Forecasting Bitcoin Price Using An Adaptive Grey System Deep Neural Network

By

Yousaf Abdul Khaliq

A thesis submitted in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

Middle Tennessee State University

December 2022

Thesis Committee:

Dr. Medha Sarkar

Dr. Jaishree Ranganathan

Dr. Arpan Sainju

ACKNOWLEDGEMENTS

Mom, Dad, and God.

ABSTRACT

Bitcoin was created in 2009 by a person or group of persons under the name Satoshi Nakamoto. Bitcoin trading quickly grew along with the creation of numerous other cryptocurrencies in what is now the crypto market. Linear and non-linear methods have been applied to the prediction of bitcoin price including Support Vector Machines, Autoregressive Integrated Moving Average, Random Forests, and Recurrent Neural Networks among many others. Grey System Theory, developed by Deng Julong in 1982, is a linear forecasting method known for performing well with limited data sets. The aim of this research is to forecast bitcoin price using a non-linear approach that incorporates Grey System Theory. The result is a well generalized non-linear model trained on only 60 days of bitcoin price data.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In 2009, Satoshi Nakamoto created a cryptocurrency called bitcoin: a ”peer to peer electronic cash system” that would effectively cut out the middle man or ”third party”. For this to work, there would need to be a solution to the double spend problem - when the same instance of digital currency is transacted multiple times. Nakamoto accomplished this by creating a peer to peer network that would generate a chain of blocks (later dubbed the blockchain) where each block was an immutable record of bitcoin transactions between users. A block could only be added to the chain by way of a unique consensus algorithm called ”proof of work”. The consensus nature of the algorithm also eliminated the possibility of collusion between nodes on the network, thereby creating a ”trustless network”, one in which trust is not needed[1].

Bitcoin trading quickly grew along with the creation of numerous other cryptocurrencies in what is now the crypto market. The price of one bitcoin in 2010 was well under a penny. The price broke $10 in 2012. About a decade later, one bitcoin reached its all-time-high of $67,567 thus granting a bitcoin investment the highest potential return of all time. But the asset has proven to be just as volatile as it is profitable. The constant boom and bust cycles create few winners and many losers. At the time of this writing, one bitcoin is worth just under $20,000, having fallen over 60% from the all-time-high [1].

Price data for bitcoin is fairly easy to obtain as it is tracked by various sources and market exchanges. This data is considered sequential, or more specifically, time-series data: observations collected over certain increments of time. In this case, bitcoin price is tracked throughout time. In graphical terms, price is on the y-axis and time is on the x-axis. Other examples of time series data include weather measurements, asset prices, annual sales, and server performance, all collected over time.

The aim of this research is to forecast bitcoin price using a non-linear approach that

incorporates Grey System Theory [2]. Both linear and non-linear methods have commonly been used to predict asset price, and bitcoin is no exception. Out of the linear group, current research includes Support Vector Machines (SVM), Logistic Regression (LR), univariate Autoregressive (AR), Simple Exponential Smoothing (SES), Random Forests (RF), and most notably Autoregressive Integrated Moving Average (ARIMA)[3][4][5]. For non-linear methods - generally neural network based - current research includes Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit Networks (GRU), and WaveNets, to name a few[6][7][4][8][9][10][11].

The research is often times regression based; however, there is also work done with respect to classification. The difference between regression and classification can be simply explained using the asset price example: regression is forecasting the price while classification is predicting the direction in which the price will move. Classification problems are discrete in nature, and for asset prediction, there are only two considerable options: increase or decrease. Linear and non-linear methods can and have been applied to classification problems[12].

Research involving the prediction of bitcoin price began increasing around the year 2018, with many more papers published in 2020 and 2021. Linear and non-linear methods are commonly compared with only a handful of novel approaches. In a 2018 paper on forecasting time series stock market data obtained from Yahoo Finance, [3] compared the traditional ARIMA linear method to the more modern deep learning method of LSTM. It was found that the LSTM model reduced error by about 85 percent when compared to ARIMA indicating a significant improvement[3]. However, a 2019 paper from [7] forecasting bitcoin price, shows results that conclude the linear methods of ARIMA and SVR outperformed their LSTM model due to a faster minimum convergence ability. [7] also claims that due to the Efficient Market Hypothesis (the theory that a share price reflects all information possible, thereby eliminating the possibility of beating the market), linear methods perform

better when predicting the highly volatile bitcoin price than machine learning methods. Another 2020 paper on bitcoin price forecasting compares linear methods along with a simple artificial neural network to find that the SVM outperformed all the models while the ARIMA and Bayesian methods outperformed all the univariate models [4]. Lastly, a 2020 paper forecasting bitcon price movement using continuous and discrete datasets showed the random forest method performing the best on the continuous dataset and the artificial neural network performing the best on the discrete dataset.

In terms of classification - predicting which direction the price will move - two papers come to the forefront. In 2018, [9] compared the results of an LSTM model and an ARIMA model tasked with classification. It was found that the LSTM model performed significantly better than the ARIMA model with a 52 percent accuracy; however, the LSTM model took much longer to train. Another paper from 2020, used random forests to forecast bitcoin price direction while also analyzing how interest rates, inflation, and market volatility could affect the price. The analysis showed a 5-day price prediction accuracy between 75 and 80 percent and an 85 percent accuracy for 10 to 20 day price prediction [5].

These comparative analyses and papers provide a good foothold for further research; however, they lack in novel approaches to forecasting bitcoin price. A handful of papers and research present novel ideas from 2020 to 2022 using methods inspired from actual trading strategies, the Monte Carlo approach, and even performing twitter and social media platform sentiment analysis [13][14][10]. In the Monte Carlo approach from [13], the future value of bitcoin is forecasted using a fractional Ornstein-Uhlenbeck driven by a Levy process with a time-variant Hurst parameter. The Hurst parameter revealed bitcoin's dynamic random behavior while also exhibiting periods of mean reversion. A Normal Inverse Gaussian appeared to provide the best fit distribution and the entire approach resulted in 95 percent price prediction accuracy on three different dates in 2019 [13]. Another novel idea came in a 2022 paper from [14], where correlations between general sentiment of the community

and bitcoin price were analyzed. The research produced a Recurrent Radial Basis Function Network using big data from social media networks to predict bitcoin price with the model being able to achieve a 93 percent accuracy[14].

When reflecting upon current research, a few key areas of potential present themeselves: novel ideas using traditional approaches, traditional ideas using novel approaches, and novel ideas using novel approaches. Additionally, most of the current research in bitcoin price forecasting suggest their respective models could be used to make investment and/or trading decisions; however, not all of their results show a high accuracy on testing data. Meaning, their models train well but do not generalize well. In some cases, a result of 50 percent accuracy is presented as high performance when 50 percent is simply the common sense baseline (any human can predict with 50 percent accuracy whether the price will go up or down at any time). A novel approach that results in high accuracy against testing (new) data is lacking in current research.

Our work finds a marriage between forecasting bitcoin price, neural networks, and a linear method called Grey System Theory. The first order Grey Model, GM(1,1), is a linear system that learns constant parameters that can fit a small dataset and can also make reliable short-term predictions; however, in a high variance dataset like bitcoin price, the GM(1,1) has difficulty making reliable predictions[15]. Hence, our motivation to combine the novel physics informed neural network with the GM framework. This enables us to learn non-linear grey system parameters, which is able to capture the non-linearity in the dataset. There is plenty of work on forecasting with Grey System Theory, but not much on forecasting bitcoin prices. A 2020 paper from [16], does produce a 98 percent accuracy level based on certain time frames using the GM(1,1) model, but significant potential exists in using an adaptive grey model applied to time series forecasting. RNN's and their many variations have been used to forecast non-linear time series; our research will provide a comparison of our Adaptive GS method alongside a simple RNN, LSTM, GRU, and Bidirectional-LSTM

networks.

It is our intention to give a clear and broad view of bitcoin and our method for forecasting bitcoin prices. We will begin by providing a closer look into the world of cryptocurrency and blockchain in Chapter 2. Chapter 3 expands on the fundamentals of artificial neural networks and the specific architectures we will be comparing. Chapter 4 dives into Grey System Theory and our Adaptive Grey System Deep Neural Network approach. Lastly, Chapter 5 presents the results.

CHAPTER 2

CRYPTOCURRENCY

Bitcoin. What is it? Why was it created? What problem does it solve, if any? In this chapter, we will be answering these important questions and more.

2.1 Trusted Third Parties and the History of Money

To understand cryptocurrency, we have to take a step back and review basic economic concepts of money, or mediums of exchange. Before money, as we know it today, goods and services were exchanged directly - without any medium - by way of the barter system. For example, Jon gave Mary 10 chickens for 1 goat. The barter system served its purpose but it did not scale well, and as human civilizations grew, so did the need for strong currency and economic systems.

Thus began a transition to commodity money. **Commodity money** is an object that represents a certain amount of purchasing power while also having intrinsic value. Gold and silver are popular examples but salt, copper, tea, silk, and many others were also used. Good commodity money is durable, rare, intrinsically valuable, and easily exchangeable. Gold and silver fits these characteristics well. But who decides the value of these objects? Governing bodies generally declared the value within their nations but what about international trade? And what about the supply of these monies? Many economic concepts come into play when it comes to currency and trade and as markets continued to develop, so did money[17].

One of the difficulties of commodity money, specifically coins, was that the gold or silver content of the coin couldn't always be trusted. This would make international trade especially difficult. So they traded debts, and IOU's turned into paper money. All parties agreed the IOU had value by trusting a solvent middleman. Thus introducing the **trusted third party**. The first recorded trusted third party was the Medici family of France. This "trust" gave the lender power and they quickly realized they could "create" money in the form of more IOU's or debt that could not be taxed nor debased. Fast forward to today,

and we find entire countries buying and selling each other's debts. Many other financial inventions arose throughout history; however, when it comes to bitcoin, a trusted third party is the most significant[18].

As we continue through time and now focusing on the United States, from 1781 to the 1940's, the U.S.A. adopted the **Gold Standard**: a system in which the currency derived its value from gold and could be exchanged for gold at a certain rate. To prevent random financial institutions popping up across the country without any regulation, a centralized bank was created - the Federal Reserve. This allowed the United States to expand or contract the dollar and control supply, all tied to the value of gold also reducing the regular boom and bust cycles experienced in the American markets[19].

In 1929, the great depression affected the world. The Federal Reserve printed all the money it could to bring life back to the economy but it needed gold. So in 1933 a law was passed forcing all American citizens to sell their gold to the government at a fixed price or go to prison. An even better rate was offered internationally making gold flow into the country allowing for an increase in dollars printed. Buying other government's gold allowed dollars to be spread across the globe making the dollar a widely used and recognized currency. In the devastation of World War 2, the U.S. dollar proved to be the most stable currency so other countries pegged their currencies to the dollar, even further solidifying its position as the world's strongest currency. America now owned more than half of the world gold reserves. Governments began disputing the value of their currency versus the dollar and demanded to buy gold back for their paper currency. Although the gold standard had slowly declined in use since the Great Depression, in 1971, President Nixon officially ended the issue by severing the gold standard, ending the ability of foreign nations buying American gold, and thus making the dollar a **fiat currency**: money with no intrinsic value and backed solely by the full faith and credit of the United States government; the ultimate trusted third party[19].

2.2 **The Double Spend Problem**

In the digital age of today, what is the point of paper money? A customer can buy a cup of coffee through various digital means - with just a tap of their phone, even - rarely does today's consumer make a purchase with cash. But what exactly happens when a credit or debit card is used to make a transaction? From a broad point of view, with every swipe, a trusted third party collects the consumer and vendor's information and exchanges the digital funds accordingly. The main purpose of the trusted third party here is to prevent the digital instance of the funds to be spent more than once. Imagine a line of code representing the money, and then a bad actor simply copying and pasting that line of code and using it for another transaction. This is called the **double spend problem** and it creates a costly issue for banks, vendors, and consumers[20].

2.3 **Blockchain**

A commonly overlooked but highly important distinction about the word "Bitcoin" is that Bitcoin with a capital 'B' refers to the Bitcoin protocol and entire blockchain network. Whereas bitcoin with a lower case 'b' refers to the digital asset. The **blockchain** can be thought of as a digital ledger in which every entry, or transaction, must be approved and no approved entry can be changed. Entries are grouped together in blocks which are "chained" together by way of a hash value; every block points directly to the previous block all the way to the genesis block[21].

One of the most important features of the blockchain is how the blocks are approved. This is done by a **consensus algorithm**: a computer process used in distributed systems to reach an agreement on a single data value. A **distributed system**, in its most basic form, is a group of independent computers (nodes) linked by a network. The consensus algorithm and the distributed nature of Bitcoin's architecture gives it the crucial quality of **decentralization**[21]. A decentralized system is where decisions are made by every entity in the network, as opposed to centralization where decisions are made by one or few entities.

The Federal Reserve is an example of centralization[19].

The significance of decentralization can be further explained by the classic Byzantine Generals Problem. A group of Byzantine generals have surrounded an enemy city and must decide when to attack. The generals can only communicate with each other via one messenger; however, one or more generals may be a traitor and can cause the entire army to attack at the wrong time by sending their traitorous message to the other generals. The problem is to find an algorithm such that the loyal generals are able to reach an agreement upon when to attack. This problem has many applications towards reliable computer systems and their inevitably faulty components[22].

In the case of the Bitcoin blockchain, the ”generals” are the nodes in the network and the decision to attack is, instead, the verification of a bitcoin transaction by way of the consensus algorithm. Many approaches to consensus have been developed; Bitcoin uses the **Proof of Work (PoW)** approach. In PoW, the participants act as a kind of verifier by calculating a hash value for the block header. This value must meet a set numerical requirement and if one node achieves the correct calculation, all other nodes verify the work. This work requires vast computational power and time consumption, and thus, an incentive is given to the nodes in the form of bitcoin. The nodes performing these calculations are called **miners** and the work they perform is called **mining**. Bitcoin mining is also how the total supply of bitcoin in circulation increases[21].

2.4 Current Market

Cryptocurrency is primarily traded through exchanges, a marketplace to buy, sell, and hold. Coinbase, Binance, Gemini, and Kucoin are some of the more popular exchanges that have millions, even billions of dollars of daily trading volume. As of October 2022, the global **market capitalization (MC)** for cryptocurrencies is roughly $1 trillion with over 20,000 different cryptocurrencies in existence. That means the entire world has invested a total of $1 trillion in cryptocurrencies. To put this in perspective, the current MC of Apple

stock is over $2 trillion. Bitcoin has a current MC of $400 billion, roughly 40% of the entire cryptocurrency market making it the market leader. All cryptocurrencies besides bitcoin are referred to as **altcoins** with Ethereum having a MC of $140 billion making it the largest altcoin and second largest cryptocurrency behind bitcoin.

A simple way to understand how the value of bitcoin fluctuates is by understanding the relationship between supply and demand. If an asset is in high demand - everyone wants it - it can be priced high. If it is in high demand, but there is also a high supply, it would be priced more fairly at an equilibrium price. If it is in high demand, but there is a low supply, the price will skyrocket[17]. Bitcoin has a maximum circulating supply of 21 million with the current circulating supply to be around 19 million (supply increases as more bitcoin is mined). This means it is quite scarce, almost like a digital gold[23].

There are two types of analysis when it comes to asset price: fundamental and technical. **Fundamental analysis** involves measuring an asset's intrinsic value by way of economic and financial factors. For example, a stock's value could be analyzed by researching the company, leadership team, utility/usefulness of the asset, financial reports, etc. On the other hand, purely analyzing market data and price action to identify trends is **technical analysis**[23]. Forecasting bitcoin price using linear and/or non-linear methods using market data would be considered technical analysis, which is the focus of this research.

CHAPTER 3

ARTIFICIAL NEURAL NETWORKS

An **artificial neural network (ANN)** can be described as a data filtration and representation extraction mechanism. It receives inputs which pass through **layers**. The general purpose of a layer is to extract a different, more useful representation of the previous input. This process of continuously feeding data forward and backward through the ANN repeats until an output is achieved which is measured for validity and accuracy. The more layers a model has, the "deeper" it is said to be, hence the name deep learning. **Deep learning** is a subset of machine learning with the most notable difference being the use of multiple hidden layers in deep learning [12]. Each layer also has a predetermined number of **units or neurons** through which data flows. A forward and backward pass through the training data is called an **epoch**. The connections between each layer's units have **weights** and a **bias** which is used to offset results in an, ideally, more useful direction. The weights and biases are updated as the model runs with the hope of finding potential solutions to the specified problem. Optimizing weights and obtaining the result can be thought of as the "learning" part of machine learning. The architecture of a simple ANN can be visualized in Figure (1) where the lines between the nodes represent the weights.

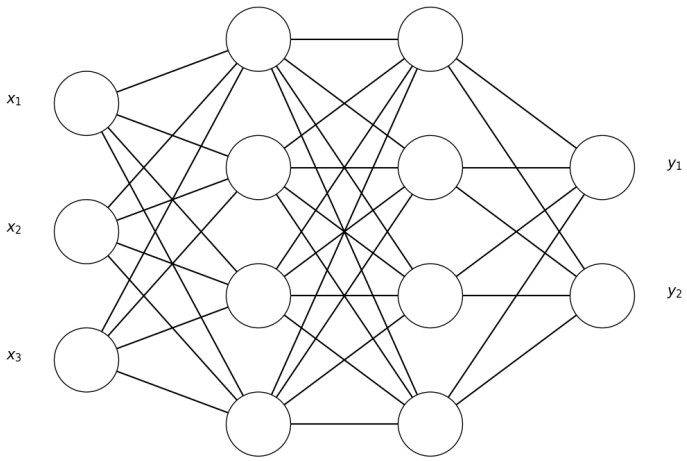

Figure 1: Simple Representation of an ANN

From the mathematical perspective, an ANN is a composition of functions where the task is to find optimal output functions that match some target function.

The ANN displayed in Figure (1) can be described as follows;

$$Y(X,\theta) = \sigma(W_L\sigma(\ldots\sigma(W_2\sigma(W_1X+b_1)+b_2)\ldots)+b_L), \tag{3.0.1}$$

where $X = [x_1, x_2, x_3]^T$ is the input vector and $Y = [y_1, y_2]^T$ is the output vector, and $L-1$ is the number of hidden layers in the neural network. The neural network in (3.0.1) is a composition of linear functions together with a nonlinear function denoted $\sigma(\cdot)$, σ is usually called an activation function [24]. A backpropagation algorithm usually called an optimizer is used to adjust the neural network trainable parameters θ, given as follows;

$$\theta = [W_1, \ldots, W_L, b_1, \ldots, b_L].$$

After each epoch, θ is updated until (3.0.2) is minimized.

$$\theta^* = \min_{\theta} Y(X, \theta). \tag{3.0.2}$$

After understanding the input - layer - output feed forward architecture, the next concept to be understood is that of a tensor. A **tensor** is a data structure that can be shaped and formed to accommodate data and model requirements [12]. Tensors create the ability to provide inputs in a way that the computer can understand: through code and numerical values. Deep learning systems use tensors as their data structure making them fundamental to the field. The **shape** of a tensor is a tuple of integers that describes the number of axes and number of elements along each axis of a tensor. The **dimensionality** of a tensor refers to the number of axes it has [12]. For instance, a rank-2 tensor, or 2D tensor, has two axes. This is commonly known as a matrix with the rows being one axis and the columns being another axis. A rank-1 tensor has one axis and is known as a vector or an array of scalar values. A scalar is a rank-0 tensor. Dimensionality can also describe a structure such as a vector; however, in this case, dimensionality describes the number of elements along the axis and not the number of axes themselves. This is an important distinction to make and can easily cause confusion. For example, a 2D tensor is not the same as a 2D vector. A 2D tensor has two axes and a 2D vector has one axis with 2 elements along it[12]. Timeseries data almost always follows a 3D tensor with the shape (samples, timesteps, features).

In order to shape tensors, manipulate them as we see fit, and ultimately glean useful representations from them, operations like multiplication and addition must be performed. Dot product between matrices and addition of vectors and scalars are common tensor operations[12].

The most important operation performed on a tensor is done through an **activation function**. This crucial function is what propels deep learning from linearity to non-linearity and allows a model to search for and find solutions impossible for a linear model to find.

Each unit has a designated activation function whose purpose is to decide whether or not the unit is "activated" or not. To be activated means the weights and biases are updated. These values are updated by taking partial derivatives and to do that, the chain rule is applied. If a weight is changed in the first hidden layer, its effect can be seen in the output of the next layer and the following layer and so on all the way to the output layer. This enables the subsequent units and layers to update accordingly through a process called **backpropagation**. To understand this process and how the chain rule enables the model to update weights and biases in the right direction, more calculus is needed. Specifically, finding derivatives and what that means when solving a problem using deep learning.

Take a smooth and continuous function whose input is a tensor i.e. a tensor function, taking its derivative results in what's known as a **gradient**. Just like the derivative of a scalar function is the slope at a particular point, a gradient describes the curvature of a multidimensional surface described by the function [12]. Knowing a function's derivative allows us to understand how a small change in the input affects the output. In deep learning, a **loss or cost function** is used as a model assessment and different problems call for different loss functions. A simple example of a loss function is $Output = Actual - Predicted$. The loss function is minimized through an optimization process called **gradient descent** where the weights between the layers are updated. The "speed" of this process is determined by the **learning rate** - a value chosen by the machine learning engineer. Changing the attributes, such as the weights, of a neural network can be done in many different ways. These, such as gradient descent, are called **optimizers**.

3.1 Hyperparameter Tuning and Regularization

Many decisions need to made in order to improve the performance of a model. This process is called **hyperparameter tuning**. So far, the following parameters have been discussed and are choices that need to be made to improve a model's performance:

- number of hidden layers

- number of units
- number of epochs/batch size
- learning rate
- tensor shape
- training split (discussed below)
- loss function
- activation function
- optimizer
- dropout rate (if applicable)

When training a model, one of the most important aspects of model performance to monitor is its **training accuracy** and its **testing accuracy**. Before training, the data must be split into training data and testing data. These are fairly self-explanatory in that the model is trained using the training data and tested on the testing data. The testing data simulates what it would be like for the model to handle "new" data. How well the model performs on training data is its training accuracy and how well it performs on testing data is its testing accuracy. If a model has poor training and testing accuracy, it is said to be **underfitting** and has made assumptions about the data and has not "learned" enough. If a model's training accuracy is higher than its testing accuracy, this is a clear sign of **overfitting**. This means the model has memorized patterns in the training data but did not particularly "learn" the data so it does not perform well on new data. It is like a student that crams all the material the night before an exam but does not actually learn the material. That student may do well on the specific material they memorized but perform poorly on the exam. Performance on new data is called **generalization** and a model that generalizes well is a core objective for

machine learning based problem solving. The entire process of hyperparameter tuning with the intention of fitting the data accurately to prevent overfitting is called **regularization**. A widely used regularization technique - and one used in this research - is **dropout**. Dropout was developed by G. Hinton at the University of Toronto when he was at the bank and noticed that the tellers rotated positions when serving bank customers. He discovered it was a way to make it harder for the bank tellers to collude and steal from the bank. Applied to machine learning, dropout randomly removes units from layers in order to reduce their ability to memorize patterns in the data and promote generalization and ”learning” patterns in the data[12]. In the following sections, we introduce different model architectures, their mathematical representations, and sample code implementations using the **keras library**, an open source library developed by Google labs that streamlines the machine learning coding.

3.2 **Models**

3.2.1 Recurrent Neural Network

A **Recurrent Neural Network (RNN)** is a specific type of ANN which is able to retain information based on what it has seen thus far. A **hidden state** is maintained throughout the processing of data and the state is updated between each sequence from the input and previous hidden state. The hidden state can be thought of as a kind of ”memory” and an RNN can be thought of as a For Loop that reuses quantities in the previous iteration of the loop. As shown in Figure (2), an input and a hidden state is processed (in this case, by the tanh activation function) and an output and new state is generated. These two values become the input and new state of the next iteration until all the data is processed. The mathematical representation of these processes can be seen in equation (3.2.1). Tanh represents the hyperbolic tangent activation function, which has an output range between -1 and 1[12].

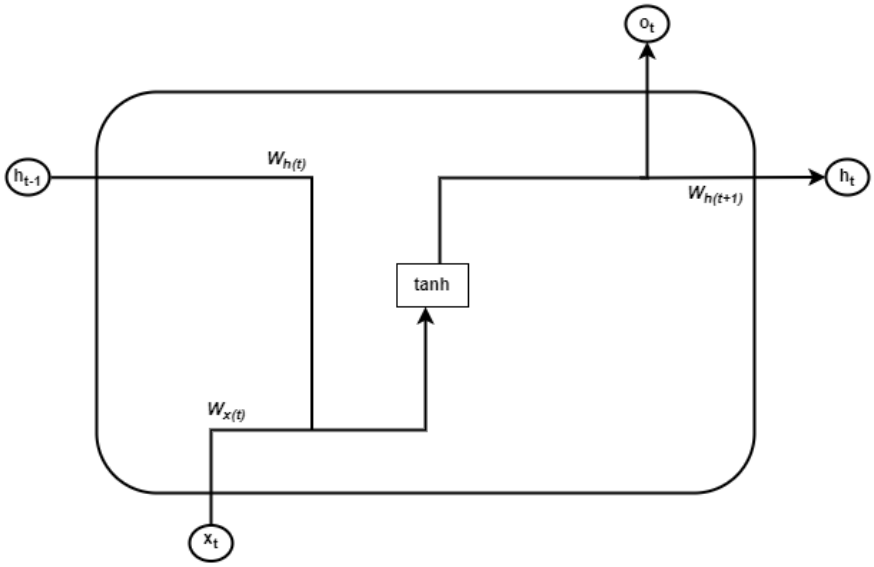

Figure 2: Simple Representation of a RNN

$$h_t = \tanh(W_{h_t} * h_{t-1} + W_{x(t)} * x_t + b), \tag{3.2.1}$$

- x_t is the input at time t,
- h_t is the hidden memory of the cell at time t,
- $W_{x(t)}$ is the weight matrix of x at time t,
- $W_{h(t)}$ is the weight matrix of h_{t-1} at time t.

Figure (3) shows the python code for the implementation of a simple RNN model. Importing "SimpleRNN" from the Keras library allows for effortless implementation and decision making. In each layer, decisions are made regarding number of units, activation function, and dropout percentage. Another common activation function is the **Rectified Linear Unit (ReLU)**. In the compilation step (model.compile), the optimizer and loss function are chosen, in this case, the adam optimizer and the mean squared error loss function.

The ReLu activation function is a simple max(x, 0) function where x is the input. It is an element-wise operation applied to a tensor resulting in non-negative output. Our research deals with asset prices that never go below zero, so the ReLU activation function is used often[24].

```
from tensorflow.keras.layers import SimpleRNN

model3 = Sequential()
model3.add(SimpleRNN(units = 50, activation = 'relu', return_sequences =
    True, input_shape = (X_train.shape[1], 4)))
model3.add(Dropout(0.2))
model3.add(SimpleRNN(units = 60, activation = 'relu', return_sequences =
    True))
model3.add(Dropout(0.3))
model3.add(SimpleRNN(units = 80, activation = 'relu', return_sequences =
    True))
model3.add(Dropout(0.4))
model3.add(SimpleRNN(units = 120, activation = 'relu'))
model3.add(Dropout(0.5))
model3.add(Dense(units =1))

model3.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

Figure 3: Simple RNN Implementation in Keras

Each time-step is a loop of the previous output. In theory, this means that the final output should contain the relevant information of the entire dataset meaning only the final output is needed; however, a problem known as the **vanishing gradient** - the calculated gradient reduces to zero over time and therefore does not propagate useful information through the network - prevents this from being the case[6]. This leads to the next model of choice: Long Short Term Memory.

3.2.2 Long Short Term Memory

The Long Short Term Memory (LSTM) model is a variant of the RNN. The LSTM is one of the most common models applied to timeseries data. It adds a way to “carry” information across time steps and helps address the vanishing gradient problem. This ability essentially

saves information for later, which makes older information from earlier timesteps remain relevant and prevents them from vanishing[12][6]. As seen in Figure (4), C can be called "carry" and imagined as a conveyor belt of information where signals can jump on and off when necessary. This conveyor belt regulates the next output and the next state (remember this is an RNN so it has states). These values are computed by way of transformations and use of an activation function. The LSTM model contains a **cell state** and **gates**: forget, input, and output. These gates control the flow of information through the LSTM model. The **forget gate** decides what information to keep from the previous timesteps. A sigmoid function compresses this value between 0 and 1 where 0 forgets and 1 remembers (equation (3.2.2)).

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{3.2.2}$$

The **input gate** quantifies the importance of the incoming information also by way of a sigmoid activation function (equation (3.2.3)) for decision making along with a tanh activation function for assigning a weight between -1 and 1 to the flowing information (equation (3.2.4)).

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{3.2.3}$$

$$\tilde{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C), \tag{3.2.4}$$

The **output gate** controls the output of the cell which becomes the next hidden state [6].

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{3.2.5}$$

$$h_t = o_t * \tanh(C_t), \tag{3.2.6}$$

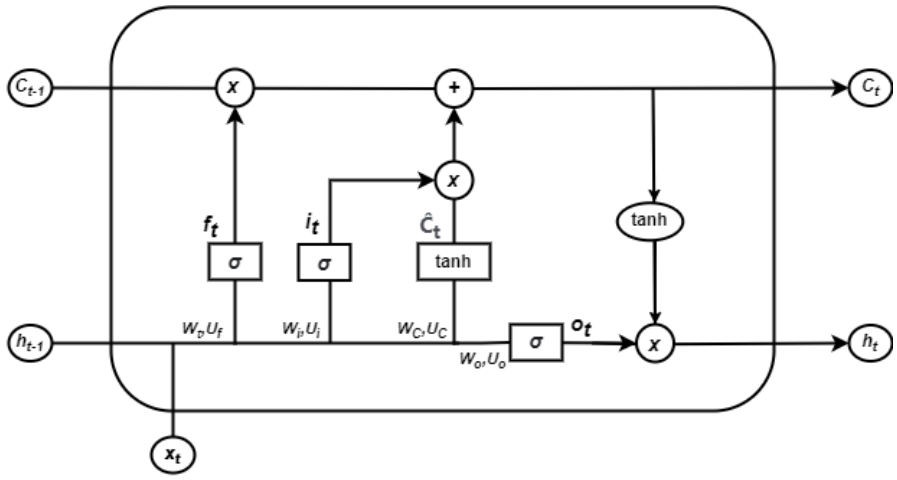

Figure 4: Simple Representation of an LSTM Network

Implementation of the LSTM can be seen in Figure (5). It is similar to the implementation of the simple RNN especially due to the Keras library. Again, we use the ReLu activation function, dropout regularization method, adam optimizer, and mean squared error loss function.

```
model1 = Sequential()
model1.add(LSTM(units = 50, activation = 'relu', return_sequences = True,
    input_shape = (X_train.shape[1], 4)))
model1.add(Dropout(0.2))
model1.add(LSTM(units = 60, activation = 'relu', return_sequences = True))
model1.add(Dropout(0.3))
model1.add(LSTM(units = 80, activation = 'relu', return_sequences = True))
model1.add(Dropout(0.4))
model1.add(LSTM(units = 120, activation = 'relu'))
model1.add(Dropout(0.5))
model1.add(Dense(units =1))

model1.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

Figure 5: Simple LSTM Implementation

3.2.3 Gated Recurrent Unit (GRU)

Another variant of the RNN is the Gated Recurrent Unit (GRU) model. It bears a resemblance to the LSTM and also aims to resolve the vanishing gradient problem; however,

the GRU gated architecture is different from LSTM's. Instead of a forget, input, and output gate like the LSTM, the GRU has only a reset and an update gate while also notably lacking an output gate. The **reset gate** is used to decide how much of the past information to forget. The **update gate** is used to determine how much of the past information should be passed forward.

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \tag{3.2.7}$$

$$r_t = \sigma(W_r x_r + U_r h_{t-1} + b_r), \tag{3.2.8}$$

$$\tilde{h}_t = \tanh(W x_t + U_r h_{t-1} + b_{\tilde{h}}), \tag{3.2.9}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t, \tag{3.2.10}$$

Due to the GRU's fewer parameters and operations than the LSTM, it is generally easier to train as seen in equations (3.2.7)(3.2.8)(3.2.9)(3.2.10). Figure (6) shows a simple representation of the GRU architecture.

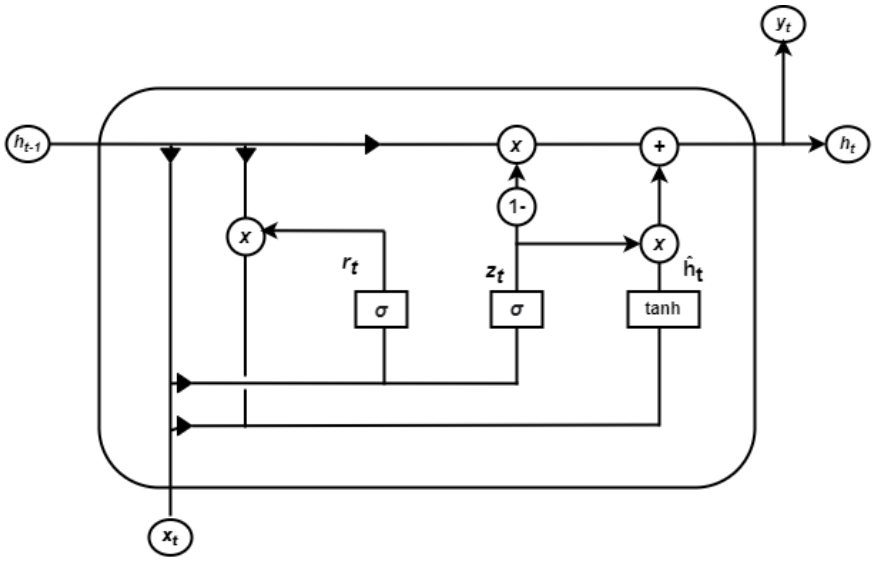

Figure 6: Simple Representation of a GRU Network

```
from tensorflow.keras.layers import GRU

model4 = Sequential()
model4.add(GRU(units = 50, activation = 'relu', return_sequences = True,
    input_shape = (X_train.shape[1], 4)))
model4.add(Dropout(0.2))
model4.add(GRU(units = 60, activation = 'relu', return_sequences = True))
model4.add(Dropout(0.3))
model4.add(GRU(units = 80, activation = 'relu', return_sequences = True))
model4.add(Dropout(0.4))
model4.add(GRU(units = 120, activation = 'relu'))
model4.add(Dropout(0.5))
model4.add(Dense(units =1))

model4.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

Figure 7: Simple GRU Implementation

Many more models exist each with their own strengths and weaknesses. For the purposes of this research, we have chosen to use a simple RNN, LSTM, and GRU model to compare with the Adaptive Grey Model which we introduce in the next chapter.

CHAPTER 4

ADAPTIVE GREY MODEL

4.1 **Grey System Theory**

If being in the "grey area" could be turned into theory and defined by mathematics, it would be called the Grey System Theory (GS). Developed by Deng Julong of Huazhong University of Science and Technology in 1982, GS defines a system as having either no information (black), full information (white), or a mixture of known and unknown information (grey) [2]. In the real World, inaccuracies are more probable, so black and white information are less likely. Most situations fall in between. GS uses partial and incomplete information to make forecast. One major advantage of using GS is that it does not require a lot of data and it does not require the available data to come from a particular distribution [15]. The GS has been used in many prediction and forecasting tasks such as time-series forecasting [25].

4.2 **A residual Grey forecasting method**

One of the widely used GS prediction models is the GM(1,1) [2, 25], which is a single variable first order grey model. GM(1,1) uses four or more data points, and it can get a high accuracy forecast [25] [26]. Creating this model begins with the construction of the Accumulation Generating Operation (AGO) which increases the smoothness of the data or sequence of data points used. The following steps essentially takes a data sequence and "slides" it forward, making a new prediction with each iteration. Mathematically, this is done by solving differential equations and finding parameter values. Similar to how neural networks perform backpropagation and gradient descent to solve for parameters that reduce the loss function. The following steps and equations were obtained from [15] and [26] and the results are shown in Chapter 5.

1. Construct the 1 - AGO sequence of X sequence. Set sequence

$$X^{(0)} = (x^{(0)}(1), x^{(0)}(2) \ldots, x^{(0)}(n)),$$

wherein

$$x^{(0)}(k) \geq 0, k = 1, 2 \ldots, n.$$

And $X^{(1)}$ is the 1 - AGO sequence of $X^{(0)}$:

$$X^{(1)} = (x^{(1)}(1), x^{(1)}(2) \ldots, x^{(1)}(n)),$$

among which

$$x^{(1)}(k) = \sum_{i=1}^{k} x^{(0)}(i), k = 1, \ldots n \tag{4.2.1}$$

2. Set the mean sequence of $Z^{(1)}$ of the 1 - AGO sequence

$$Z^{(1)} = (z^{(1)}(2), z^{(1)}(3) \ldots, z^{(1)}(n)),$$

among which

$$z^{(1)}(k) = \frac{1}{2}(x^{(1)}(k) - (x^{(1)}(k-1)) \tag{4.2.2}$$

The above two steps are to “whiten” the existing data by constructing the 1 - AGO sequence and the mean value sequence respectively, so as to fully explore its inherent laws and information.

3. Construct and solve the whitening differential equation. Set

$$x^{(0)}(k) + az^{(1)}(k) = b$$

as the mean value for of GM(1,1), hence the whitening differential equation would be:

$$\frac{dx^{(1)}}{dt} + ax^{(1)} = b \tag{4.2.3}$$

Set the parameter vector as $\hat{a} = (a,b)^T$, therefore least squares method could be used to estimate

$$\hat{a} = (B^T B)^{-1} B^T Y \tag{4.2.4}$$

$$Y = \begin{bmatrix} x^{(0)}(2) \\ x^{(0)}(3) \\ \vdots \\ x^{(0)}(n) \end{bmatrix}, B = \begin{bmatrix} -z^{(1)}(2) & 1 \\ -z^{(1)}(3) & 1 \\ \vdots & \vdots \\ -z^{(1)}(n) & 1 \end{bmatrix} \tag{4.2.5}$$

4. Calculate the response time. Parameter vector $\hat{a}$ could be calculated from (4.2.4) and (4.2.5), therein - a is development coefficient, which reflects the development trend of $\hat{x}^{(1)}$ and $x^{(0)}$, and b is the gray action. The time response of $\hat{x}^{(1)}(k)$ would be:

$$\hat{x}^{(1)}(k) = \left(x^{(0)}(1) - \frac{b}{a}\right)e^{-a(k-1)} + \frac{b}{a}, k = 1,2,\ldots,n \tag{4.2.6}$$

from (4.2.6), the reduced reduction equation would be:

$$\hat{x}^{(0)}(k) = \hat{x}^{(1)}(k) - \hat{x}^{(1)}(k-1), k = 1,2,\ldots,n \tag{4.2.7}$$

$$\hat{x}^{(0)}(k) = (1-e^{a})\left[\left(x^{(0)}(1) - \frac{b}{a}\right)e^{-a(k-1)}\right], k = 1,2,\ldots,n \tag{4.2.8}$$

And the analog sequence would be obtained from (4.2.8).

5. Construct residual sequence and conduct conditional decision. If the accuracy is not up to the requirement, the residual tail segment method is used for correction. The

residual sequence is defined as:

$$\varepsilon^{(0)} = (\varepsilon^{(0)}(1), \varepsilon^{(0)}(2), \ldots, \varepsilon^{(0)}(n)) = (x^{(0)}(1) - \hat{x}^{(0)}(1), x^{(0)}(2) - \hat{x}^{(0)}(2), \ldots, x^{(0)}(n) - \hat{x}^{(0)}(n)) \tag{4.2.9}$$

if $\forall k \geq k_0$, the symbol of k_0 is the same as that of $\varepsilon^{(0)}(k)$ and $n - k_0 \geq 4$, then

$$\left(\left|\varepsilon^{(0)}(k_0)\right|\right), \left(\left|\varepsilon^{(0)}(k_0+1\right|\right), \ldots, \left(\left|\varepsilon^{(0)}(n)\right|\right) \tag{4.2.10}$$

is modellable residual segment, which can be constructed according to steps (1) to (4). Its response time after the reduced reduction correction is:

$$\hat{x}^{(0)}(k+1) = \begin{cases} (1-e^{a})\left(x^{(0)}(1) - \frac{b}{a}\right)e^{-ak}, & k < k_0 \\ (1-e^{a})\left(x^{(0)}(1) - \frac{b}{a}\right)e^{-ak} \pm a_\theta\left(\varepsilon^{(0)}(k_0) - \frac{b_\theta}{a_\theta}\right)e^{-a_\theta(k-k_\theta)}, & k \geq k_0 \end{cases} \tag{4.2.11}$$

That is, the analog value of the GM(1,1) model is used as the prediction result before k_0, and the simulated value after the residual segment compensation is used as the prediction result after k_0. Then perform the error test again and put it into use.

4.3 **Adaptive Grey System Deep Neural Network**

The GS version of the GM(1,1) was a linear regression, in which it estimates two constant parameters, a and b, as seen from equation (4.2.3) [15]. The Adaptive Grey System Deep Neural Network changes a and b into functions of time, thus converting the linear constant parameters into non-linear functions. This transforms the GM(1,1) into a non-linear approach where a and b can be estimated by way of a feed forward deep neural network that is able to capture the non-linearity of the bitcoin price data.

We accomplished this by creating a system of three neural networks and three loss functions all working simultaneously and interconnectedly. The loss functions serve as the goal - what the network is supposed to optimize - where the outputs of the three neural networks are all contained.

$$DataLoss = \min(|x^0 - x_p^0| + |x^1 - x_p^1| + |z^1 - z_p^1|) \tag{4.3.1}$$

$$L1 = |\frac{dx_p^{(1)}}{dt} + a(t)x_p^{(1)} - b(t)| \tag{4.3.2}$$

$$L2 = |x_p^0 + a(t)z_p^{(1)} - b(t)| \tag{4.3.3}$$

Equation (4.3.1) intends to get the predictions as close to the actual values as possible and equations (4.3.2) and (4.3.3) satisfy the GM(1,1) differential equations.

With these loss functions and three, three-layer neural networks each with 60 units, the tanh activation function, and the adam optimizer, we have trained the AGS deep neural network with just 60 days of bitcoin data. The results of the trained model are shown in Chapter 5.

CHAPTER 5

RESULTS AND DISCUSSION

The bitcoin price data used to produce the following results was the daily price from June 13, 2017 to November 8, 2022 (1975 days) and it was obtained from Yahoo Finance. 1900 days were used to train the model, 60 for testing, and the last 15 days were used to validate predictions. We chose this split due to the volatile nature of bitcoin and the intention to make very short-term predictions. Making an accurate, short-term, 15 day prediction would be a success; however, since the validation set is so small, plotting the training loss vs validation loss proved fruitless and we did not include them. The data was also normalized before training, meaning, all values were shrunk down to be between 0 and 1. This allows for easier and more consistent computation among many other benefits. Figure (8) shows the entirety of the bitcoin price data set.

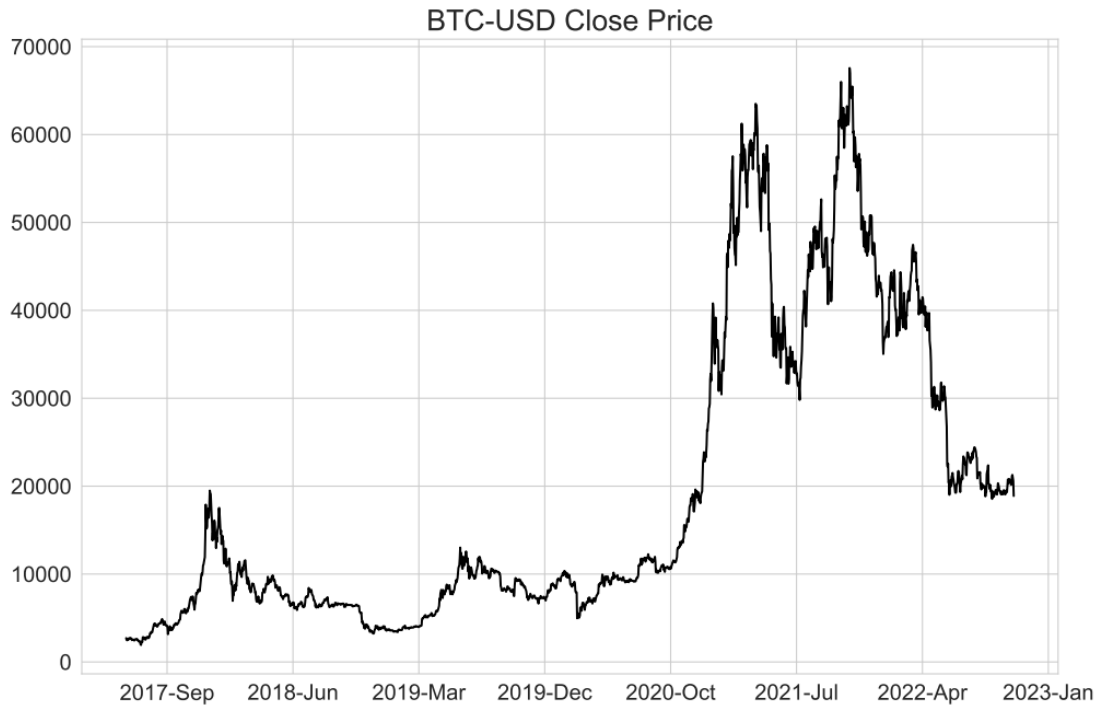

Figure 8: Bitcoin Daily Price Data

For the simple RNN, LSTM, and GRU models, a similar hyperparameter tuning process was conducted. This process involved an iterative increase in units followed by increasing layers. Units started at 50 then doubled and tripled for one, two, and three layer models. For two and three layer models, 50 percent dropout was applied to each layer followed by 10 percent. The same dropout experiment was also applied to the three layer model; however, switching between dropout between only one layer: the first or second. This experiment yielded positive results for the Simple RNN but not for the LSTM or GRU.

The reasoning behind iteratively increasing units and layers was simple. These increases in units and layers means the model, in theory, is able to expand the ability of the model to search for possible solutions in the hypothesis space. More gradients are calculated which means a deep search for the solutions. The reasoning behind the dropout experiments was to reduce the possibility of the models memorizing the patterns of the data rather than learning them. It is a regularization method intended to reduce overfitting.

The adam optimizer and the ReLu activation function was used across all experiments with the models set to perform 4 epochs. The loss function used was the Mean Absolute Error (MAE) which is calculated by $\frac{1}{N}\sum_{i=1}^{N}|Y_i - \tilde{Y}_i|$.

5.1 **Simple RNN Results**

As the number of units and layers increased, so did the performance of the simple RNN. The simple RNN model showed the best results with 100 units, 3 layers, and 10 percent dropout in the middle layer. Exceeding these numbers showed a significant decrease in performance along with increased computation time.

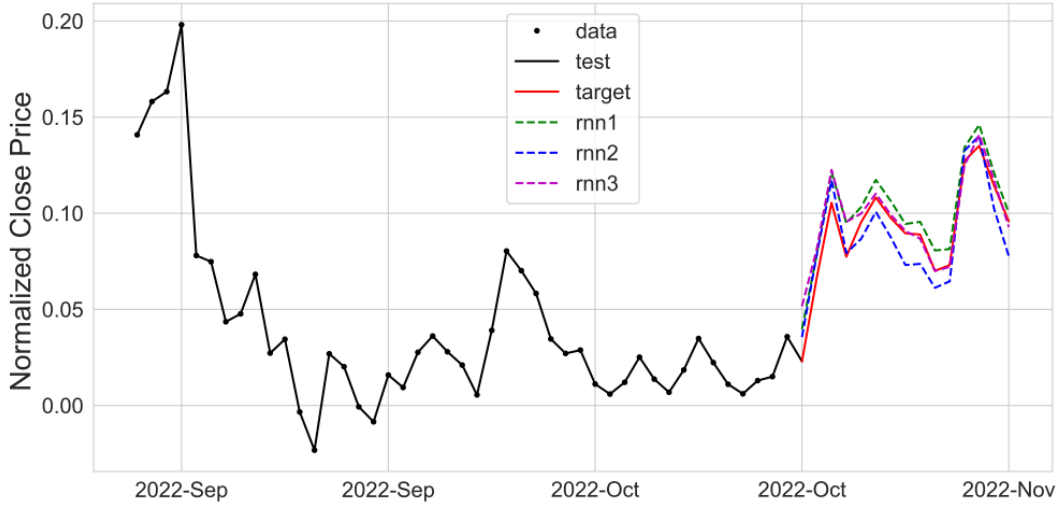

Figure 9: Simple RNN Results

```
# rnn3

simple_rnn_model_3 = Sequential()
simple_rnn_model_3.add(SimpleRNN(100, activation = 'relu',
    return_sequences = True, input_shape = x_train_uni.shape[-2:]))
simple_rnn_model_3.add(SimpleRNN(100, activation = 'relu',
    return_sequences = True))
simple_rnn_model_3.add(Dropout(0.1))
simple_rnn_model_3.add(SimpleRNN(100, activation = 'relu'))
simple_rnn_model_3.add(Dense(1))

simple_rnn_model_3.compile(optimizer='adam', loss='mae')
```

Figure 10: rnn3 Code

5.2 **Simple LSTM Results**

Neither dropout nor increased number of layers improved the simple LSTM performance. Only a single layer with 100 units showed good results, with little to no improvement beyond 100 units. Slight dropout improved the two layer model, but it was still not as strong as the single layer model's performance.

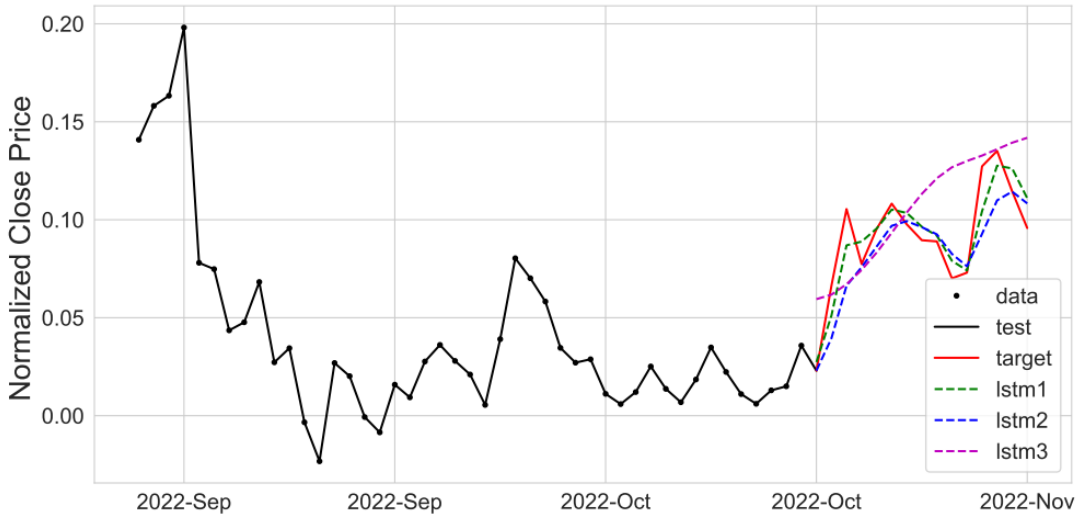

Figure 11: Simple LSTM Results

```
# lstm1

simple_lstm_model_1 = Sequential([
    LSTM(100, activation = 'relu', input_shape=x_train_uni.shape[-2:]),
    Dense(1)
])
simple_lstm_model_1.compile(optimizer='adam', loss='mae')
```

Figure 12: lstm1 Code

5.3 **Simple GRU Results**

Dropout did not improve the simple GRU models as it did with RNN and slightly with LSTM. Simply increasing the units made the GRU highly accurate with 150 units showing almost perfect prediction over the 15 day period as seen in Figure (13).

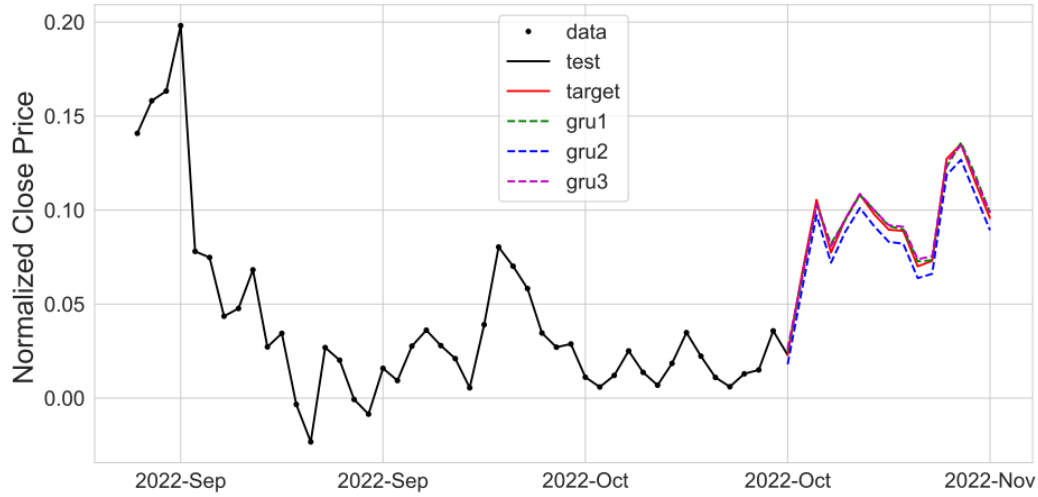

Figure 13: Simple GRU Results

```
# gru1

simple_gru_model_1 = Sequential([
    GRU(150, activation = 'relu', input_shape=x_train_uni.shape[-2:]),
    Dense(1)
])

simple_gru_model_1.compile(optimizer='adam', loss='mae')
```

Figure 14: gru1 Code

5.4 **GS Results**

Grey System Theory and the majority of associated models are linear in nature and Figure (15) shows the prediction a simple GM(1,1) model with no non-linear parameters over a very short period of time with a small dataset.

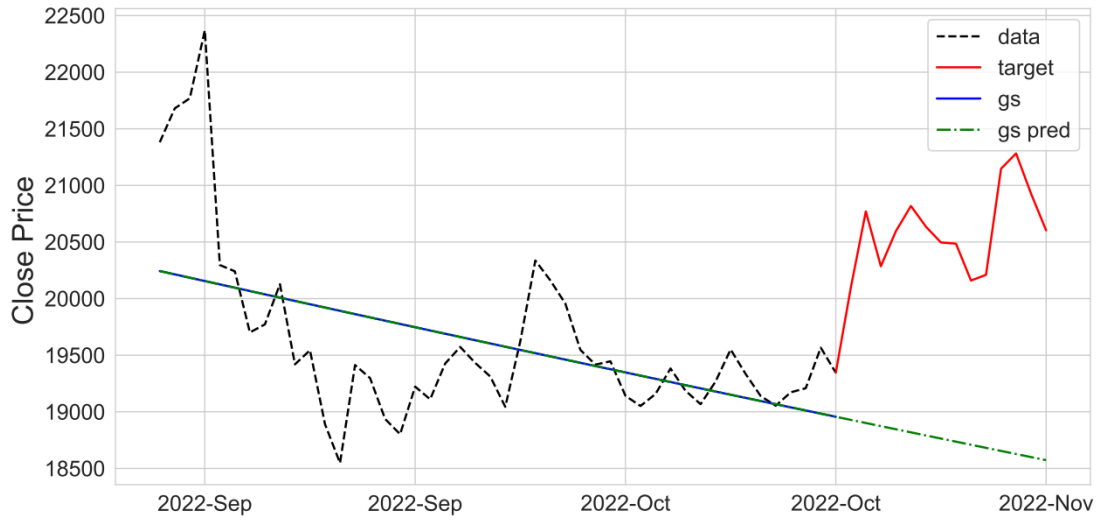

Figure 15: GS Results

5.5 **Adaptive GS Results**

Using the GM(1,1) construction steps discussed in Chapter 4, converted into Python code and run as a physics informed neural network, we were able to enable this linear method to learn non-linear grey system parameters. Figure (16) shows the results of the Adaptive GS deep neural network over a two month period.

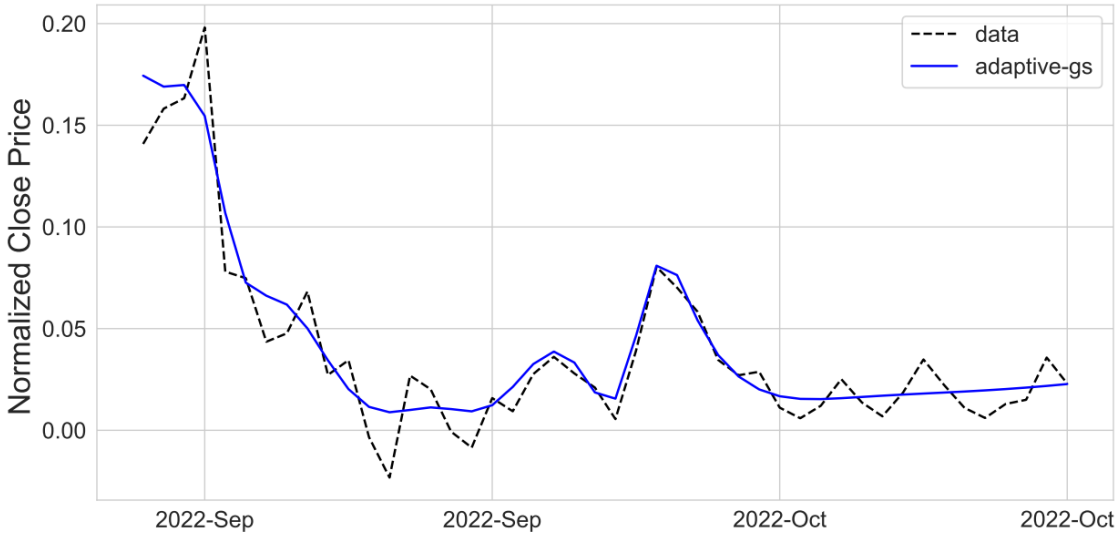

Figure 16: Adaptive GS Results

CHAPTER 6

CONCLUSION

The invention of bitcoin and the associated blockchain technology has had a profound impact on the world, especially the financial markets. No other asset has seen such returns in the entire history of investing. And as the old saying goes, what goes up must come down. Many investors have lost fortunes investing in this, and other, incredibly volatile asset.

Countless known and unknown factors contribute to the price of bitcoin; however, this research performs a purely quantitative and technical analysis of bitcoin price action in order to make short term predictions. Linear forecasting methods have been applied to bitcoin price along with non linear methods in more recent years. Neural networks are one such non linear approach to forecasting and this research utilizes the Recurrent Neural Network architectures - including Long Short Term Memory and Gated Recurrent Units - to perform predictions. Ultimately, a simple GRU model with 150 units performed the best in predicting the bitcoin price movement over a 15 day period.

Our work finds a marriage between forecasting bitcoin price, neural networks, and a linear method called Grey System Theory. The first order Grey Model, GM(1,1), is a linear system that learns constant parameters that can fit a small dataset and can also make reliable short-term predictions; however, in a high variance dataset like bitcoin price, the linear GM(1,1) has difficulty making reliable predictions[15]. Hence, our motivation to combine the linear GM(1,1) approach with deep neural networks. This enables the learning of non-linear grey system parameters, which then captures the non-linearity in the dataset. The resulting adaptive GS deep neural network trained on just 60 days, showed a well generalized trained model.

Future work may include building more complex and computationally intensive Recurrent Neural Networks to generate long term predictions while continuing to compare the results with more complex adaptive GS models with increased non-linear parameters. The

adaptive GS model will then be extended to produce predictions. These methods may also be applied to other predictive analytic fields with incomplete or limited data such as farming yields, energy consumption, and environmental sciences.

The code for the Simple RNN, LSTM, GRU, GS, and Adaptive GS can be found on github at the following link: https://github.com/ykhaliq2/MasterThesis.git

BIBLIOGRAPHY

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[2] Julong Deng. Introduction to grey system theory. *The Journal of Grey System*, 1(1):1–24, 1989.

[3] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1394–1401, 2018.

[4] M. Khedmati, F. Seifi, and M. J. Azizi. Time series forecasting of bitcoin price based on autoregressive integrated moving average and machine learning approaches. *International Journal of Engineering*, 33(7):1293–1303, 2020.

[5] Syed Abul Basher and Perry Sadorsky. Forecasting bitcoin price direction with random forests: How important are interest rates, inflation, and market volatility? *Machine Learning with Applications*, 9:100355, 2022.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[7] Leonardo Felizardo, Roberth Oliveira, Emilio Del-Moral-Hernandez, and Fabio Cozman. Comparative study of bitcoin price prediction using wavenets, recurrent neural networks and other machine learning methods. In *2019 6th International Conference on Behavioral, Economic and Socio-Cultural Computing (BESC)*, pages 1–6, 2019.

[8] Hakan Pabuçcu, Serdar Ongan, and Ayse Ongan. Forecasting the movements of bitcoin prices: an application of machine learning algorithms. *Quantitative Finance and Economics*, 4:679–692, 11 2020.

[9] Sean McNally, Jason Roche, and Simon Caton. Predicting the price of bitcoin using machine learning. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 339–343, 2018.

[10] Luisanna Cocco, Roberto Tonelli, and Michele Marchesi. Predictions of bitcoin prices through machine learning based frameworks. *PeerJ Computer Science*, 7:e413, 03 2021.

[11] Mingxi Liu, Guowen Li, Jianping Li, Xiaoqian Zhu, and Yinhong Yao. Forecasting the price of bitcoin using deep learning. *Finance Research Letters*, 40:101755, 2021.

[12] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., 2021.

[13] Jules Clément Mba, Sutene Mwambetania Mwambi, and Edson Pindza. A monte carlo approach to bitcoin price prediction with fractional ornstein-uhlenbeck levy process. *Forecasting*, 4(2):409–419, 2022.

[14] Mario Casillo, Marco Lombardi, Angelo Lorusso, Francesco Marongiu, Domenico Santaniello, and Carmine Valentino. Sentiment analysis and recurrent radial basis function network for bitcoin price prediction. In *2022 IEEE 21st Mediterranean Electrotechnical Conference (MELECON)*, pages 1189–1193, 2022.

[15] Junhang Duan, Ling Zhu, Wei Xing, Xi Zhang, Zhong Peng, and Huating Gou. Research on residual gm optimization based on pemea-bp correction. *Scientific Reports*, 10(1):2045–2322, 2020.

[16] Mahboubeh Faghih Mohammadi Jalali and Hanif Heidari. Predicting changes in bitcoin price using grey system theory. *Financial Innovation*, 6, 2020.

[17] John Chown and John F Chown. *A History of Money: from AD 800*. Psychology Press, 1994.

[18] P. J. Skevington and Tamara Paul Hart. Trusted third parties in electronic commerce. *BT Technology Journal*, 15:39–44, 1997.

[19] Murray Newton Rothbard. *History of Money and Banking in the United States: The Colonial Era to World War II, A*. Ludwig von Mises Institute, 2002.

[20] Usman Chohan. The double spending problem and cryptocurrencies. *SSRN Electronic Journal*, 01 2017.

[21] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International journal of web and grid services*, 14(4):352–375, 2018.

[22] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the works of leslie lamport*, pages 203–226. SRI International, 2019.

[23] Fang Fan, Carmine Ventre, Michail Basios, Leslie Kanthan, David Martinez, Fan Wu, and Lingbo Li. Cryptocurrency trading: a comprehensive survey. *Financial Innovation*, 8, 12 2022.

[24] Vinod. Nair and Geoffrey.E. Hinton. Rectified linear units improve restricted boltzmann machines. *ICML 2010*, pages 807–814, 2010.

[25] Nai-ming Xie and Si-feng Liu. Discrete grey forecasting model and its optimization. *Applied Mathematical Modelling*, 33(2):1173–1186, 2009.

[26] José Ramón San Cristóbal, Francisco Correa, María Antonia González, Emma Diaz Ruiz de Navamuel, Ernesto Madariaga, Andrés Ortega, Sergio López, and Manuel Trueba. A residual grey prediction model for predicting s-curves in projects. *Procedia Computer Science*, 64:586–593, 2015. Conference on ENTERprise Information Systems/International Conference on Project MANagement/Conference on Health and

Social Care Information Systems and Technologies, CENTERIS/ProjMAN / HCist 2015 October 7-9, 2015.