI
df.qretrain.soft$rMACD <-  data.qretrain.d$rMACD
df.qretrain.soft$rSIG  <-  data.qretrain.d$rSIG
df.qretrain.soft$rFS   <-  data.qretrain.d$rFS
df.qretrain.soft$rSS   <-  data.qretrain.d$rSS
df.qretrain.soft$rS    <-  data.qretrain.d$rS
df.qretrain.soft$rULT  <-  data.qretrain.d$rULT
df.qretrain.soft$rVol <-   data.qretrain.d$rVol


#### 2.5.5. Emperical Bayesian Threshold ##########################
a= 64 - length(data.qretrain.d$ropen)
b= a + 1
z = zeros(a)

#Wavelet Transform
bayes.dwto <- dwt(c(z,data.qretrain.d$ropen),  wf="la8")
#Emperical Bayesian Threshold
bayes_impo <- ebayesthresh.wavelet(bayes.dwto)
#nverse Transform
df.qretrain.byes$ropen <- idwt(bayes_impo)[b:64]

#Wavelet Transform
bayes.dwth <- dwt(c(z,data.qretrain.d$rhigh),  wf="la8")
#Emperical Bayesian Threshold
bayes_imph <- ebayesthresh.wavelet(bayes.dwth)
#nverse Transform
df.qretrain.byes$rhigh <- idwt(bayes_imph)[b:64]

#Wavelet Transform
bayes.dwtl <- dwt(c(z,data.qretrain.d$rlow),  wf="la8")
#Emperical Bayesian Threshold
bayes_impl <- ebayesthresh.wavelet(bayes.dwtl)
```

```
#nverse Transform
df.qretrain.byes$rlow <- idwt(bayes_impl)[b:64]

#Wavelet Transform
bayes.dwtcl <- dwt(c(z,data.qretrain.d$rclose),  wf="la8")
#Emperical Bayesian Threshold
bayes_impcl <- ebayesthresh.wavelet(bayes.dwtcl)
#nverse Transform
df.qretrain.byes$rclose <- idwt(bayes_impcl)[b:64]

df.qretrain.byes$rRSI <- data.qretrain.d$rRSI
df.qretrain.byes$rMACD <- data.qretrain.d$rMACD
df.qretrain.byes$rSIG <- data.qretrain.d$rSIG
df.qretrain.byes$rFS <- data.qretrain.d$rFS
df.qretrain.byes$rSS <- data.qretrain.d$rSS
df.qretrain.byes$rS <- data.qretrain.d$rS
df.qretrain.byes$rULT <- data.qretrain.d$rULT
df.qretrain.byes$rVol <- data.qretrain.d$rVol
############################################################
#### 2.6. training ####
#### 2.6.1. Rearrangement before training ####
df.pd.target.L2E <- df.qretrain.L2E %>% select(rclose)
df.pd.input.L2E  <- df.qretrain.L2E %>% select(ropen, rhigh, rlow,
rRSI, rMACD, rSIG, rFS, rSS, rS, rULT, rVol)

df.pd.target.chi <- df.qretrain.chi %>% select(rclose)
df.pd.input.chi  <- df.qretrain.chi %>% select(ropen, rhigh,
rlow, rRSI, rMACD,rSIG, rFS, rSS, rS, rULT, rVol)

df.pd.target.hard <- df.qretrain.hard %>% select(rclose)
df.pd.input.hard  <- df.qretrain.hard %>% select(ropen, rhigh,
rlow, rRSI, rMACD,rSIG, rFS, rSS, rS, rULT, rVol)

df.pd.target.soft <- df.qretrain.soft %>% select(rclose)
df.pd.input.soft  <- df.qretrain.soft %>% select(ropen, rhigh,
```

```
      rlow, rRSI, rMACD, rSIG, rFS, rSS, rS, rULT, rVol)

      df.pd.target.byes <- df.qretrain.byes %>% select(rclose)
      df.pd.input.byes  <- df.qretrain.byes %>% select(ropen, rhigh,
      rlow, rRSI, rMACD,rSIG, rFS, rSS, rS, rULT, rVol)

      ## retrain result matrix
      Vec_i_d_k_pd_L2E  = rep(0, repeat_net)
      Vec_i_d_k_pd_chi  = rep(0, repeat_net)
      Vec_i_d_k_pd_hard = rep(0, repeat_net)
      Vec_i_d_k_pd_soft = rep(0, repeat_net)
      Vec_i_d_k_pd_byes = rep(0, repeat_net)

#### 2.6.2. training for quarter training: loop q ####
      for (k in c(1:repeat_net)) {
        setVariable(matlab,
                    input_retrain_L2E  = as.matrix(df.pd.input.L2E),
                    input_retrain_chi  = as.matrix(df.pd.input.chi),
                    input_retrain_hard = as.matrix(df.pd.input.hard),
                    input_retrain_soft = as.matrix(df.pd.input.soft),
                    input_retrain_byes = as.matrix(df.pd.input.byes),
                    target_retrain_L2E  = as.matrix(df.pd.target.L2E),
                    target_retrain_chi  = as.matrix(df.pd.target.chi),
                    target_retrain_hard = as.matrix(df.pd.target.hard),
                    target_retrain_soft = as.matrix(df.pd.target.soft),
                    target_retrain_byes = as.matrix(df.pd.target.byes))
#########################################################################
temp.expression <- paste0("xtc␣=␣zeros(1,5);","[xtc(1),␣~,␣~,␣~]
=␣WNN_network_retrain(input_retrain_L2E,
target_retrain_L2E(:,1),␣net_L2E_", k, ");",
"[xtc(2),␣~,␣~,␣~]␣=␣WNN_network_retrain(input_retrain_chi,
target_retrain_chi(:,1),␣net_chi_", k, ");",
"[xtc(3),␣~,␣~,␣~]␣=␣WNN_network_retrain(input_retrain_hard,
target_retrain_hard(:,1),␣net_hard_", k, ");",
"[xtc(4),␣~,␣~,␣~]␣=␣WNN_network_retrain(input_retrain_soft,
```

```
target_retrain_soft(:,1),␣net_soft_", k, ");",
"[xtc(5),␣~,␣~,␣~]␣=␣WNN_network_retrain(input_retrain_byes,
target_retrain_byes(:,1),␣net_byes_", k, ");"
)
evaluate(matlab, temp.expression)
pd.close <- getVariable(matlab, "xtc")
Vec_i_d_k_pd_L2E[k]  <- pd.close$xtc[1]
Vec_i_d_k_pd_chi[k]  <- pd.close$xtc[2]
Vec_i_d_k_pd_hard[k] <- pd.close$xtc[3]
Vec_i_d_k_pd_soft[k] <- pd.close$xtc[4]
Vec_i_d_k_pd_byes[k] <- pd.close$xtc[5]
}
matrix.rclose.pd.L2E[ d,i] = mean(Vec_i_d_k_pd_L2E)
matrix.rclose.pd.chi[ d,i] = mean(Vec_i_d_k_pd_chi)
matrix.rclose.pd.hard[d,i] = mean(Vec_i_d_k_pd_hard)
matrix.rclose.pd.soft[d,i] = mean(Vec_i_d_k_pd_soft)
matrix.rclose.pd.byes[d,i] = mean(Vec_i_d_k_pd_byes)
    }
} # loop by n_rebuild

 #### 2.7. rearrangement of prediction ####
 ## calculate
  Vec_pd_L2E  = rowMeans(matrix.rclose.pd.L2E)
  Vec_pd_chi  = rowMeans(matrix.rclose.pd.chi)
  Vec_pd_hard = rowMeans(matrix.rclose.pd.hard)
  Vec_pd_soft = rowMeans(matrix.rclose.pd.soft)
  Vec_pd_byes = rowMeans(matrix.rclose.pd.byes)

  ## put them in the result dataframe
  ind_date <- df.result$yyyyqq == loop.qtr
  df.result$Close_pd_L2E[ ind_date] = Vec_pd_L2E
  df.result$Close_pd_chi[ ind_date] = Vec_pd_chi
  df.result$Close_pd_hard[ind_date] = Vec_pd_hard
  df.result$Close_pd_soft[ind_date] = Vec_pd_soft
  df.result$Close_pd_byes[ind_date] = Vec_pd_byes
```

```
  #### 2.8. save ####
  df.result.save <- df.result %>%
    mutate(q_now = q)
  save(df.result.save, file = paste0(chr_secrity_name,
  delay, hid, "RResult_save.Rdata"))
} # loop by quarter
close(matlab)
```

A.0.2 R-Code: Trading Algorithm Back Test

```
library(tictoc)
library(tidyverse)
library(stringr)
library(ggplot2)
#### 0. Load data ####=====================================================
## load Data as data.raw
ch1 <-c( 'sp4002_rawdata.csv','sp4002_rawdata.csv','dowjones2_rawdata.csv',
         'dowjones2_rawdata.csv','sp5002_rawdata.csv','sp4002_rawdata.csv',
         'dowjones2_rawdata.csv', 'nasdaq2_rawdata.csv')
nam1 <-c( 'spCHI4002','spL2E4002','dowjonesCHI2','dowjonesL2E2','sp5002',
          'sp4002','dowjones2', 'nasdaq2')
res1 <-c('sp4002CHIResult_save.Rdata','sp4002L2EResult_save.Rdata',
         'dowjones2CHIResult_save.Rdata','dowjones2L2EResult_save.Rdata',
         'sp5002Result_save.Rdata','sp4002Result_save.Rdata',
         'dowjones2Result_save.Rdata','nasdaq2Result_save.Rdata')
i=8
data.raw <- read.csv2(file = ch1[i])
chr_secrity_name <- nam1[i]
result <-res1[i]
data.raw<-data.raw %>%
    mutate(Date_raw = stringr::str_pad(Date, 9 , side = "left", pad = 0),
           Date_str = stringr::str_pad(Date_raw, 10 , side = "left", pad = 2),
           Date = as.Date(Date_str))
```

```
## load result
load(result)

data.forecast <- df.result.save %>%
  mutate(Date_raw = stringr::str_pad(Date, 9 , side = "left", pad = 0),
         Date_str = stringr::str_pad(Date_raw, 10 , side = "left", pad = 2),
         Date = as.Date(Date_str))%>%
  select(Date,
         Close_pd_L2E, Close_pd_chi, Close_pd_hard,
         Close_pd_soft, Close_pd_byes)
rm(df.result.save)

#### 1. Process data ####
data.raw <-  data.raw %>% select(-X)

#### 1.1. Variable Name Adjustment ####
# Get Column Names
vec_colname <- colnames(data.raw)

# Set First Column Name to Date
vec_colname[1] <- "Date"
colnames(data.raw) <- vec_colname

#### 1.3. Return preparation ####
data.backtest <- data.raw %>%

  # Date
  mutate(Date = as.Date(Date,format = "%m/%d/%Y"),
         year = lubridate::year(Date)) %>%
  select(Date, year, close) %>%

  # returns
  mutate(return_raw = close / lag(close,  1) - 1) %>%

  # merge with predictions
```

```
  left_join(data.forecast, by = "Date")

#### 2. backtest ####
#### 2.1. longshort and longonly signals ####

## data: base
data.base <- data.backtest %>%

# signal to buy or sell at each day
  mutate(signal_long_L2E  = Close_pd_L2E  > 0,
         signal_long_chi  = Close_pd_chi  > 0,
         signal_long_hard = Close_pd_hard > 0,
         signal_long_soft = Close_pd_soft > 0,
         signal_long_byes = Close_pd_byes > 0) %>%

# drop NA forecasts
  drop_na(signal_long_L2E,  signal_long_chi,
          signal_long_hard, signal_long_soft, signal_long_byes) %>%

# longshort and longonly returns
  mutate(sign_return = sign(return_raw),
         multiplier_longonly_L2E  = as.numeric(signal_long_L2E),
         multiplier_longonly_chi  = as.numeric(signal_long_chi),
         multiplier_longonly_hard = as.numeric(signal_long_hard),
         multiplier_longonly_soft = as.numeric(signal_long_soft),
         multiplier_longonly_byes = as.numeric(signal_long_byes),
         multiplier_longshort_L2E  = replace(multiplier_longonly_L2E,
                                             multiplier_longonly_L2E  < 1,
                                             -1),
         multiplier_longshort_chi  = replace(multiplier_longonly_chi,
                                             multiplier_longonly_chi  < 1,
                                             -1),
         multiplier_longshort_hard = replace(multiplier_longonly_hard,
                                             multiplier_longonly_hard < 1,
                                             -1),
```

```
        multiplier_longshort_soft = replace(multiplier_longonly_soft,
                                            multiplier_longonly_soft < 1,
                                            -1),
        multiplier_longshort_byes = replace(multiplier_longonly_byes,
                                            multiplier_longonly_byes < 1,
                                            -1)) %>%

# return calculation
  mutate(return_longshort_L2E  = return_raw * multiplier_longshort_L2E,
         return_longshort_chi  = return_raw * multiplier_longshort_chi,
         return_longshort_hard = return_raw * multiplier_longshort_hard,
         return_longshort_soft = return_raw * multiplier_longshort_soft,
         return_longshort_byes = return_raw * multiplier_longshort_byes,

         return_longonly_L2E  = return_raw * multiplier_longonly_L2E,
         return_longonly_chi  = return_raw * multiplier_longonly_chi,
         return_longonly_hard = return_raw * multiplier_longonly_hard,
         return_longonly_soft = return_raw * multiplier_longonly_soft,
         return_longonly_byes = return_raw * multiplier_longonly_byes)

#### 2.1. Signal analysis: accuracy ####
df.backtest.accuracy <- data.base %>%
  # prediction accuracy
  mutate(accuracy_longshort_L2E  = sign_return == multiplier_longshort_L2E,
         accuracy_longshort_chi  = sign_return == multiplier_longshort_chi,
         accuracy_longshort_hard = sign_return == multiplier_longshort_hard,
         accuracy_longshort_soft = sign_return == multiplier_longshort_soft,
         accuracy_longshort_byes = sign_return == multiplier_longshort_byes) %>%

  group_by(year) %>%
  summarise(count = n(),
            accuracy_rate_L2E = mean(accuracy_longshort_L2E),
            accuracy_rate_chi = mean(accuracy_longshort_chi),
            accuracy_rate_hard = mean(accuracy_longshort_hard),
            accuracy_rate_soft = mean(accuracy_longshort_soft),
```

```
            accuracy_rate_byes = mean(accuracy_longshort_byes)) %>%
  ungroup()

#### 2.2. Signal analysis: confusion matrix ####
data.base.confusion <- data.base %>%
  # category signal
  mutate(signal_TP_L2E = (sign_return ==  1)
  & (multiplier_longshort_L2E ==  1),
         signal_TN_L2E = (sign_return ==  1)
         & (multiplier_longshort_L2E == -1),
         signal_FP_L2E = (sign_return == -1)
         & (multiplier_longshort_L2E ==  1),
         signal_FN_L2E = (sign_return == -1)
         & (multiplier_longshort_L2E == -1),

         signal_TP_chi = (sign_return ==  1)
         & (multiplier_longshort_chi ==  1),
         signal_TN_chi = (sign_return ==  1)
         & (multiplier_longshort_chi == -1),
         signal_FP_chi = (sign_return == -1)
         & (multiplier_longshort_chi ==  1),
         signal_FN_chi = (sign_return == -1)
         & (multiplier_longshort_chi == -1),

         signal_TP_hard = (sign_return ==  1)
         & (multiplier_longshort_hard ==  1),
         signal_TN_hard = (sign_return ==  1)
         & (multiplier_longshort_hard == -1),
         signal_FP_hard = (sign_return == -1)
         & (multiplier_longshort_hard ==  1),
         signal_FN_hard = (sign_return == -1)
         & (multiplier_longshort_hard == -1),

         signal_TP_soft = (sign_return ==  1)
         & (multiplier_longshort_soft ==  1),
```

```
          signal_TN_soft = (sign_return ==  1)
          & (multiplier_longshort_soft == -1),
          signal_FP_soft = (sign_return == -1)
          & (multiplier_longshort_soft ==  1),
          signal_FN_soft = (sign_return == -1)
          & (multiplier_longshort_soft == -1),

          signal_TP_byes = (sign_return ==  1)
          & (multiplier_longshort_byes ==  1),
          signal_TN_byes = (sign_return ==  1)
          & (multiplier_longshort_byes == -1),
          signal_FP_byes = (sign_return == -1)
          & (multiplier_longshort_byes ==  1),
          signal_FN_byes = (sign_return == -1)
          & (multiplier_longshort_byes == -1))

df.backtest.confusion <- data.base.confusion %>%
  group_by(year) %>%
  summarise(TP_L2E = mean(signal_TP_L2E),
            TN_L2E = mean(signal_TN_L2E),
            FP_L2E = mean(signal_FP_L2E),
            FN_L2E = mean(signal_FN_L2E),
            TP_chi = mean(signal_TP_chi),
            TN_chi = mean(signal_TN_chi),
            FP_chi = mean(signal_FP_chi),
            FN_chi = mean(signal_FN_chi),
            TP_hard = mean(signal_TP_hard),
            TN_hard = mean(signal_TN_hard),
            FP_hard = mean(signal_FP_hard),
            FN_hard = mean(signal_FN_hard),
            TP_soft = mean(signal_TP_soft),
            TN_soft = mean(signal_TN_soft),
            FP_soft = mean(signal_FP_soft),
            FN_soft = mean(signal_FN_soft),
            TP_byes = mean(signal_TP_byes),
```

```
            TN_byes = mean(signal_TN_byes),
            FP_byes = mean(signal_FP_byes),
            FN_byes = mean(signal_FN_byes)) %>%
  ungroup()

#### 2.3. Signal analysis: precision ####
df.backtest.precision <- df.backtest.confusion %>%
  transmute(year = year,
            precision_L2E  = TP_L2E  / (TP_L2E  + FP_L2E),
            precision_chi  = TP_chi  / (TP_chi  + FP_chi),
            precision_hard = TP_hard / (TP_hard + FP_hard),
            precision_soft = TP_soft / (TP_soft + FP_soft),
            precision_byes = TP_byes / (TP_byes + FP_byes),
            recall_L2E  = TP_L2E  / (TP_L2E  + FN_L2E),
            recall_chi  = TP_chi  / (TP_chi  + FN_chi),
            recall_hard = TP_hard / (TP_hard + FN_hard),
            recall_soft = TP_soft / (TP_soft + FN_soft),
            recall_byes = TP_byes / (TP_byes + FN_byes))

#### 2.4. Return analysis: cumulative return ####
df.backtest.cumreturn <- data.base %>%
  group_by(year) %>%
  summarise(cumret_longshort_L2E  = prod(1 + return_longshort_L2E),
            cumret_longshort_chi  = prod(1 + return_longshort_chi),
            cumret_longshort_hard = prod(1 + return_longshort_hard),
            cumret_longshort_soft = prod(1 + return_longshort_soft),
            cumret_longshort_byes = prod(1 + return_longshort_byes),
            cumret_longonly_L2E  = prod(1 + return_longonly_L2E),
            cumret_longonly_chi  = prod(1 + return_longonly_chi),
            cumret_longonly_hard = prod(1 + return_longonly_hard),
            cumret_longonly_soft = prod(1 + return_longonly_soft),
            cumret_longonly_byes = prod(1 + return_longonly_byes)) %>%
  ungroup()

#### 3. save ####
```

```
save(df.backtest.accuracy,
     df.backtest.confusion,
     df.backtest.precision,
     df.backtest.cumreturn,
     file = paste0("Analysis_", chr_secrity_name, upd, ".Rdata"))
# end of this file
```

A.0.3 Figures

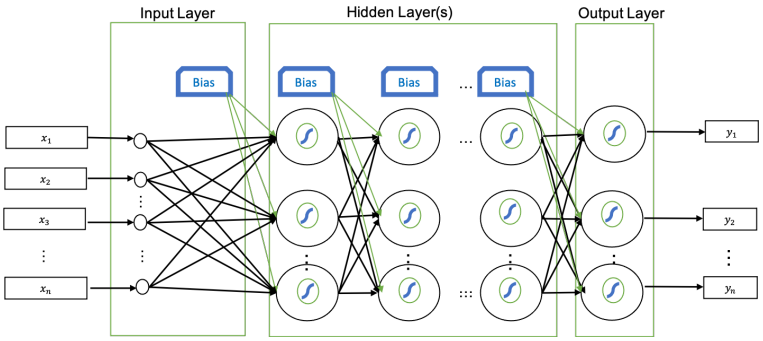


Figure A.1 : MLP architecture

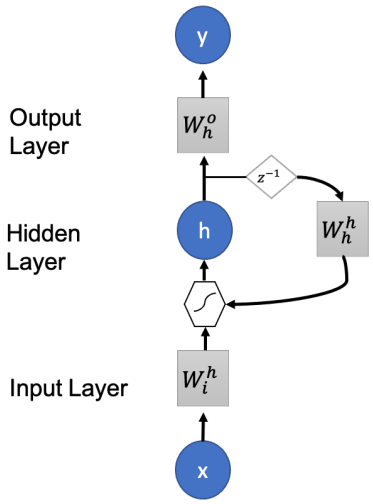


Figure A.2 : Simple RNN architecture. The circles represent input x, hidden, h, and output nodes, y, respectively. The solid squares W_i^h,W_h^h,W_h^0 are the matrices for input layer, hidden layer, and output layer weights respectively. The polygon represents the nonlinear activation function and z^{-1} is the time shift operator

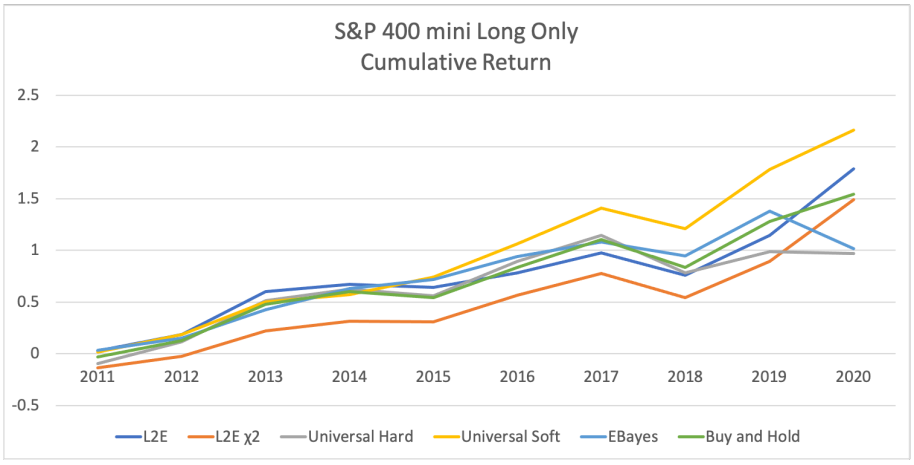


Figure A.3 : SP 400 Long Only Cumulative Return

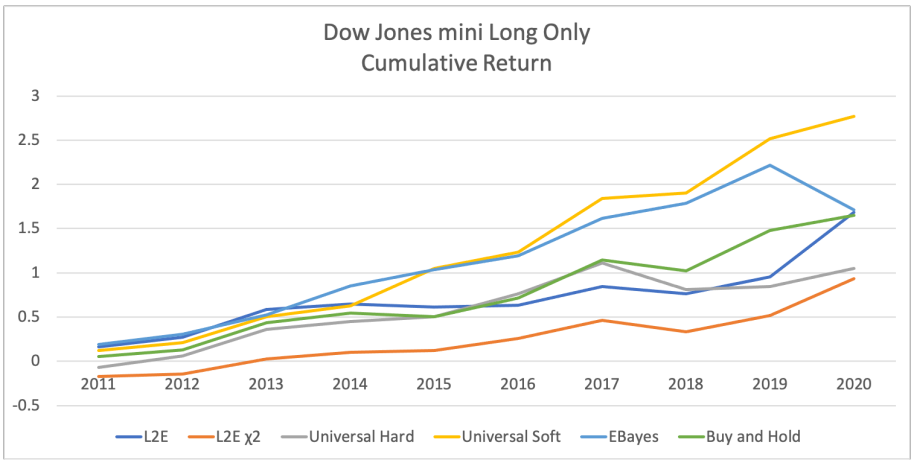


Figure A.4 : Dow Jones Long Only Cumulative Return

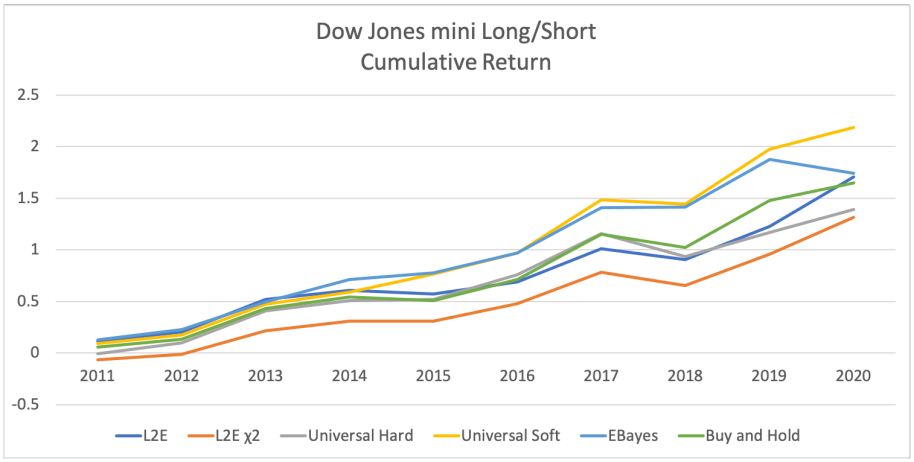


Figure A.5 : Dow Jones Long/Short Cumulative Return

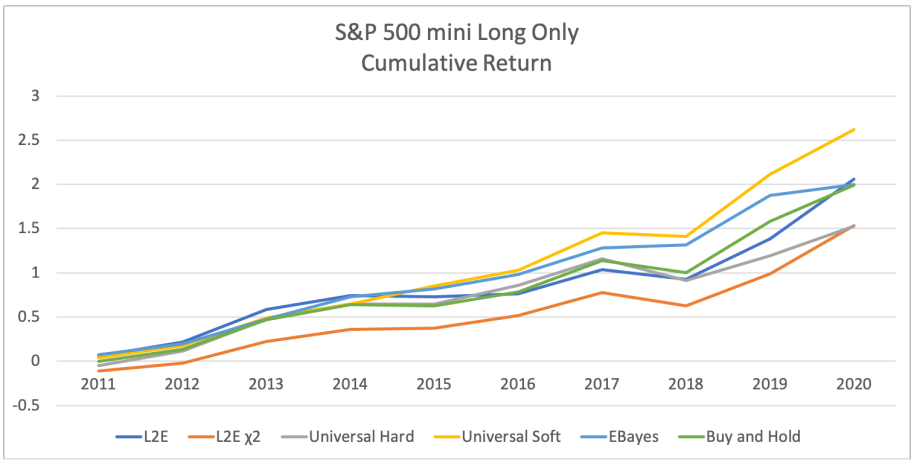


Figure A.6 : SP 500 Long Only Cumulative Return

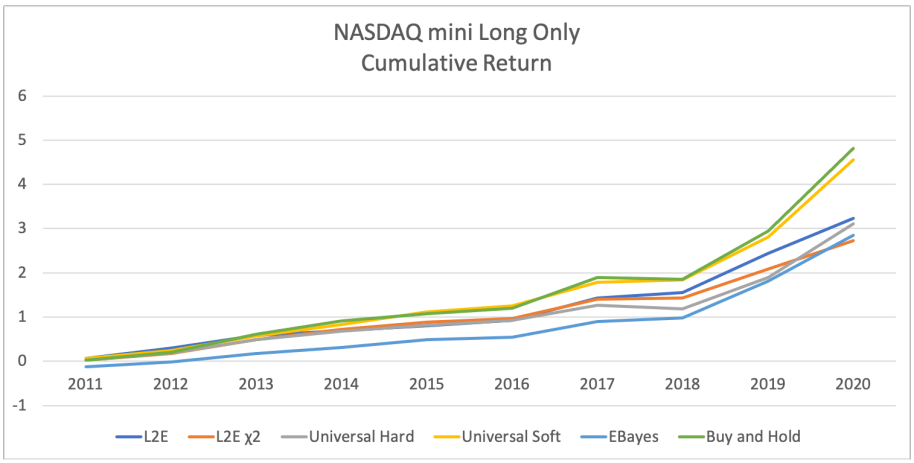


Figure A.7 : NASDAQ Long Only Cumulative Return

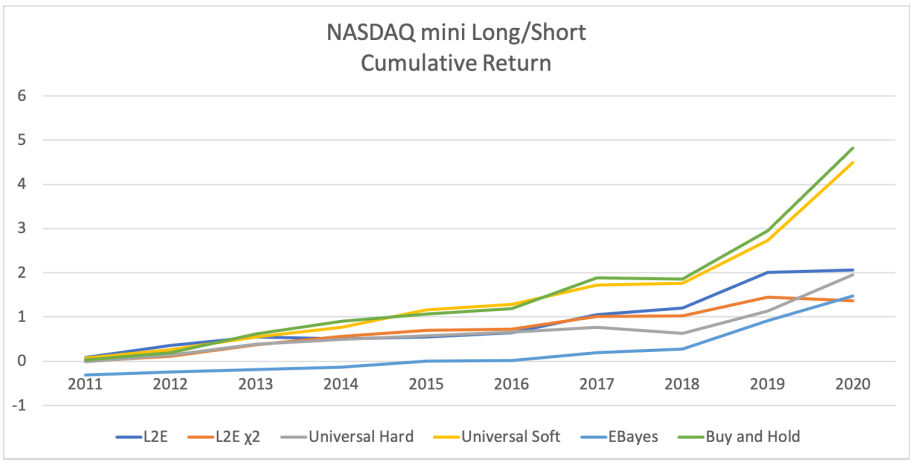


Figure A.8 : NASDAQ Long/Short Cumulative Return

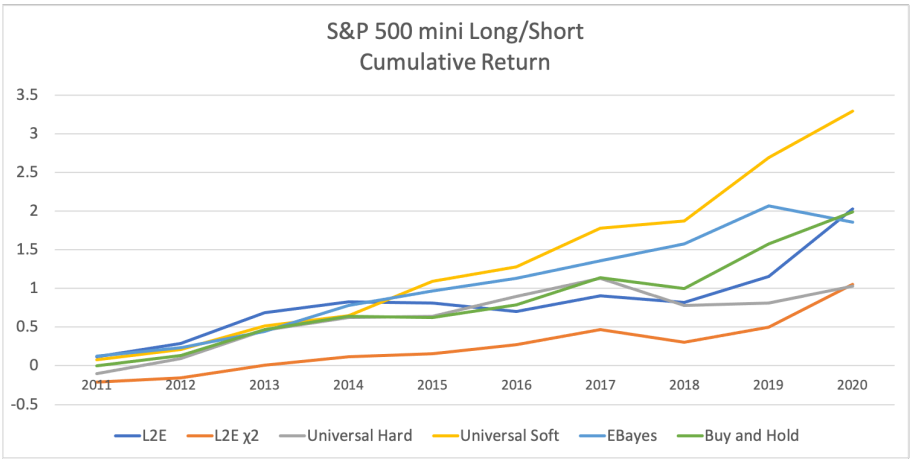


Figure A.9 : SP 500 Long/Short Cumulative Return

Appendix B

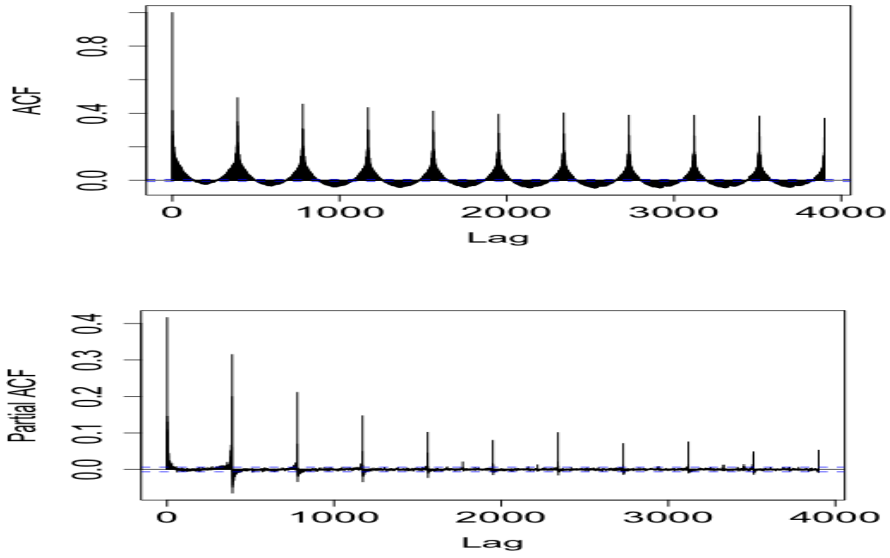


Figure B.1 : Microsoft ACF/PACF for volume

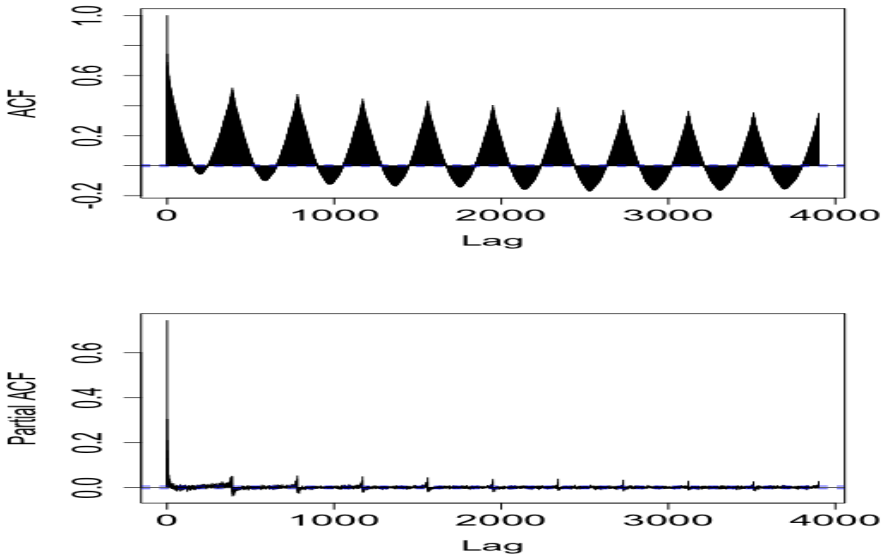


Figure B.2 : Microsoft acf (left), pacf (right), volume (top) and log volume(bottom) binned at the 1-minute intervals.

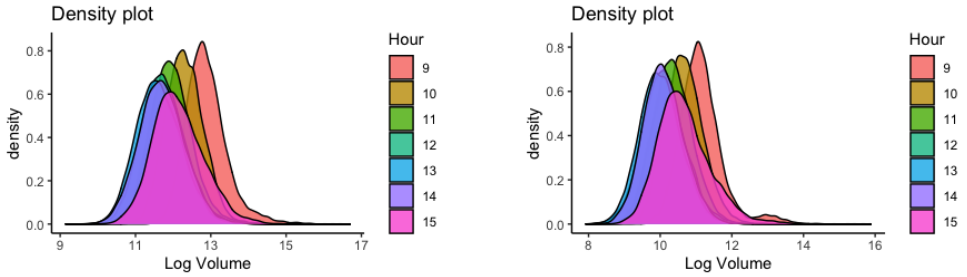


Figure B.3 : Density plots for hourly log volume and for Apple (top) and Microsoft (bottom).

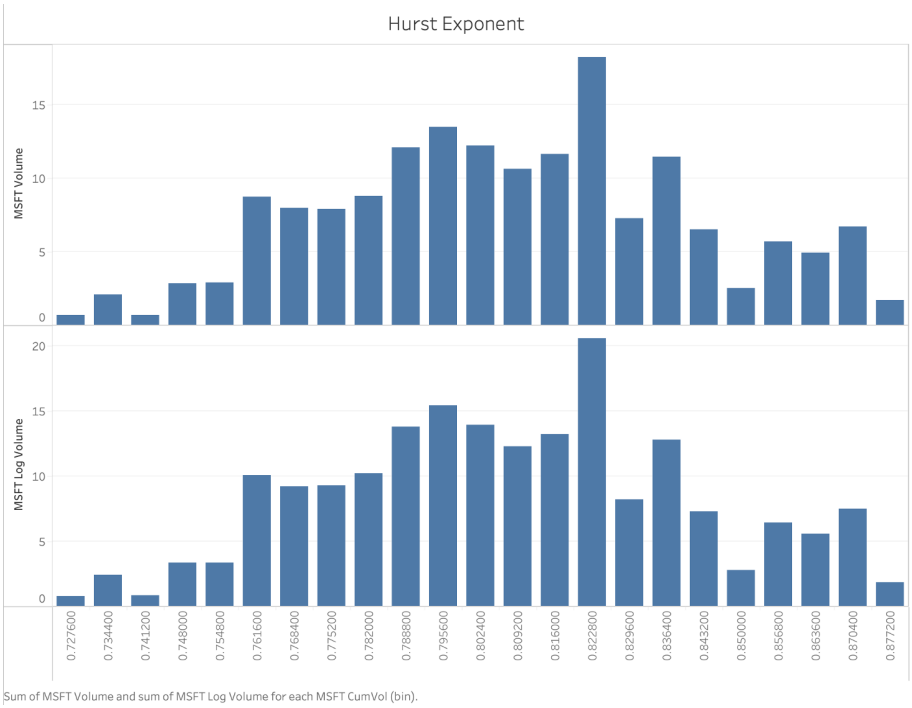


Figure B.4 : Hurst exponent distribution for Microsoft(right) using samples of size 10,000.

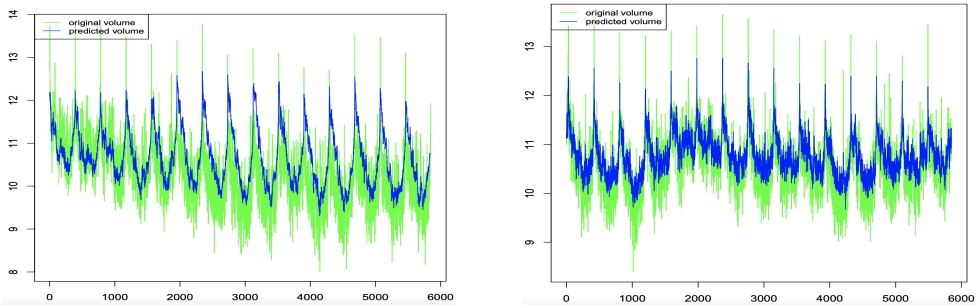


Figure B.5 : Results from the SNARX (left) and from the S-NARX-HmM (right) models' prediction of Microsoft's log intraday trading volume.

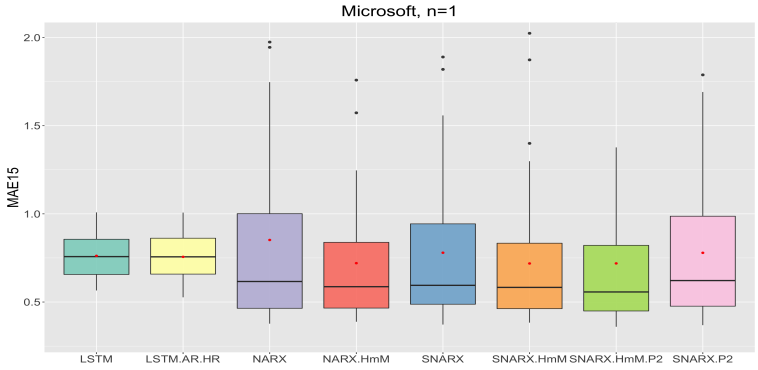


Figure B.6 : MAE15 results for Microsoft's log (vol), n = 1

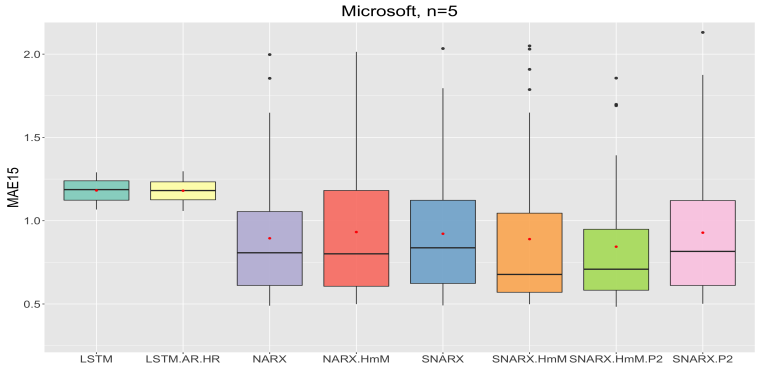


Figure B.7 : MAE15 results for Apple's log (vol), n = 5

Bibliography

[1] (2010). Is high-frequency trading inducing changes in market microstructure and dynamics?

[2] (2020). Denoising non-stationary signals by dynamic multivariate complex wavelet thresholding.

[Addison] Addison, P. S. *The Illustrated Wavelet Transform Handbook*.

[4] Allaire, J. and Chollet, F. (2022). *keras: R Interface to 'Keras'*. R package version 2.8.0.

[5] Alvim, L., dos Santos, C. N., and Milidiu, R. L. (2010). Daily volume forecasting using high frequency predictors. In *Proceedings of the 10th IASTED International Conference*, volume 674, page 248.

[6] Ardalani-Farsa, M. and Zolfaghari, S. (2010). Chaotic time series prediction with residual analysis method using hybrid elman–narx neural networks. *Neurocomputing*, 73(13):2540–2553. Pattern Recognition in Bioinformatics Advances in Neural Control.

[7] Atsalakis, G. S. and Valavanis, K. P. (2009). Surveying stock market forecasting techniques – part ii: Soft computing methods. *Expert Systems with Applications*, 36(3, Part 2):5932–5941.

[8] Bahrammirzaee, A. (2010). A comparative survey of artificial intelligence applications in finance: artificial neural networks, expert system and hybrid intelligent systems. *Neural Computing and Applications*, 19(8):1165–1195.

[9] Baltas, N. and Kosowski, R. (2013). Momentum strategies in futures markets and trend-following funds. In *Paris December 2012 Finance Meeting EUROFIDAI-AFFI Paper*.

[10] Beg, M. O., Awan, M. N., and Ali, S. S. (2019). Algorithmic machine learning for prediction of stock prices. In *FinTech as a Disruptive Technology for Financial Institutions*, pages 142–169. IGI Global.

[11] Bianchi, F. M. (2017). *Recurrent Neural Networks for Short-Term Load Forecasting An Overview and Comparative Analysis / by Filippo Maria Bianchi, Enrico Maiorino, Michael C. Kampffmeyer, Antonello Rizzi, Robert Jenssen.* SpringerBriefs in Computer Science. Springer International Publishing, Cham, 1st ed. 2017. edition.

[12] Bodie, Z., Kane, A., and Marcus, A. J. (2014). *Investments*. McGraw-Hill Education, 10th edition.

[13] Bonassi, F., Farina, M., and Scattolini, R. (2020). Stability of discrete-time feed-forward neural networks in narx configuration. *ArXiv*, abs/2012.03741.

[14] Brownlees, C. T., Cipollini, F., and Gallo, G. M. (2010). Intra-daily

Volume Modeling and Prediction for Algorithmic Trading. *Journal of Financial Econometrics*, 9(3):489–518.

[15] Brownlees, C. T. and Vannucci, M. (2013). A bayesian approach for capturing daily heterogeneity in intra-daily durations time series. *Studies in Nonlinear Dynamics and Econometrics*, 17(1):21–46.

[16] Bucci, A. (2020). Realized Volatility Forecasting with Neural Networks. *Journal of Financial Econometrics*, 18(3):502–531.

[17] Chen, R., Feng, Y., and Palomar, D. (2016). Forecasting intraday trading volume: a kalman filter approach. *Available at SSRN 3101695*.

[18] Colby, R. W. (2003). *The encyclopedia of technical market indicators*. McGraw-Hill.

[19] Criscuolo, A. M. and Waelbroeck, H. (2019). Event-aware volume forecasting. predicting intraday volume using a coarse-grained hawkes model.

[20] Demir, S., Mincev, K., Kok, K., and Paterakis, N. G. (2020). Introducing technical indicators to electricity price forecasting: A feature engineering study for linear, ensemble, and deep machine learning models. *Applied Sciences*, 10(1):255.

[21] Di Persio, L. and Honchar, O. (2016). Artificial neural networks approach to the forecast of stock market price movements. *International Journal of Economics and Management Systems*, 1(1):158–162. Cited By :20.

[22] Dincer, N. G. and Akkuş, Ö. (2018). A new fuzzy time series model based on robust clustering for forecasting of air pollution. *Ecological Informatics*, 43:157–164.

[23] DiPietro, R., Rupprecht, C., Navab, N., and Hager, G. D. (2018). Analyzing and exploiting NARX recurrent neural networks for long-term dependencies.

[24] Dixon, M., Klabjan, D., and Bang, J. H. (2017). Classification-based financial markets prediction using deep neural networks. *Algorithmic-finance/af176*.

[25] Du, K. and Swamy, M. (2013). *Neural Networks and Statistical Learning*. SpringerLink : Bücher. Springer London.

[26] Engle, R. F. and Sokalska, M. E. (2012). Forecasting intraday volatility in the US equity market. Multiplicative component GARCH. *Journal of Financial Econometrics*, 10(1):54–83.

[27] Gallegati, M. and Semmler, W. (2014). Series Title Dynamic Modeling and Econometrics in Economics and Finance. Springer, Cham.

[28] Gulumbe, Suleiman, Asare, and Abubakar (2016). "forecasting volatility of nigerian crude price using non-linear auto-regressive with exogenous (narx) inputs model". *Imperial Journal of Interdisciplinary Research (IJIR)*.

[29] Hasan, M. M., Ali Pourmousavi, S., Jahanbani Ardakani, A., and Saha, T. K. (2020). A data-driven approach to estimate battery cell temperature using a nonlinear autoregressive exogenous neural network model. *Journal of Energy Storage*, 32:101879.

[30] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[31] Hsieh, T.-J., Hsiao, H.-F., and Yeh, W.-C. (2011). Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm. *Applied Soft Computing*, 11(2):2510–2525. The Impact of Soft Computing for the Progress of Artificial Intelligence.

[32] Hu, Y., Ni, J., and Wen, L. (2020). A hybrid deep learning approach by integrating lstm-ann networks with garch model for copper price volatility prediction. *Physica A: Statistical Mechanics and its Applications*, 557:124907.

[33] Huptas, R. (2017). Forecasting intraday traded volume with the weibull acv model: an application to polish stocks. The 11th Professor Aleksander Zelias International Conference on Modelling and Forecasting of Socio-Economic Phenomena, pages 103–112.

[34] Huptas, R. (2019). Point forecasting of intraday volume using

bayesian autoregressive conditional volume models. *Journal of Forecasting*, 38(4):293–310.

[35] Hurst, H. (1951a). Long term storage capacity of reservoirs transactions of the american society of civil engineers 116.

[36] Hurst, H. E. (1951b). Long-term storage capacity of reservoirs. *Transactions of the American society of civil engineers*, 116(1):770–799.

[37] Ito, R. (2016). Spline-dcs for forecasting trade volume in high-frequency financial data.

[38] J, C. P. M. and M, M. (2016). Forecasting east asian indices futures via a novel hybrid of wavelet-pca denoising and artificial neural network models. *PLoS ONE*, 11(16). e0156338.doi:10.1371/journal.pone.0156338.

[39] Jansen, S. (2020). *Machine Learning for Algorithmic Trading - Second Edition*. Packt, 2 edition.

[40] Kaastra, I. (1994). Forecasting futures trading volume using neural networks.

[41] Khashei, M., Bijari, M., and Raissi Ardali, G. A. (2009). Improvement of auto-regressive integrated moving average models using fuzzy logic and artificial neural networks (anns). *Neurocomputing*, 72(4):956–967. Brain Inspired Cognitive Systems (BICS 2006) / Interplay Between Natural and Artificial Computation (IWINAC 2007).

[42] Kyaw, N. A., Los, C. A., and Zong, S. (2006). Persistence characteristics of latin american financial markets. *Journal of Multinational Financial Management*, 16(3):269–290.

[43] Li, Z. and Tam, V. (2017). Combining the real-time wavelet denoising and long-short-term-memory neural network for predicting stock indexes. pages 1–8.

[44] Li, Z. and Tam, V. (2017). Combining the real-time wavelet denoising and long-short-term-memory neural network for predicting stock indexes. pages 1–8.

[45] Libman, D., Haber, S., and Schaps, M. (2019). Volume prediction with neural networks. *Frontiers in Artificial Intelligence*, 2:21.

[46] Lin, T., Horne, B., Tino, P., and Giles, C. (1996). Learning long-term dependencies in narx recurrent neural networks. *IEEE transactions on neural networks*, 7(6):1329–1338.

[47] Lipka, J. and Los, C. (2002). Persistence characteristics of european stock indexes. *Kent State University. Working paper, Kent.*

[48] Liu, X. and Lai, K. K. (2017). Intraday volume percentages forecasting using a dynamic svm-based approach. *Journal of Systems Science and Complexity*, 30(2):421–433.

[49] Ma, S. and Li, P. (2021). Predicting daily trading volume via various hidden states. *arXiv preprint arXiv:2107.07678.*

[50] Manigandan, P., Alam, M., Alharthi, M., Khan, U., Alagirisamy, K., Pachiyappan, D., and Rehman, A. (2021). Forecasting natural gas production and consumption in united states-evidence from sarima and sarimax models. *Energies*, 14(19):6021.

[51] Marcjasz, G., Uniejewski, B., and Weron, R. (2019). On the importance of the long-term seasonal component in day-ahead electricity price forecasting with narx neural networks. *International Journal of Forecasting*, 35(4):1520–1532.

[52] Markov, V., Vilenskaia, O., and Rashkovich, V. (2017). Quintet volume projection. *The Journal of Trading*, 12(2):28–43.

[53] Miffre, J. and Rallis, G. (2007). Momentum strategies in commodity futures markets. *Journal of Banking & Finance*, 31(6):1863–1886.

[54] Miller, R. S. and Shorter, G. (2016). *High frequency trading: Overview of recent developments*, volume 4. Congressional Research Service Washington, DC.

[55] Moskowitz, T. J., Ooi, Y. H., and Pedersen, L. H. (2012). Time series momentum. *Journal of Financial Economics*, 104(2):228–250. Special Issue on Investor Sentiment.

[56] Nason, G. (2008). *Wavelet Methods in Statistics with R*. Springer Publishing Company, Incorporated, 1 edition.

[57] Navon, A. and Keller, Y. (2017). Financial time series prediction using deep learning.

[58] Nettles, J., Brayer, N., Jenner, C., Ngo, A., Putnam, C., Shank, T., Todd, A., and Beling, P. (2015). Forecasting intraday volume distributions. pages 97–102.

[59] Olbryś, J. and Oleszczak, A. (2020). Intraday patterns in trading volume. evidence from high frequency data on the polish stock market. In *International Conference on Computer Information Systems and Industrial Management*, pages 390–401. Springer.

[60] Patel, J., Shah, S., Thakkar, P., and Kotecha, K. (2015). Predicting stock market index using fusion of machine learning techniques. *Expert Systems with Applications*, 42(4):2162–2172.

[61] Peters, E. E. (1991 - 1991). *Chaos and order in the capital markets : a new view of cycles, prices, and market volatility / Edgar E. Peters.* Wiley finance editions. Wiley, New York.

[62] Peters, E. E. (1994 - 1994). *Fractal market analysis : applying chaos theory to investment and economics / Edgar E. Peters.* J. Wiley Sons, New York.

[63] Pirrong, C. (2005). Momentum in futures markets. *Available at SSRN 671841*.

[64] Qian, B. and Rasheed, K. (2004). Hurst exponent and financial market predictability. In *IASTED conference on Financial Engineering and Applications*, pages 203–209. Proceedings of the IASTED International Conference Cambridge, MA.

[65] Qin, Y., Song, D., Chen, H., Cheng, W., Jiang, G., and Cottrell, G. (2017). A dual-stage attention-based recurrent neural network for time series prediction. *arXiv e-prints*, page arXiv:1704.02971.

[66] Rojas, I., Pomares, H., and Valenzuela, O. (2017). *Advances in Time Series Analysis and Forecasting: Selected Contributions from ITISE 2016*. Contributions to statistics. Springer International Publishing AG, Cham.

[67] Samborski, S., Liang, X., Ge, Z., Sun, L., He, M., and Chen, H. (2019). Lstm with wavelet transform based data preprocessing for stock price prediction. *Mathematical Problems in Engineering*, 2019:1340174.

[68] Sancetta, A. (2019). Intraday End-of-Day Volume Prediction*. *Journal of Financial Econometrics*, 19(3):472–495.

[69] Satish, V., Saxena, A., and Palmer, M. (2018). Predicting intraday trading volume and volume percentages.

[70] Scott, D. W. (2001). Parametric statistical modeling by minimum integrated square error. *Technometrics*, 43(3):274–285.

[71] Sezer, O. B., Gudelek, M. U., and Ozbayoglu, A. M. (2020). Financial

time series forecasting with deep learning : A systematic literature review: 2005–2019. *Applied Soft Computing*, 90:106181.

[72] Shumway, R. H. (2017 - 2017). *Time series analysis and its applications : with R examples / Robert H. Shumway, David S. Stoffer.* Springer texts in statistics. Springer, Cham, Switzerland, fourth edition. edition.

[73] Shumway, R. H., Stoffer, D. S., and Stoffer, D. S. (2000). *Time series analysis and its applications*, volume 3. Springer.

[74] Silverman, B. and Johnstone, I. (2005). Ebayesthresh: R programs for empirical bayes thresholding. *Journal of Statistical Software*, 12.

[75] Staudemeyer, R. C. and Morris, E. R. (2019). Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*.

[76] Stepchenko, A., Chizhov, J., Aleksejeva, L., and Tolujew, J. (2017). Nonlinear, non-stationary and seasonal time series forecasting using different methods coupled with data preprocessing. *Procedia Computer Science*, 104:578–585. ICTE 2016, Riga Technical University, Latvia.

[77] Sun, T., Wang, J., Ni, J., Cao, Y., and Liu, B. (2019). Predicting futures market movement using deep neural networks. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 118–125. IEEE.

[78] Tang, T.-L. and Shieh, S.-J. (2006). Long memory in stock index futures markets: A value-at-risk approach. *Physica A: Statistical Mechanics and its Applications*, 366:437–448.

[79] Taylor, J. W. (2010). Exponentially weighted methods for forecasting intraday time series with multiple seasonal cycles. *International Journal of Forecasting*, 26(4):627–646.

[80] Tse, Y. (2015). Momentum strategies with stock index exchange-traded funds. *The North American Journal of Economics and Finance*, 33:134–148.

[81] Turkman, K. F. (2014). *Non-Linear Time Series Extreme Events and Integer Value Problems / by Kamil Feridun Turkman, Manuel González Scotto, Patrícia de Zea Bermudez.* Springer International Publishing, Cham, 1st ed. 2014. edition.

[82] Valipour, M. (2015). Long-term runoff study using sarima and arima models in the united states. *Meteorological Applications*, 22(3):592–598.

[83] Vargas, M. R., dos Anjos, C. E., Bichara, G. L., and Evsukoff, A. G. (2018). Deep learning for stock market prediction using technical indicators and financial news articles. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

[84] Wang, L. and Gupta, S. (2013). Neural networks and wavelet de-noising for stock trading and prediction. in: Pedrycz w., chen sm. (eds) time

series analysis, modeling and applications. *Intelligent Systems Reference Library*, 47.

[85] Wang Y, Xu C, Wang Z, and Yuan J (2019). Seasonality and trend prediction of scarlet fever incidence in mainland china from 2004 to 2018 using a hybrid sarima-narx model. *PeerJ*, 7:e6165.

[86] Williams, E. E. and Dobelman, J. A. (2017). *Quantitative Financial Analytics: The Path to Investment Profits*. World Scientific Publishing Company.

[87] Xu, H., Liang, J., and Zang, W. (2021). Prediction of agricultural commodities futures prices: A dqn-lstm method.

[88] Yan, H. and Ouyang, H. (2018a). Financial time series prediction based on deep learning. *Wireless Personal Communications*, 102(2):683–700.

[89] Yan, H. and Ouyang, H. (2018b). Financial timer series prediction based on deep learning. *Wireless Pers Commun*.

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

[91] Zhong, X. and Enke, D. (2017). Forecasting daily stock market return using dimensionality reduction. *Expert Systems with Applications*, 67:126–139.