BITCOIN PRICE PREDICTION USING DEEP LEARNING

A THESIS SUBMITTED TO

THE FACULTY OF ACHITECTURE AND ENGINEERING

OF

EPOKA UNIVERSITY

BY

KLEA XHIXHO

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR

THE DEGREE OF MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

JUNE ,2020

Approval sheet of the Thesis

This is to certify that we have read this thesis entitled "Bitcoin price prediction using deep learning "and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Dr. Ali Osman TOPAL

Head of Department

Date: June 26,2020

Examining Committee Members:

Dr. Ali Topal (Computer Engineering) _____

Assoc. Prof. Dr. Dimitros A. Karras (Computer Engineering) _____

Dr. Maaruf Ali (Computer Engineering) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name Surname: Klea Xhixho

Signature: _____

ABSTRACT

BITCOIN PRICE PREDICTION USING DEEP LEARNING

Xhixho, Klea

M.Sc., Department of Computer Engineering

Supervisor: Prof. Dr Dimitrios Karras

Bitcoin is a sort of computerized cash in which a record of exchanges is kept up and new units of cash are produced by the computational arrangement of scientific issues, and which works autonomously of a central bank. The price of a bitcoin is equal to 7000 USD and is growing more day by day. Since this is not a low price, approximately 10 million people are investing on it. This would be exceptionally curiously for investors to figure the Bitcoin esteem but the same time making it troublesome to predict. This study centers on predicting daily Bitcoin prices using some models (MLP, LSTM, CNN, GRU, ARIMA) of deep learning. In each of these models we will interpret the train and test loss, mean squared error, the performance in time and in the end predicted prices. After interpreting all this, we will compare all these results between different models.

Keywords: *Bitcoin, machine learning, LSTM, CNN, GRU, RMSE, MLP, ARIMA*

ABSTRAKT

PARASHIKIMI I ÇMIMIT TË BITCOIN DUKE PËRDORUR DEEP LEARNING

Xhixho, Klea

Master Shkencor, Departamenti i Inxhinierisë Kompjuterike

Udhëheqësi: Prof. Dr Dimitrios Karras

Bitcoin është një monedhë virtuale e cila funksionon në mënyrë autonome nga banka qëndrore. Çmimi i një bitcoin është i barabartë me 7000 USD dhe po rritet më shumë nga dita në ditë. Meqenëse ky nuk është një çmim i ulët, rreth 10 milion njerëz janë duke investuar në të. Ata janë të interesuar në parashikimin e çmimit ditor të një bitcoin por kjo është bërë disi e vështirë pëe tu realizuar. Për këtë arsye ky studim përqendrohet në parashikimin e çmimeve ditore të Bitcoin duke përdorur disa modele të Deep Learning (MLP, LSTM, CNN, GRU, ARIMA). Në secilin prej këtyre modeleve ne do të interpretojmë humbjen që do të pësojë çmimi gjatë proceseve të trajnimit dhe testimit, gabimet e bëra gjatë parashikimit, performancën në kohë dhe në fund çmimet e parashikuara. Pas interpretimit të gjithë kësaj, ne do t'i krahasojmë të gjitha këto rezultate midis modeleve të ndryshme.

Fjalët Kyçe: *Bitcoin, LSTM, CNN, GRU, RMSE, MLP, ARIMA*

ACKNOWLEDGEMENTS

I would like to express my special thanks to my supervisor Prof. Dr. Dimitrios Karras for his continuous guidance, encouragement, motivation, and support during all the stages of my thesis. I sincerely appreciate the time and effort he has spent to improve my experience during my graduate years.

TABLE OF CONTENS

LIST OF FIGURES

LIST OF ABBREVIATIONS

BTC	→ Bitcoin Core
SVM	→ Support Vector Machine
MLP	→ Multilayer Perceptron
CNN	→ Convolutional Neural Networks
GASEN	→ Genetic Algorithm based Selective Neural Network Ensemble
LSTM	→ Long Short-Term Memory
ResNet	→ Residual Neural Network
DNN	→ Deep Neural Network
RNN	→ Recurrent Neural Network
ARIMA	→ Autoregressive Integrated Moving Average
GPU	→ Graphics Processing Unit
CPU	→ Central Processing Unit
NNAR	→Neural Network Autoregression
GRU	→ Gated recurrent units
ReLU	→ Rectified Linear Unit
MSE	→ Mean Squared Error

CHAPTER 1

INTRODUCTION

The first chapter in the first section describes the research background. In that part I have talked about the history of Bitcoin, the creation of Bitcoin, how does it work, what is blockchain and how does it work, Bitcoin value price, Bitcoin trading platforms, prediction and Deep Learning and how does it work. In the second and third give the purpose and goals of the thesis, respectively. The fourth section follows up the related work that I took into consideration when I start working out with this paper.

1.1 Research Background

1.1.1 A brief history about Bitcoin

Milton Friedman, economist, and Nobel laureate had foreseen a digital currency coming as early as 1999, and the possible impact it would have on the global economy as a whole. Nearly 10 years later, he saw daylight in his dream. In November 2008, Bitcoin first came into the world through an article called "Bitcoin: A Peer-to-Peer Electronic Cash System" written by Satoshi Nakamoto, under what is considered to be a pseudonym. It reflects the first digital currency in the world and the first peer-to-peer payment system. It was probably Introduced as answer to the 2008 financial collapse, and since the end of 2010, Satoshi himself has not been seen. The network is autonomous so, it is run by its users. It is opensource, meaning that the software can be downloaded and distributed for any purpose by everyone. Unlike a typical fiat currency, the aim of Bitcoin was to create a digital currency, independent of third-party financial institutions' involvement. Instead, transactions between network users would be direct electronically and instantly.

The network is not directly influenced by monetary policies and political decisions, thus free from the potential instabilities (such as inflation) that could result from this.

While many say this is the benefit of Bitcoin, others say it is a big downside. Since it is supposed to be free of political interference, it is likely, for example, that political decisions

affect the demand for bitcoins and ultimately affect the price. Chinese decision, for example, to bar financial institutions and payment institutions from doing business associated with bitcoins. Bitcoins are treated as a cryptocurrency. Every bitcoin is divisible to the eighth decimal and is called one Satoshi. To put it another way, 0.00000001 BTC=1 Satoshi. Since there is no financial institution, such as a central bank that controls the currency, Bitcoin relies on customized cryptography to regulate bitcoin creation. While cash is entirely anonymous, however, Bitcoin transactions are traceable and linked to specific accounts. If one can identify this account's ownership, one can also see every historic transaction that was made with this account. A better description would, therefore, be pseudonymous, rather than anonymous.

1.1.2 The creation of Bitcoins

The development process of bitcoins is called mining. Since the network is self-sustaining, it must fully function on its own and it must be validated by the system each time a transaction is made before the transaction can be considered complete. The software is coded to reward those who provide the computer power needed to verify the transactions – the "miners".

This validation occurs in blocks, where every single block is a set of transactions that await verification.

The software is designed to successfully add one block to the overall blockchain every ten minutes. This blockchain is completely critical to remove double-spending and the falsified bitcoin circulation. Since every transaction is registered and processed, the system must always be aware of where each bitcoin is. But just mining does not reward you. Only the miner who does manage to successfully process the block is rewarded by doing so with newly created bitcoins. This is done by solving a complex math problem, which every two weeks becomes more difficult. It is like the exploitation of natural resources makes them scarcer and harder for others to find. As the mathematical problems became more difficult, more advanced computers were needed to solve them and a "normal" computer is nowadays insufficient for this, unlike at the beginning.

1.1.3 How does Bitcoin work?

Bitcoin processing methods are incredibly complex, so we are going to try to skim the surface of that. Each Bitcoin user receives two keys, one private and one public. You may define the public key (your "Bitcoin address") as your bank account, and the private key as your bank card reader, enabling you to authenticate an online transaction. In essence, the public key can be described as a glass deposit box: everyone can see exactly what is inside, but only the person with the right key (the private key) can use the money inside.

The whole network is a large zero-sum game, so the system is set up to keep track of all previously made historical transactions (i.e., all historical deposits and withdrawals from any existing Bitcoin account). In a sense, one does not transfer the bitcoin itself when transferring a bitcoin, but rather Bitcoin ownership.

It is very similar to money during a gold standard, when gold certificates could be exchanged multiple times, without affecting the actual gold itself. Money in your bank account is not actual money either, but there is a confidence that those numbers are actual money. The money is never seen when transferring funds to someone else, yet the bank registers and keeps track of the transaction, and therefore the same transaction cannot be made twice. This is how the Bitcoin network works too.

Bitcoin is visible. All transactions are publicly available, so “everybody is monitoring everybody” and a transaction is not fully completed unless the whole system agrees on the transaction. There must be agreement on the exact history of each bitcoin; where it was previously, and where it is now. Once the transaction is complete it cannot happen again as the system will react and deny it unanimously. A Network transaction is irreversible, much like a cash transaction. It is allocated with a combination of many components to make each transaction unique and non-replicable. This is called hashing, which simply involves creating a unique code, using the coin's knowledge history (i.e. where it was previously) which adding it to the new transaction. The system uses a mathematical algorithm to do this, and the next time the coin is transferred, the new hash will incorporate information from all its old addresses as well as from the current one.

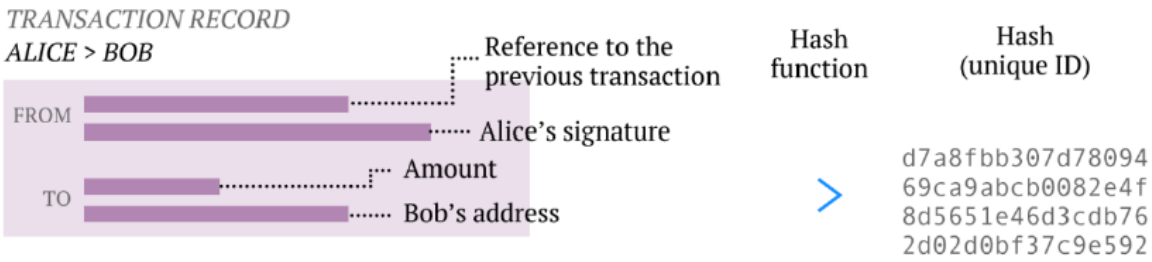


Figure 1. The transaction of Bitcoin

Since the system is always up-to-date and the latest hash is registered as valid, one can does not cheat it. Attempting to do this (e.g. by attempting to transfer a bitcoin that has been transferred before) would result in the machine telling you metaphorically: "This coin is not legitimate since it is no longer in your possession." If there is an attempt to move the same funds to two separate accounts concurrently, the network would mark the first one submitted as "on hold" before it can be added to the blockchain in full.

1.1.4 What is blockchain and how it works?

Money exists to facilitate trade. Through the centuries, trade has ended up incredibly complex. Everybody exchanges with everybody, worldwide. Trade is recorded in bookkeeping. This data is regularly confined and closed to the public. That is why, we utilize third parties and middlemen we trust to encourage and endorse our exchanges. Think of governments, banks, bookkeepers, notaries, and the paper money in your wallet. We call these 'Trusted Third Parties'. This brings us to the pith of Bitcoin. Bitcoin's program empowers an arrangement of computers to keep up collective bookkeeping through the web. This bookkeeping is public and accessible in one computerized ledger which is completely distributed over the network. We call this the Blockchain.Within the Blockchain, all exchanges are logged counting information on the date, time, members, and amount of every single transaction. Each node within the arrange claims a full duplicate of the Blockchain. Based on complicated state-of-the-art numerical principles, the exchanges are confirmed by the so-called Bitcoin Miners, who keep up the ledger. The scientific standards also guarantee that these nodes

automatically and persistently agree about the current state of the record and each exchange in it.

In case anybody attempts to degenerate an exchange, the nodes will not arrive at a consensus and subsequently will deny joining the exchange within the Blockchain. So, each exchange is public, and thousands of nodes collectively agree that an exchange has occurred on date X at time Y. It is like there is a public accountant display at each transaction. This way, everybody has access to a shared single source of truth. That is why we can continuously believe Blockchain. The ledger does not care whether a bitcoin speaks to a certain quantity of euros or dollars or anything else of esteem or property for that matter. Users can choose for themselves what a unit of bitcoin represents.

1.1.5 Bitcoin trading platforms

There are many online bitcoin exchanges today. A few major actors such as Binance, Huobi Global, and Coinbase Pro currently dominate the market, but there are plenty of other trading platforms available.

In this thesis, I have chosen to explore the platform called Binance. It is Bitcoin 's largest and fast-growing exchange that has built the largest ecosystem around crypto-powered finance. Using a debit/credit card, you can buy bitcoin on this platform, then trade it for other coins. The platform also provides a fully functional mobile app, and this is by far the most widely used Bitcoin exchange in the world. It has evolved tremendously since its ICO to date and is now ranked in # 1 in the world's top 10 Bitcoin exchanges. Binance provides the largest bitcoin marketplace. Being a centralized exchange has taken a unique approach to expand its business and also gives day traders a decent discount. You need to register using your email ID to get started with Binance and the process is pretty simple & fast. Binance is offering a native coin called BNB which is probably the other crypto you could HODL for longer periods. The fee structure of this platform is special too.

	1st year	2nd year	3rd year	4th year	5th year
Discount Rate	50%	25%	12.5%	6.75%	no discount

Figure 2. Binance's fee structure

To begin with, they have a standard trading fee of 0.1 percent that is already quite smaller than other peers. You will get a 25 percent refund on your trading operation if you keep the BNB token. This exchange has more benefits, such as:

- Bitcoin saving account: Bitcoin gets interested
- Exchange futures and margins
- Throwing
- No KYC for every day 2 BTC withdrawal

1.1.6 Bitcoin value and price

A bitcoin is divisible in 100 million units. And each unit is both independently identifiable and programmable.

This implies that users can assign properties to each unit. Users can program a unit to speak to a Eurocent, or a share in a company, a kilowatt-hour of energy, or a digital certificate of ownership. Because of this, Bitcoin is much more than essentially cash and payments: A Bitcoin can represent many kinds of property. A thousand barrels of oil, award credits, or a vote during elections,for example. According to CoinMarketCap, the esteem of all the bitcoins within the world was $160.4 billion as of March 4, 2020. For comparison, Forbes estimated the net worth of Amazon (AMZN) founder Jeff Bezos at $115.5 billion. That produces the market cap of Bitcoin just over a third bigger than Bezos' fortune. Since this is not a low price, approximately 10 million people are investing in it. This would be exceptionally curiously for investors to figure the Bitcoin esteem but at the same time making it troublesome to predict.

1.1.7 Prediction

In reality, no one can predict a cryptocurrency 's future but if we could, we would all be billionaires. Prices in the world of cryptocurrency are very volatile. This means a coin 's value will go up or down very quickly, sometimes with no reason as to why. That makes it far more difficult to predict prices than traditional markets. There are many algorithms used on stock market data for a price forecast. However, the parameters affecting Bitcoin are different. Therefore, it is necessary to foretelling the Bitcoin value so that correct investment decisions can be made. The price of Bitcoin does not depend on the business events or intervening government authorities, unlike the stock market. Thus, to forecast the value we feel it is necessary to leverage machine learning technology to predict the price of Bitcoin.

1.1.8 Deep learning

Deep learning [1] is a technique of machine learning, teaching computers to do what comes naturally to humans: learn by example. It is a key technology behind driverless cars, enabling them to identify a stop sign or discern a pedestrian from a lamppost. Deep learning is the secret to voice control in consumer devices such as phones, laptops, televisions, and hands-free orators.

In deep learning a computer, the model learns directly from images, text, or sound to perform classification tasks. Deep learning models can achieve cutting-edge precision, often exceeding output at the human level. Models are trained using a wide collection of labeled data and architectures of neural networks, which include several layers.

In a word, precision. Deep learning achieves accuracy in identification at levels higher than ever before. This helps consumer electronics meet user expectations and is critical for safety-critical applications such as driverless automobiles. Recent advances in deep learning have improved to the point where deep learning outperforms humans in certain tasks, such as image classification of objects.

Though deep learning was first theorized in the 1980s, it has only recently become useful for two key reasons:

1. Deep learning involves large quantities of labeling data. For example, the creation of driverless cars involves millions of images and thousands of hours of video.
2. Deep learning calls for considerable computing power. High-performance GPUs have a parallel architecture that is effective for deep learning. Combined with clusters or cloud computing, this allows development teams to shorten training times from weeks to hours or less for a deep learning network.

1. How Deep Learning works?

Sometimes we refer to the models of deep learning, like deep neural networks. This is because most deep learning approaches use neural network architecture. “Deep” is the notion typically referring to the number of hidden layers within the neural network. The number of hidden layers covering a traditional network is normally 2-3, although deep networks may be as high as 150. Deep learning models are trained using large sets of labeled data and architectures in the neural network that learn features directly from the data without the need for manual extraction.

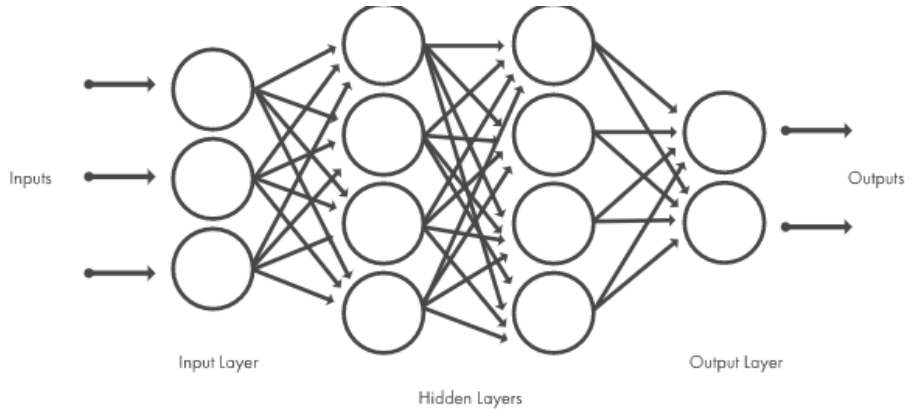


Figure 3. Neural Network which are organized in layers consisting of a set of interconnected nodes

One of the most popular types of deep neural networks is known as neural convolution (CNN or ConvNet). A CNN combines learned features with input data and uses 2D

convolution layers to make this architecture suitable for 2D data processing, such as images. This model removes the need for manual extraction of the element, so you do not have to recognize the features used to classify images. It operates by directly extracting features from pictures. The related features are not pre-trained; while the network trains on a set of pictures, they are taught.

This automated extraction of a feature makes deep learning models highly accurate for computer vision tasks such as classification of objects. CNN's learn to use tens or hundreds of hidden layers to detect different features of an image. Each hidden layer increases the complexity of features learned in the image. For example, the first hidden layer might learn how to detect edges, and the last one would learn how to detect more complex shapes specifically tailored to the shape of the object we are trying to recognize.

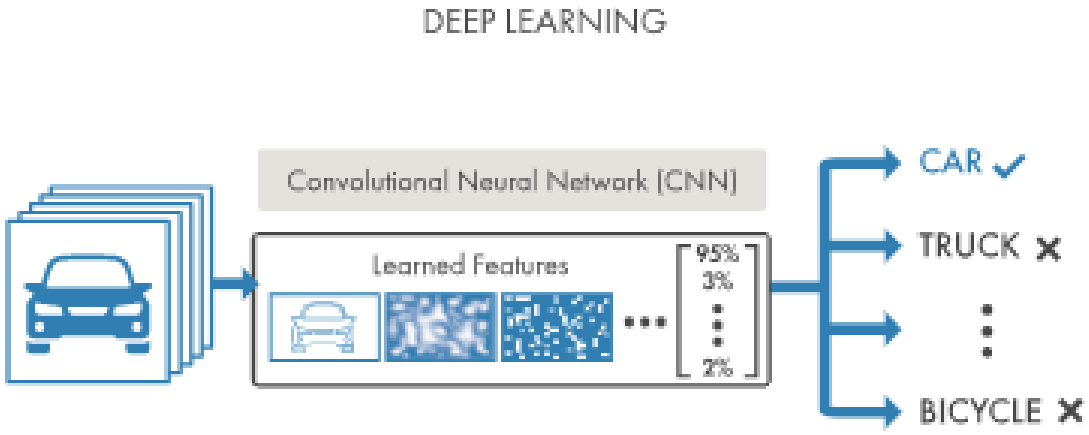


Figure 4. An example for categorizing vehicles using deep learning

1.2 Purpose

The main purpose of this thesis is to see the evaluation of Bitcoin price using some traditional neural networks and some time-series models and later to compare the results of each model with each other.

1.3 Goals

The goal of this thesis is to be able to implement all models that we can make a straight comparison between each of them. The main focus of this work is going to be directed especially in those perspectives:

- Implementing all the neural networks and time series models.
- Being able to find the time and mean squared error for each of them and compare the conclusion between each-other.

1.4 Related work

1.4.1 Using Bitcoin Ledger Network Data to Predict the Price of Bitcoin

The first paper that I took into consideration was "Using Bitcoin Ledger Network Data to Predict the Price of Bitcoin" by John Mern**,** Spenser Anderson and John Poothokaran. [2] They have divided this study into two phases. In the first phase, multiple model types are compared by them for determining the best choice for further optimization. For doing that, they have used logistic regression and SVMs (support vector machines) and deep-learning approaches (MLP, CNN). In the second phase, they optimized the training process for the highest performing model, and they evaluated the final performance of this model. As is shown in the figure below, they concluded that the CNN model had the best classification accuracy and also the best regression performance. After they found the highest performing model they were focused only on that model. They extended through June 2014 the training set and through December 2014 the test set. In this way, they used June 2014 as the validation set to perform a hyper-parameter sweep over model training parameters. 59.7% was the accuracy throughout the entire test set, and during the first month of the test set, the accuracy was 66.7%. They concluded that the convolutional neural network could generate a predictive model of Bitcoin price that generalized reasonably well to market conditions.

Model	Validation Accuracy
Logistic Regression	0.653
SVM	0.673
MLP	0.684
CNN	0.967

Figure 5. Validation Accuracy of Logistic Regression, SVM, MLP and CNN models

1.4.2 Bitcoin Price Prediction Using Ensembles of Neural Networks

The second paper that I took into consideration was "Bitcoin Price Prediction Using Ensembles of Neural Networks". [3] The relationship between the features of Bitcoin and the next day change in the price of Bitcoins are the topics that this paper explores. All this thing I mentioned above is made by using an Artificial Neural Network ensemble approach called the Genetic Algorithm which is constructed by means of Multi-Layered Perceptron. They have considered a set of 200 features of the cryptocurrency over a span of 2 years for the Bitcoin price. For the prediction of the next day direction of the Bitcoin price, they have used the ensemble. Over a span of 50 days, they also have compared a trading strategy based on the ensemble with a "previous day trend following" trading strategy. In that comparison, the trading strategy based on GASEN had a higher value than "the previous day trend following". This is shown in the figure below. They also have tested the dataset with the MLP model in the ensemble. As is shown in the figure below, the MLP model in the ensemble did not perform as well as the ensemble. They concluded that GASEN was the best for the classification task. Its accuracy consists of 58% to 63%. And with a simple trading strategy, this percentage will rise close to 85%.

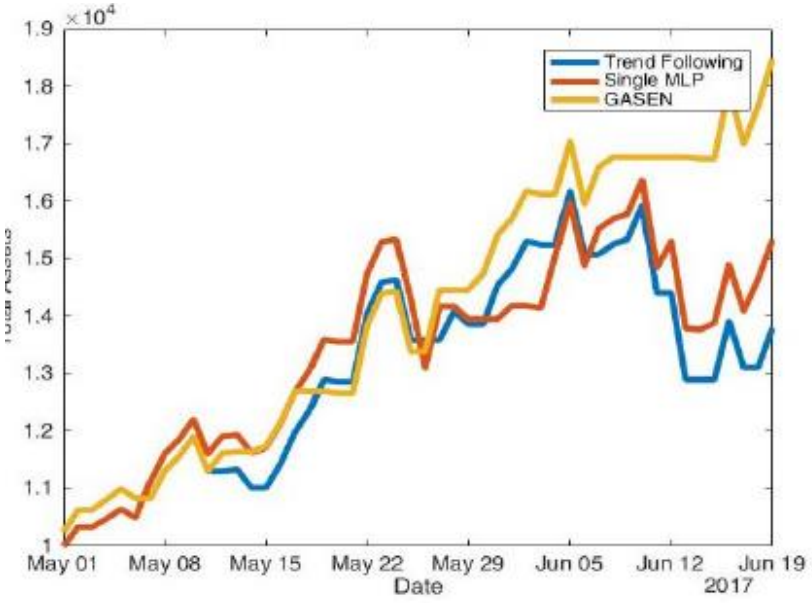


Figure 6. Total Asset Value over Time

1.4.3 A Comparative Study of Bitcoin Price Prediction Using Deep Learning

The third article that I took into consideration was "A Comparative Study of Bitcoin Price Prediction Using Deep Learning" by [4]. This article was published on 25 September 2019. Authors have used various deep-learning models for predicting Bitcoin prices. Some of those models are long short-term memory (LSTM), deep residual networks (ResNet), deep neural network (DNN), convolutional neural networks (CNN), and some combinations from them. Firstly, they have used regression to predict the future Bitcoin price and after they have used classification to see if the future price will go up or down. For the first part, the LSTM was the model that performed higher than others. And for the second one was DNN. Despite this, they concluded that none of the models was the winner. According to them, all the models were comparable to each other.

1.4.4 Prediction of Bitcoin Price using Data Mining

The fourth paper that I took into consideration was "Prediction of Bitcoin Price using Data Mining" by Dharminder Singh Virk [5]. This paper was published by the School of Computing National College of Ireland. This paper with the mean of data mining predicts the

accuracy of Bitcoin price. The author has created a new dataset from ten cryptocurrencies datasets which has a strong correlation with Bitcoin. For finding out the best accuracy he has used Support Vector Classifier, Neural Network Classifier, Random Forest Classifier, and Gradient Boost Classifier. And he concluded that the highest accuracy was reached by the support vector classifier. This paper has used Recurrent Neural Network, Gradient Boosting Regressor, and Recurrent Neural Network methods in regression. Here, the highest value of R-squared was reached by Gradient Boosting Regressor.

1.4.5 Prediction of Bitcoin using Recurrent Neural Network

The fifth paper that I took into consideration was "Prediction of Bitcoin using Recurrent Neural Network" [6]. This paper has used Random Forest and Long Short-Term Memory RNN algorithms for Bitcoin price prediction. After that, the author has made a comparison between these algorithms. This comparison is made by using different sizes and amounts of data. He also has considered the complexity of the database and also the time and the speed. In the end, he concluded that for a complex database the best prediction is made by LSTM RNN.

Method	Accuracy
Random Forest	84%
LSTM RNN	92%

Figure 7. Price Prediction for Bitcoin using Random Forest and LSTM RNN

1.4.6 Comparative study of Bitcoin price prediction using WaveNets, Recurrent Neural Networks and other Machine Learning Methods

The sixth paper that I took into consideration was "Comparative study of Bitcoin price prediction using WaveNets, Recurrent Neural Networks, and other Machine Learning Methods" by Leonardo Felizardo, Roberth Oliveira, Emilio Del-Moral-Hernandez and Fabio

Cozman. [7]. For predicting Bitcoin price this article has used different methodologies like LSTM, ARIMA, SVM, Random Forest, and WaveNets. According to this paper when the prediction gap increases all models performed badly. This was indicated by error metrics. Models performed equally badly for long-term predictions too. This showed that the accuracy was affected more from random components. According to these results, the authors concluded that a linear model is more efficient in a time-series that has a random component.

1.4.7 Bitcoin Price Prediction with Neural Networks

The seventh paper that I took into consideration was “Bitcoin Price Prediction with Neural Networks” by Kejsi Struga and Olti Qirici . [8] This paper has used the LSTM version of RNN for predicting Bitcoin price. They have predicted the Bitcoin price for 30 and 60 days ahead. They concluded that using the deep neural network has made us understand better Bitcoin and LSTM architecture.

1.4.8 Predicting the price of Bitcoin using Machine Learning

The eighth paper that I took into consideration was “Predicting the price of Bitcoin using Machine Learning” by Sean McNally. [9] It was presented at the School of Computing National College of Ireland on 9t September 2016. This paper aimed to find out with what precision Bitcoin price direction can be predicted. To archive this goal the author has implemented RNN and LSTM models. The highest accuracy has archived by LSTM. For a comparison to a deep learning model, the author has implemented the ARIMA, a time series forecasting model. Which performed poorly in comparison with the deep learning models. And in the end, this paper has benchmarked the RNN and LSTM on a GPU and a CPU. The author has chosen the same batch size and the same temporal length for both models. He concluded that in terms of overall training time, the LSTM trained faster on the GPU then the RNN. This is shown in the table underneath.

Model	Epochs	Intel Core i7 6700 2.6GH\z	NVIDIA GeForce 940M 2GB
RNN	50	56.71s	33.15s
LSTM	50	59.71s	38.85s
RNN	500	462.31s	258.1s
LSTM	500	1505s	888.34s
RNN	1000	918.03s	613.21s
LSTM	1000	3001.69s	1746.87s

Figure 8. Performance Comparison

1.4.9 Bitcoin price prediction using Deep Neural Networks

The ninth paper that I took into consideration was “Bitcoin price prediction using Deep Neural Networks” by Michelle Appel. [10] It was presented at the Artificial Intelligence University of Amsterdam in June 2016. This paper has predicted Bitcoin's price through the LSTM algorithm. The author has made an experiment to see the effect of different feature combinations. For this experiment, he found a certain optimal combination of 9 features. After that, the author has used different prediction delays and different sequence lengths for testing the effectivity of prediction. According to those tests he found out that a prediction delay of 0 and a sequence length of 1 can get the best absolute prediction.

1.4.10 Next-Day Bitcoin Price Forecast

The tenth paper that I took into consideration was “Next-Day Bitcoin Price Forecast” by Ziaul Haque Munim, Mohammad Hassan Shakil, and Ilan Alon. [11] It was published at Risk and Financial Management on 20 June 2019. This paper has used Neural Network Autoregression (NNAR) and Autoregressive Integrated Moving Average (ARIMA) model for analyzing forecasts of Bitcoin price. For doing that the authors have used both with and without re-estimation of the forecast model for each step. They have used two different training and testing samples for cross-validation. They concluded that the best performance in the first training sample has archived by NNAR while the ARIMA performed better in the

second training sample. In the two test-sample forecasts ARIMA performed better and this performance was identical for ARIMA models with and without re-estimation.

CHAPTER 2

PREPROCESSING

The second chapter, in the first section, describes the dataset. In the second section, I have used some exploratory data analysis like feature correlation, stationarity, autocorrelation, and partial correlation, to explain our dataset. The third section describes the preprocessing of our dataset. The preprocessing has done in some steps like finding the null values, encoding categorical data, splitting the dataset into training and test sets, and feature selection.

2.1 Dataset

For doing this study is needed a dataset with prices of Bitcoin during the last years. So, I have chosen a dataset that contains this information from 2011 to 2020. [12]

	Date	Bitcoin Core (BTC) Price	Money Supply	Price Volatility	Daily Transactions	Block Size	Transaction Fees	Inflation Rate
0	6/2/2011	0.89	5321500.0	164.844710	876	1586	0.178263	174.722
1	7/2/2011	0.89	5330350.0	164.987700	861	1685	0.101126	173.646
2	8/2/2011	0.88	5340500.0	165.328338	888	1316	0.315472	172.767
3	9/2/2011	0.91	5348900.0	164.880682	879	1615	0.177370	171.759
4	10/2/2011	1.03	5358500.0	167.764321	1246	1973	0.065517	171.090

Figure 9. Sample from the head of the dataset

	Date	Bitcoin Core (BTC) Price	Money Supply	Price Volatility	Daily Transactions	Block Size	Transaction Fees	Inflation Rate
3328	18/03/2020	5295.56	18275387.5	133.018137	279400	1233985	62.618105	3.84473
3329	19/03/2020	5325.03	18276637.5	133.331181	233312	1167475	60.566768	3.83996
3330	20/03/2020	6016.76	18277975.0	141.060454	265650	1303697	68.826596	3.83701
3331	21/03/2020	6271.22	18279337.5	142.316516	282961	1267319	79.237573	3.83288
3332	22/03/2020	6181.23	18280887.5	142.315337	238483	1072780	44.318007	3.83100

Figure 10. Sample from the tail of the dataset

My dataset has 3333 rows and 8 columns. Let me explain the columns:

1. **Bitcoin Core Price BTC** → Price of Bitcoin.
2. **Money Supply** → The Bitcoin Core amount in circulation.
3. **Price Volatility** → The annualized price volatility changes daily. The volatility of price is measured as the standard deviation in regular returns, multiplied to annualize by the square root of 365, and expressed as a decimal.
4. **Daily Transactions** → The number of transactions included in the everyday blockchain.
5. **Block size** → Bitcoin Core (BTC) transactions are collected by miners into separate data packets, called blocks. Each block is cryptographically linked to the previous block, creating a "blockchain." As more people use the Bitcoin Core (BTC) network for Bitcoin Core (BTC) transactions, the size of block increases.
6. **Transaction Fees** → Total amount of Bitcoin Core (BTC) fees, measured in Bitcoin Core (BTC), earned by all miners within a 24-hour period.
7. **Inflation rate** → The federal fund rate will decide the shape of the economy's future interest rate.

2.2 Exploratory data analysis

And now let us have a look in the picture below. There is a graph that is created from our dataset and presents the changes in Bitcoin price over past years. Like we see above from 2011 to 2016 the price of Bitcoin is almost 0. At the beginning of 2017, the price of Bitcoin has started to increase. The end of 2017 reached the highest price for our dataset. At the beginning of 2018, the price of Bitcoin has decreased again. Through this year the price was increased and decreased very often. In the middle of 2019, the price had a second higher price for our dataset. But at the beginning of 2020, the price dropped again. How we can see in this illustration the price of Bitcoin changes very often. For that reason, it is important to predict it. This thesis aims to do that prediction for the following months of 2020.

Figure 11. Bitcoin price shape through time

2.2.1 Features correlation

Data and correlation of features is considered an important step in the data preproces sing selection phase, this happens especially for continuous data types of the features. So, what is data correlation? Data Correlation: Is the way how we understand the relation in your dataset between multiple variables and attributes; You can get some insights by using Correlation, like:

- One or more attributes are contingent upon another attribute or trigger of another attribute.
- It combines one or more attributes with other attributes.

So, what are the beneficials of Correlation?

- Correlation may help to predict one attribute from another (Great way of imputing missing values).
- The correlation can (sometimes) indicate that a causal relationship exists.
- For many modelling techniques, correlation is used as a basic quantity

Let us take a closer look at what this means, and how useful correlation can be. There are three correlations:

1. Positive correlation: means that if we see increases of feature A, feature B will increase as well or if we see decreases of feature A, feature B will decrease as well. The relationship is linear.
2. Negative correlation: means that if we see increases of feature A, the feature B will decrease and vice versa.
3. Any Correlation at all: These two attributes are not related.

Each of these types of correlation may exist in a spectrum represented by values from 0 to 1. If values are 0.5 or 0.7 the correlation is highly positive. For a strong and perfect positive correlation, then a correlation score value of 0.9 or 1 will represent the result. If there is a strong negative correlation, then a value of -1 will represent it.

In our Bitcoin price prediction study, we are interested to show the relationships of the Bitcoin price variable with other variables that are part of our dataset. The figure underneath shows that.

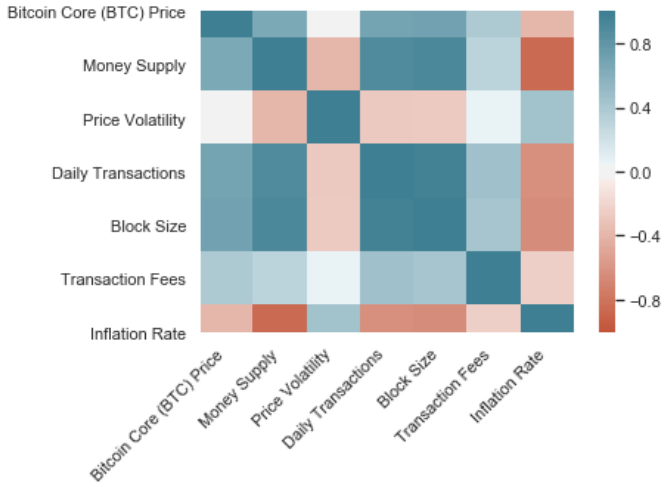


Figure 12. The correlation between the features and the Bitcoin price

For explaining the figure above, we need to know the meaning of each color that is shown in it.

1. Blue → High positive correlation
2. Red → High negative correlation
3. White → No correlation between two features

As is shown at the figure the Bitcoin price variable has a positive correlation with four features which are money supply, daily transactions, block size, and transaction fees. Three first features have a dark blue that means that the correlation between them and Bitcoin price is higher. The fourth one has a lighter blue which means that the correlation with Bitcoin price is lower. Our variable has a negative correlation with the inflation rate and no relation with price volatility.

2.2.2 Stationarity

A time series is stationary when one of its statistical properties are all constant over time, such as mean, variance, autocorrelation, etc. Most statistical forecasting methods assume that mathematical transformations can render the time series approximately stationary (i.e., "stationarized"). A stationary series is relatively easy to predict: you simply predict that in the future, its statistical properties will be the same as they were in the past. The predictions for the stationaries series can then be "untransformed" to obtain predictions for the original series by reversing whatever mathematical transformations were previously used. Consequently, when searching for an appropriate predictive model, finding the sequence of transformations required to stationarize a time series often provides important hints. An important part of the process of fitting an ARIMA model is the stationarisation of a time series by differentiating (where required). Another reason to try to stationarize a time series is to get meaningful sample statistics like means, variances, and correlations with other variables. Such statistics as future conduct descriptors are only useful if the series is stationary. For example, if the series

increases continuously over time, the mean and variance of the sample will increase with the size of the sample, and they will also underestimate the mean and variance of future periods. And if a series' mean and variance are not well-defined then its correlations with other variables are not either. You should, therefore, be careful in trying to extrapolate regression models that are fitted to non-stationary data. Most market and economic time series are far from stationary when represented in their original measuring units and will also usually display patterns, cycles, random walking, and other non-stationary behavior, even after deflation or seasonal adjustment. If the series has a stable long-run pattern and appears to return to the trend line after a disturbance, it can be stationarized by de-trending (e.g. by fitting a trend line and subtracting it out before fitting a formula, or by using the time index as an independent variable in a regression or ARIMA model), either in combination with logging or deflation. It is said a series like this is trend stationary. Nevertheless, even de-trending is sometimes not enough to make the series stationary, in which case it may be necessary to transform it into a series of differences from period to period and/or from season to season. If the mean, variation, and self-correlations of the original series are not constant in time, maybe the statistics of the series shifts between times or between seasons will be constant even after detrending. This is called difference stationary. (Sometimes it can be hard to tell the difference between a trend-stationary and a differential-stationary series, and a so-called unit root test can be used to get a more definitive answer.)

For understanding if our data is stationary, I have performed a seasonal decomposition of the data. This decomposition transforms our time series model into four different time series. These four components series are:

1. Observed → Actual price movements.
2. Trend → The series increases or decreases in value.
3. Seasonality → Patterns which repeat with a fixed time period.
4. Residual → The part of the original time series left after removing seasonal and trend series.

All these series components I have summarized in the graph below. The trend there is not constant that means that data is non-stationary.

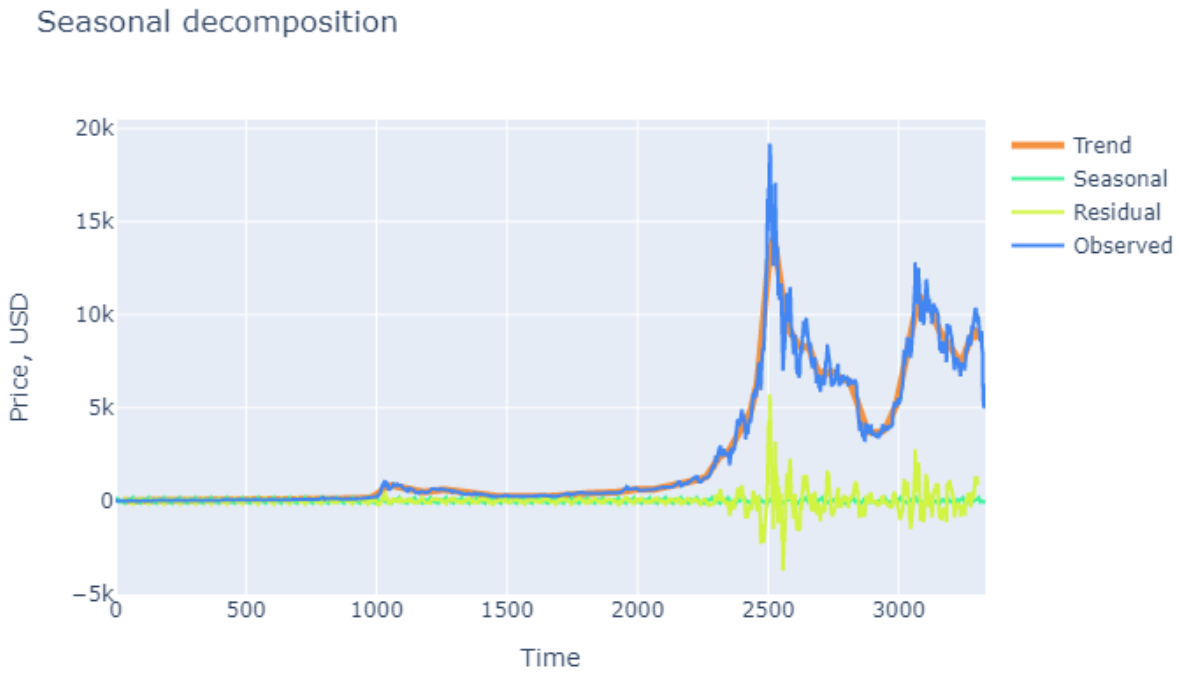


Figure 13. Seasonal decomposition

2.2.3 Autocorrelation and Partial Autocorrelation

Two things that I will examine for explain our data are autocorrelation and partial correlation.

1. Autocorrelation

Autocorrelation is a data feature that indicates the degree of similitude over successive time intervals between the values of the same variables.
If you have a number series and there is a pattern such that values can be predicted based on

preceding values in the series, the number series is said to exhibit autocorrelation. Often called serial correlation and serial dependency. The presence of autocorrelation in a model's residuals is a sign the model may be unsound. A correlogram (ACF plot) is used to diagnose autocorrelation and can be tested using the Durbin-Watson test. The auto part of autocorrelation is self-related from the Greek word, and autocorrelation means data that is correlated to itself, rather than being correlated with some other data.

1.1 Positive and negative autocorrelation

The above example shows positive autocorrelation of the first order, where the first order shows that observations that are one apart are correlated, and positive means that the correlation between the observations is positive When data showing a positive first order correlation is plotted, as on the left, the points appear in a smooth snake-like curve. If connected, the points form a zigzag pattern we are dealing with negative first-order correlation, as shown on the right.

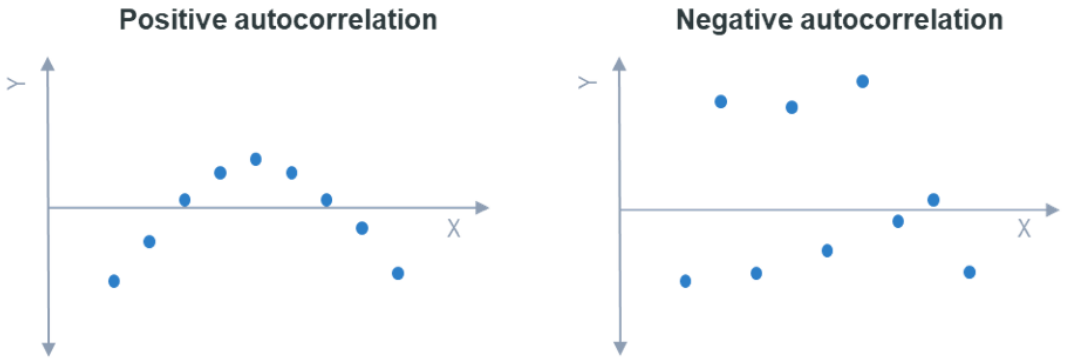


Figure 14. Positive/Negative correlation

1.2 The implications of autocorrelation

When autocorrelation is observed from a model in the residuals it indicates that the model is incorrectly defined (i.e., wrong in some sense). A cause is that the model has missing

some key variables or variables Where data is collected over time or space, and this is not specifically taken into account by the model, autocorrelation is probable. For example, if a weather model is wrong in one suburb, it will probably be wrong in a neighboring suburb the same way. The fix is either to include the missing variables or to model the autocorrelation explicitly (e.g., using an ARIMA model). The existence of autocorrelation means that there are misleading computed standard errors, and consequently p-values.

For examining the autocorrelation of my data, I have built the correlogram.

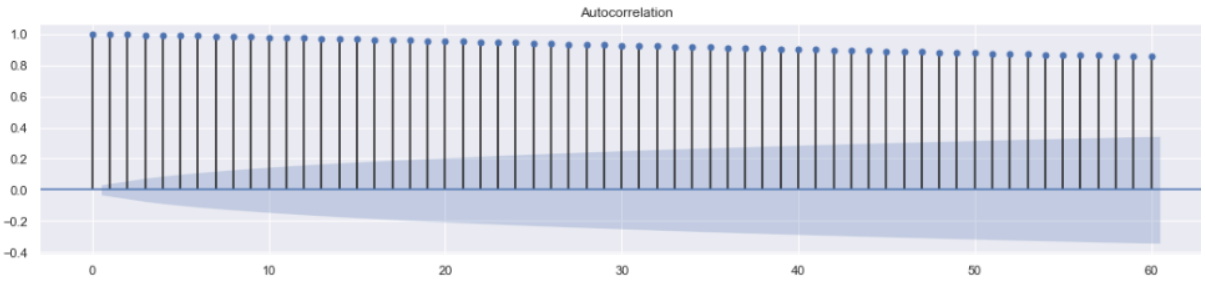


Figure 15. Autocorrelation

The plot begins at lag 1 with a relatively high autocorrelation (1), which slowly decreases. The decreasing autocorrelation is typically linear. This means that our data has a moderate positive autocorrelation which provides moderate predictability. The autocorrelation plot indicates a non-stationary process and suggests an ARIMA model.

2. Partial Autocorrelation

Partial autocorrelations are useful for identifying an autoregressive model order. Before doing the partial autocorrelation, we must do autocorrelation. If this autocorrelation concludes that the appropriate model is an AR model, we must do then the partial autocorrelation which helps to examine the order of this model. The partial autocorrelation of an AR(p)process is zero at lag p+1 and higher.

In partial correlation, we search for the point on the plot where the partial autocorrelation coefficients are effectively zero. To this end, it is helpful to position a 95 percent confidence interval for statistical significance.

For examining the autocorrelation of my data, I have built the partial autocorrelation plot. Where:

Vertical axis → Coefficient of partial autocorrelation at lag h.

Horizontal axis → Time lag h.

If we see carefully on the plot, we can see a blue border and this blue border is our 95 % confidence interval.

Our differentiated data partial autocorrelation plot with 95 per cent confidence bands shows that only the first and second lag partial autocorrelations are significant. This suggests a Model AR (2).

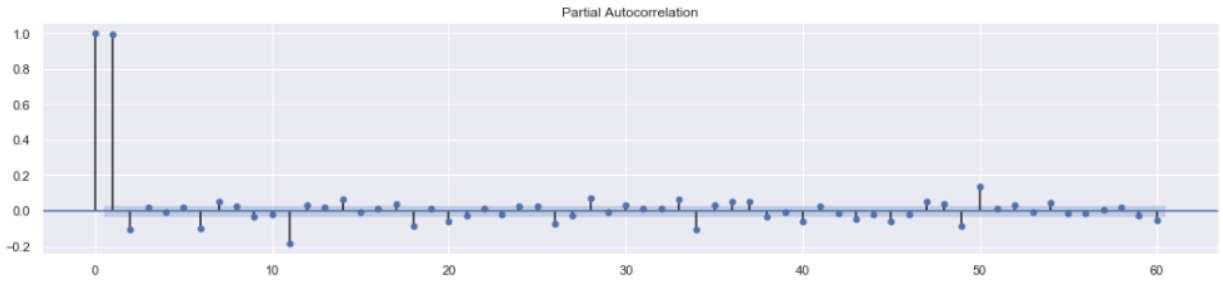


Figure 16. Partial Autocorrelation

2.3 Data preprocessing

In Machine Learning, data preprocessing is a critical step that helps to enhance data q uality to facilitate the extraction of useful insights from the data. This technique makes the raw data suitable for Machine Learning models for construction and training. Data

preprocessing in Machine Learning is, in simple words, a technique of data mining that transforms raw data into an understandable and readable format. There are some steps for doing the preprocessing of the data.

2.3.1 Relevant dataset

First, we need a relevant dataset. As we showed in the first section of this chapter, we own it.

2.3.2 Missing values

Secondly, we must identify the missing values. This is a step that I did not make in my dataset because it has not any null values.

2.3.3 Encoding the categorical data

The next step is to encode the categorical data. Categorical data means that the data has a textual nature, but Machine Learning is a mathematical model and needs numbers to work. For this reason, is needed the encoding of this textual data. If we see our dataset in the first section of this chapter, we will realize that our dataset has any categorical data. This means that I do not need to do this step.

2.3.4 Splitting the dataset into train and test sets

In the Machine Learning model, each dataset must split into two separate sets: train and test sets. A dataset subset is defined from the training set for the machine learning model being trained. We are already conscious of the output here. On the other hand, a test set is the subset of the dataset used to test the model for machine learning. This test set is used by

Machine Learning to predict outcomes. The dataset is generally split into ratio 70:30 or 80:20. This means 70% or 80% goes for training the model while 20% or 30% goes for testing. I have done this step of preprocessing to my dataset. Set of training runs from 06 February 2011 to 30 September 2019. It contains 3183 days. While the set of test runs from 25 October 2019 to 22 March 2020 and it contains 150 days. I have defined a function that has created X inputs and Y labels. Where X labels are values from a future point of time and Y values from the past. In our function, we have put the look_back parameter which tells us the amount of these values. In my case, I have set the look_back parameter to 30. This means that I have predicted the value based on the previous 30 days values. After the look_back function, I have reshaped the train and test sets to suit the model requirements. Due to the LSTM models are scale sensitive I have used the MinMaxScaler to scale the dataset. I have made the dataset suitable for time series forecasting and also for Keras. In the end, I have checked the normalization and the shapes.

2.3.5 Feature selection

Feature selection is one of the most important aspects of data mining. It is essentially concerned with extracting useful data features to make it easier for machine learning models to implement their predictions. To check the features' behavior concerning Bitcoin prices, we plot the data for all 20 features over the entire period of time as shown in Figure 14 below. According to this, I concluded that the most correlated features are Google trends, interest rates, and Ripple price.

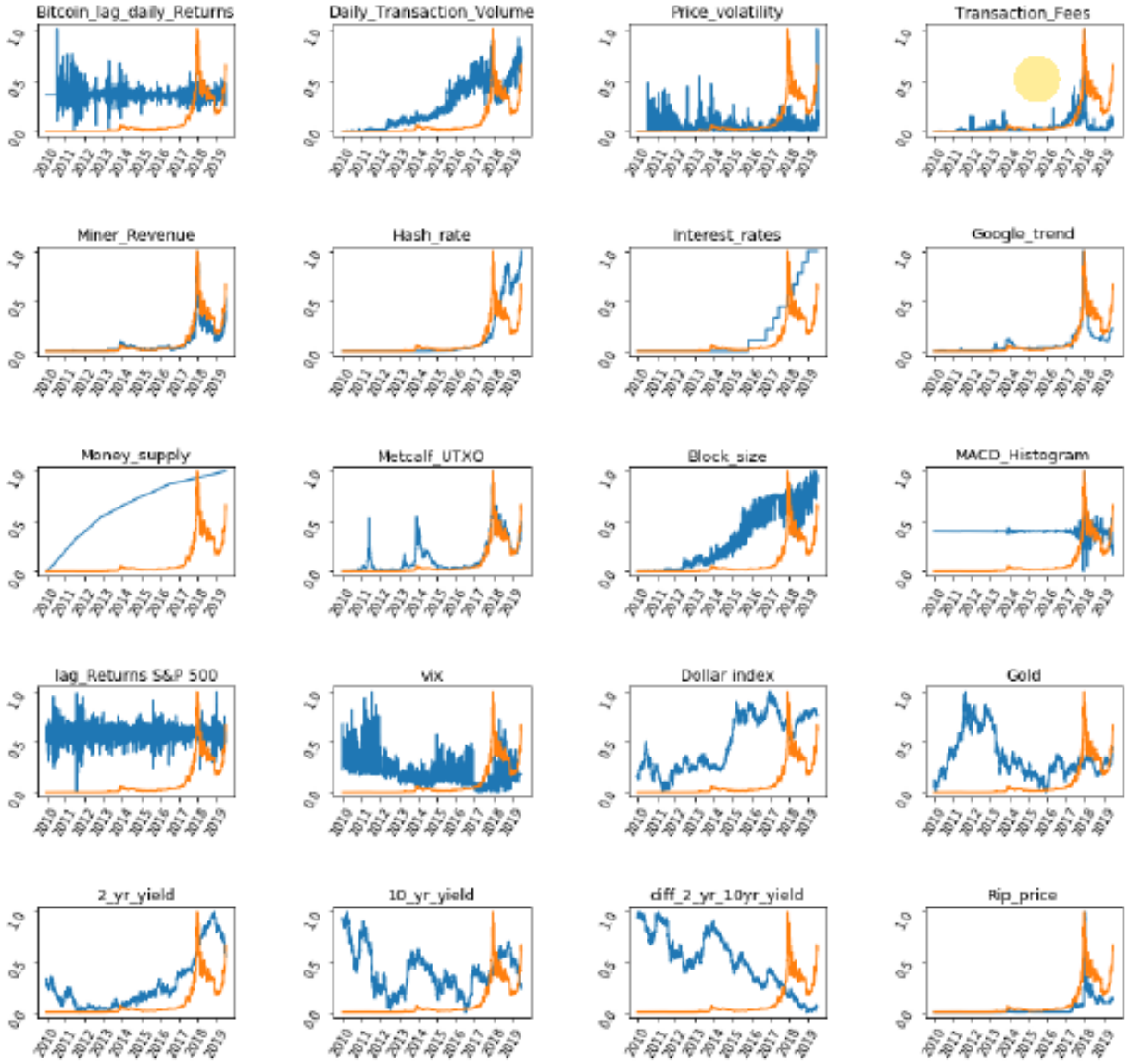


Figure 17. Feature Selection

CHAPTER 3

METHODOLOGY

This chapter will give an overview of the whole work done for this thesis. The first section explains the multilayer perceptron, MLP of Scikit-Learn implementation, and MLP from Keras implementation. The second section explains the RNN model, the LSTM model, the architecture of LSTM, and its implementation. The third part explains GRU with recurrent dropout Neural Network and its implementation. In the fourth section, we are going to explain the GRU with two layers model and also its implementation. The fifth part explains the CNN model, the architecture and, also the implementation of this model. Later on, in this chapter, the time series, the ARIMA model and its implementation are going to be explained.

3.1 What is a multilayer perceptron?

The multilayer perceptron (MLPs) is used for classifying datasets that are not linearly separable. This is achieved through the use of a stable and complex architecture to learn models of regression and classification for difficult datasets.

3.1.1 How does a multilayer perceptron works?

The Perceptron is made of an input layer and an output layer which are completely connected. MLPs have the same input and output layers but may have numerous hidden layers in between the previously layers, as we can see at the figure below.

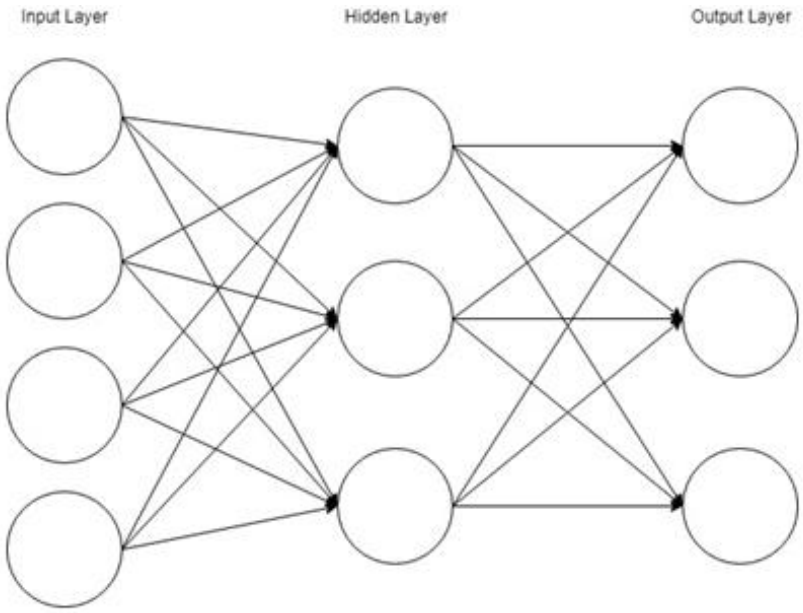


Figure 18. MLPs architecture

The algorithm for the MLP is shown below:

1. The inputs are pushed forward through the MLP by taking the real number of the input with the weights that exist between the input layer and also the hidden layer (W-H). This real number yields a price at the hidden layer. We are not pushing forward that value as we would with a perceptron though.
2. MLPs use activation functions at each of the layers they calculate. There are several activation functions to discuss: tanh, sigmoid function, rectified linear units (ReLU). Move the measured output into each of those activation functions at the actual layer.
3. When the calculated output has been pushed through the activation function at the hidden layer, move it to the next layer inside the MLP by taking the real number with the corresponding weights.

4. Repeat steps two and three until the output layer is reached.
5. At the output layer, the calculations will either be used for a backpropagation

algorithm that corresponds to the activation function that was selected for the MLP (in the case of training) or a call is going to be made supported the output (in the case of testing).

In this study, I will implement two types of MLP. First, I will use the MLP Regression of Scikit-Learn and then the MLP from Keras.

3.1.2 MLP Regression of Scikit-Learn [13]

Some important variables in Machine Learning are hyperparameters. They are very important for efficiency. So, for determining them I will use a grid search. But these parameters must be optimal and for this reason, I will use cross-validation. Cross-validation is a robust method which helps us in finding optimal hyperparameters.

Here are some parameters I have used in this method:

1. In grid search, I have specified the hidden layer number (1), the learning rate initial (impacts the outcome of the training), the hidden layer size, the momentum, and the alpha.
2. a large number of epochs (500) and activated the early stop.
3. a batch size equal to 125. Since we are working with time series, no need for shuffling.
4. the tanH as an activation function because before converging to zero it can sustain for a long time.
5. a default adaptive learning rate. The adaptive learning rate keeps the learning rate constant to the learning rate initial as long as training loss keeps decreasing. Anytime two consecutive epochs fail to decrease training loss by a minimum of tol or fail to increase validation score by a minimum of tol and if 'early_stopping' is true, the present learning rate is divided by 5.
6. a momentum term to avoid staying stuck at a local minimum. This momentum term is a value (between 0 and 1) and is used for increasing the size of the steps that are taken towards the minimum by trying to jump from local minima. We

can use momentum for smoothing out the variations if the gradient is changing direction.

7. the Adam optimizer solver to avoid overfitting.

1. Preprocessing

I have prepared our dataset according to the requirements of the model. For doing that I have split it into train and test parts. For creating X inputs and Y label for our model I have defined a function. Based on some previous and current values I have predicted the future value. So, X inputs are values from the past while Y label is the value from the future. By using the parameter look_back in our function I have set the amount of these values. The shape of the training set from the past values is 3153 observations of 210 variables, and the shape of the test set from the past values is 120 observations of 210 variables. Meanwhile, the shape of the training set from the future values is 3153 observations of 1 variable, and the shape of the test set from the future values is 120 observations of 1 variable.

3.1.3 MLP for Keras [14]

The second model of MLP that I have used in my study is MLP for Keras.

Parameters that I have used in my code for this model:

1. a batch size equal to 125.
2. I have iterated 500 times maximum over the entire training set.
3. a dropout with probability 0.1.
4. The fully connected layer will have 1 neuron
5. a 0.1 for factoring weights penalty.

[15] **The steps I have used for implementing the model:**

1. Creating the Training and Test dataset

The shape of the training and test set is the same. (120 observations of 1 variable).

2. Building the Deep Learning Regression Model

As I specified above, I have built a regression model with the use of Keras. First, I have defined the model. I have used a Sequential model because our network is made of a linear stack of layers. There I have added 2 stacked dense layers. Secondly, I have used tanH as an activation function. In the next step, I have defined an optimizer and the loss measure for training. Our loss measure is the mean squared error and for the optimizer, I have used Adam because his main advantage is that we do not need to specify the learning rate.

After that, I have fit the model on the training dataset. I have taken a large number of epochs (500) and activate the early stop. As is shown in the figure underneath.

```
Epoch 164/500
2995/2995 [==============================] - 0s 32us/sample - loss: 0.0198 - val_loss: 0.0319
Epoch 165/500
2995/2995 [==============================] - 0s 34us/sample - loss: 0.0191 - val_loss: 0.0308
Epoch 166/500
2995/2995 [==============================] - 0s 28us/sample - loss: 0.0199 - val_loss: 0.0668
Epoch 00166: early stopping
```

Figure 19. Training iterations of MLP Keras

3. Predict and Compute Evaluation Metrics

After that I got prediction and then I have made some transformation to be able to calculate MSE (mean squared error) properly in USD.

Before I start with the implementation of the LSTM model, I have given a brief description of RNN and LSTM.

3.2 Recurrent Neural Networks (RNN)

If we want to learn information about the immediately previous step, we can use RNN. [16]

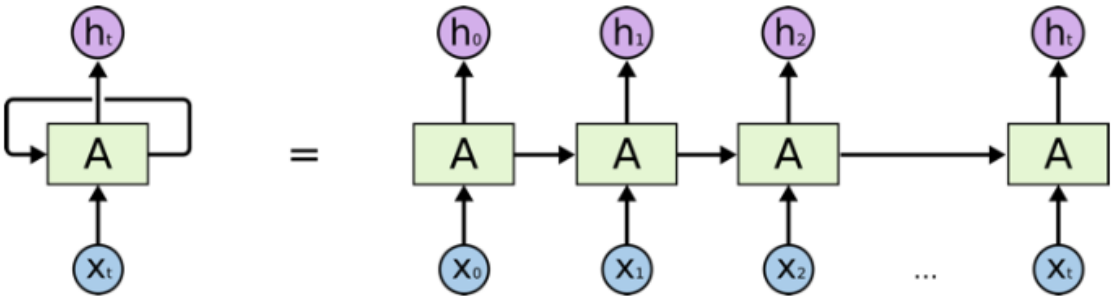


Figure 20. RNN architecture

At the figure above it is shown a typical RNN where:

X(t) → input

h(t) → output

A → neural network (gets information from the previous step in a loop)

But sometimes, in some cases, RNN does not work practically. Why does that happen? Below is the reason:

The information during the training of RNN goes into a loop again and again. This causes very large updates to the weights of the neural network model. But during an update occurs the accumulation of error gradient and all those large updates result in an unstable network. The worst thing that can happen here is the overflow of weight values and make them very large or vanish of them and make them NaN values.

3.2.1 Long Short-Term Memory

The above problem of RNN brought the need for developing a new version of RNN model that is called Long Short-Term Memory. [17] To control the memorizing process this model uses gates. Below I have talked a little bit about the architecture of LSTM.

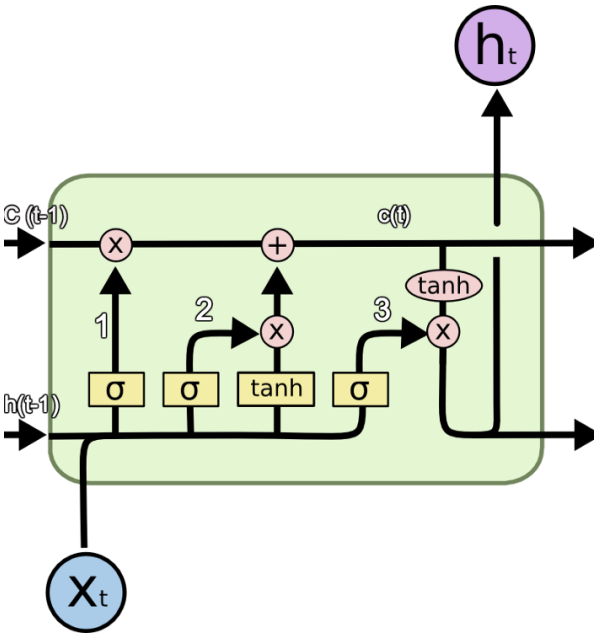


Figure 21. LSTM architecture

At the figure above it is shown a typical LSTM where:

a) X → Scaling of information
b) + → Adding information
c) σ → Sigmoid layer
d) tanh → tanh layer
e) h(t-1) → Output of last LSTM unit
f) c(t-1) → Memory from last LSTM unit
g) X(t) → Current input
h) c(t) →New updated memory
i) h(t) → Current output

- tanh is a function that overcomes the vanishing gradient problem.
- Sigmoid is used to forget or remember the information. (output 0 or 1)

In LSTM the information passes through many units. Below I have shown the connection between three main components.

1. The input X(t) and h(t-1) are taken by the sigmoid layer which decides what to remove from old output (by outputting a 0).

2. When new information comes, a sigmoid layer should decide which of them should be updated or ignored. A new vector that contains all the values from the new input is created from a tanh layer. The sigmoid layer and the tanh one are multiplied for updating the new cell state. To give c(t) the new cell state is added to old memory c(t-1).

3. In the final step, we are going to decide which part of the cell state we are going to output. This thing will be done by a sigmoid layer. For generating all possible values, we use a tanh to put the cell state. After that, we multiply those values with the output of the sigmoid gate. In this way, we only output the parts we want to.

3.2.2 The implementation of LSTM

Parameters that I have used in my code for this model:

1. a batch size equal to 125.
2. I have iterated 500 times maximum over the entire training set.
3. a dropout with probability 0.1.
4. The fully connected layer will have 1 neuron
5. a 0.0001 for factoring weights penalty.

The steps I have used for implementing the model:

1. Creating the Training and Test dataset

The shape of the training and test set is the same. (120 observations of 1 variable).

2. Building the LSTM Model

First, I have defined the model. I have used a Sequential model because our network is made of a linear stack of layers. There I have added one LSTM layer and one dense layer. Secondly, I have used tanH as an activation function. In the next step, I have defined an optimizer and the loss measure for training. Our loss measure is the mean squared error and for the optimizer, I have used Adam because his main advantage is that we do not need to specify the learning rate.

After that, I have fit the model on the training dataset. I have taken a large number of epochs (500) and activate the early stop. As is shown in the figure underneath.

```
2995/2995 [==============================] - 2s 735us/sample - loss: 0.0110 - val_loss: 0.0160
Epoch 111/500
2995/2995 [==============================] - 2s 755us/sample - loss: 0.0086 - val_loss: 0.0163
Epoch 112/500
2995/2995 [==============================] - 2s 756us/sample - loss: 0.0105 - val_loss: 0.0155
Epoch 00112: early stopping
```

Figure 22. Training iterations of LSTM

3. Predict and Compute Evaluation Metrics

After that I got prediction and then I have made some transformation to be able to calculate MSE (mean squared error) properly in USD.

3.3 Gated recurrent units

The idea and the equations behind a GRU layer are similar to that of an LSTM layer.

$$z = \sigma(x_t U^z + s_{t-1} W^z)$$
$$r = \sigma(x_t U^r + s_{t-1} W^r)$$
$$h = tanh(x_t U^h + (s_{t-1} \circ r) W^h)$$
$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

Figure 23. **Equations behind a GRU layer**

A reset gate r and an update gate z are two gates of GRU. The combination of the new input with the previous memory is made by the reset gate. For defining the space of previous memory for keeping around is used the update the gate. We can arrive in an RNN model if we set the reset to all 1's and update the gate to all 0's. In this model, the idea of learning long-term dependencies is the same as in an LSTM but with some changes.

- A GRU is made of two gates, an LSTM is made of three gates.
- The internal memory (c_t) does not exist in the GRU model.
- The reset gate r is applied to the previous hidden state, the input and forget gates are coupled by an update gate z. Meanwhile, in an LSTM, the responsibility of the reset gate is to split up into both r and z.
- In GRU, when computing the output, we don't apply a second nonlinearity.

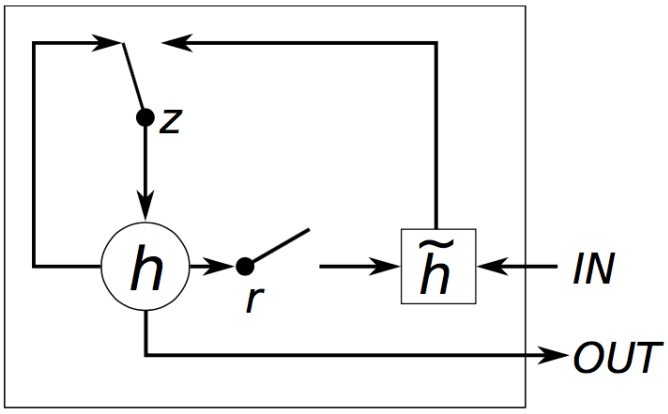


Figure 24. GRU architecture

3.3.1 GRU with recurrent dropout Neural Network

Parameters that I have used in my code for this model:

1. In each iteration, I have considered 125 training examples at once.
2. I have iterated 20 times over the entire training set.
3. a dropout with probability 0.01.
4. The fully connected layer will have 1 neuron
5. a 0.0001 for factoring weights penalty.

And now I will start with the steps I have used for implementing the model:

1. Creating the Training and Test dataset

The shape of the training and test set is the same. (120 observations of 1 variable).

2. Building the GRU Model

First, I have defined the model. I have used a Sequential model because our network is made of a linear stack of layers. There I have added one GRU layer and one dense layer. Secondly, I have used tanH as an activation function. In the next step, I have defined an optimizer and the loss measure for training. Our loss measure is the mean squared error and for the optimizer, I have used Adam because his main advantage is that we do not need to specify the learning rate.

After that, I have fit the model on the training dataset. I have taken a large number of epochs (500) and activate the early stop. As is shown in the figure underneath.

```
Epoch 138/500
2995/2995 [==============================] - 3s 1ms/sample - loss: 0.0159 - val_loss: 0.0107
Epoch 139/500
2995/2995 [==============================] - 3s 1000us/sample - loss: 0.0058 - val_loss: 0.0116
Epoch 140/500
2995/2995 [==============================] - 4s 1ms/sample - loss: 0.0122 - val_loss: 0.0111
Epoch 00140: early stopping
```

Figure 25. Training iterations of GRU

3. Predict and Compute Evaluation Metrics

After that I got prediction and then I have made some transformation to be able to calculate MSE (mean squared error) properly in USD

3.3.2 GRU with 2 layers

For capturing a higher-level interaction, we can add in our GRU model a second layer.

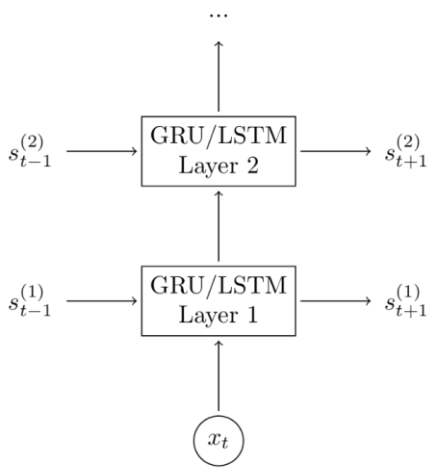


Figure 26. GRU with two-layers architecture

The implementation of GRU with two-layers

Parameters that I have used in my code for this model:

1. In each iteration, I have considered 125 training examples at once.
2. I have iterated 20 times over the entire training set.
3. a dropout with probability 0.01.
4. The fully connected layer will have 1 neuron
5. a 0.0001 for factoring weights penalty.
6. Two GRU layers.

And now I will start with the steps I have used for implementing the model:

1. Creating the Training and Test dataset

The shape of the training and test set is the same. (120 observations of 1 variable).

2. Building the GRU with two-layers Model

First, I have defined the model. I have used a Sequential model because our network is made of a linear stack of layers. There I have added two GRU layer and one dense layer. Secondly, I have used tanH as an activation function. In the next step, I have defined an optimizer and the loss measure for training. Our loss measure is the mean squared error and for the optimizer, I have used Adam because his main advantage is that we do not need to specify the learning rate.

After that, I have fit the model on the training dataset. I have taken a large number of epochs (500) and activate the early stop. As is shown in the figure underneath.

```
Epoch 27/500
2995/2995 [==============================] - 5s 2ms/sample - loss: 0.0120 - val_loss: 0.0116
Epoch 28/500
2995/2995 [==============================] - 5s 2ms/sample - loss: 0.0131 - val_loss: 0.0149
Epoch 29/500
2995/2995 [==============================] - 5s 2ms/sample - loss: 0.0105 - val_loss: 0.0129
Epoch 00029: early stopping
```

Figure 27. Training iterations of GRU with two-layers

3. Predict and Compute Evaluation Metrics

After that I got prediction and then I have made some transformation to be able to calculate MSE (mean squared error) properly in USD.

3.4 Convolutional Neural Network (CNN)

This algorithm can classify and recognize features in images for computer vision. It is designed to perform tasks and analyze visual inputs such as object detection, classification, and segmentation.

CNN is made of two parts:

- For splitting the various features of the image this model uses a convolution tool.
- For predicting the best description, a fully connected layer uses the output of the convolution layer.

3.4.1 CNN Architecture

A CNN is made of several kinds of layers:

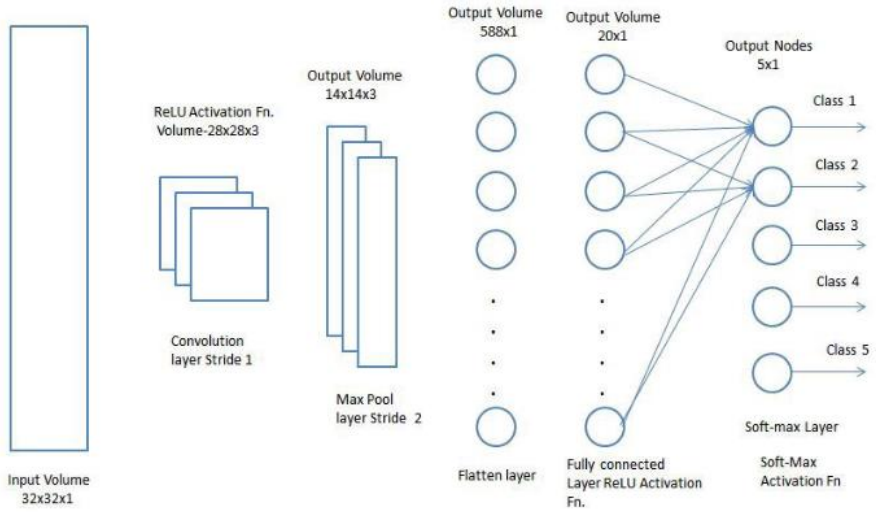


Figure 28. CNN architecture

- Convolutional layer → For predicting the class probabilities for each feature this model creates a feature map. This is made by applying a filter that scans the whole image.

- Pooling layer (downsampling) → The information that is generated for each feature from the convolutional layer, here is scaled down and is maintained only the most essential information. (The process in those two first layers usually repeat several times).

- Fully connected input layer → The output that is generated from the previous layer here is “flatten” and is turned into a single vector that can be used later as an input for the next layer.

- Fully connected layer → For predicting an accurate label this layer applies weights over the input. This input is the one that is generated by the feature analysis.

- Fully connected output layer → For determining a class for the image this layer generates the final probabilities.

3.4.2 CNN implementation [18]

Parameters that I have used in my code for this model:

1. In each iteration, I have considered 32 training examples at once.
2. I have iterated 500 times maximum over the entire training set.
3. a dropout after pooling with probability 0.01.
4. The fully connected layer will have 1 neuron
5. a 0.0001 for factoring weights penalty.
6. a 10x10 kernels throughout.
7. I will initially have 16 kernels in first convolutional layer.
8. I will initially have 32 kernels in second convolutional layer.

And now I will start with the steps I have used for implementing the model:

1. Creating the Training and Test dataset

The shape of the training and test set is the same. (120 observations of 1 variable).

2. Building the CNN Model

First, I have defined the model. I have used a Sequential model because our network is made of a linear stack of layers. There I have added two convolutional layers, two Maxpooling layers, two dropout layers, one flatten layer and one dense layer. Secondly, I have used tanH as an activation function. In the next step, I have defined an optimizer and the loss measure for training. Our loss measure is the mean squared error and for the optimizer, I have used Adam because his main advantage is that we do not need to specify the learning rate.

After that, I have fit the model on the training dataset. I have taken a large number of epochs (500) and activate the early stop. As is shown in the figure underneath.

```
Epoch 98/500
2995/2995 [==============================] - 1s 245us/sample - loss: 0.0043 - val_loss: 0.0156
Epoch 99/500
2995/2995 [==============================] - 1s 246us/sample - loss: 0.0054 - val_loss: 0.0194
Epoch 00099: early stopping
```

Figure 29. Training iterations of CNN

3. Predict and Compute Evaluation Metrics

After that I got prediction and then I have made some transformation to be able to calculate MSE (mean squared error) properly in USD.

3.5 Time series

A time-series [19] is a successive sequence of numerical data points. In investing, a time series monitors the movement of the selected data points, such as the price of a safe, over a given period of time with reported data points at regular intervals. Any variable that varies over time can be taken into a time series. In investing, a time series is widely used to measure a security's price over time. In the short term, this can be tracked such as the price of a security on the hour over a business day, or the long term, such as the price of security closing on the last day of each month over a five-year period.

3.5.1 Time series Analysis

Analysis of the time series may be useful for seeing how a given asset, security, or economic variable changes over time. It may also be used to examine how over the same time period the changes associated with the chosen data point compare with shifts in other variables.

3.5.2 Time Series Forecasting

Predicting time series uses historical values and related trend knowledge to forecast future behavior. This most often concerns trend analysis, cyclical fluctuation analysis, and seasonality issues. As for all forms of forecasting, there is no guarantee of success.

3.5.3 Autoregressive Integrated Moving Average Model (ARIMA)

For analyzing and forecasting time series data we can use a class of statistical models called the ARIMA model. [20]

The name ARIMA has an explanation that I am giving below:

- **AR** = *Autoregression*. This model uses a relation between the observation and the number of lagged observations.
- **I** = *Integrated*. To make the time series stationary this model uses the difference of raw observations.
- **MA** = *Moving Average*. This model makes use of the dependency between a residual error from a moving average model and an observation for lagging observation. In the model these components are specified as parameters where:
- **AR = p**. The lag orders. (the number of lag observations).
- **I = d**. The degree of differencing. (The number of times the raw Observations differ).
- **MA = q**. The order of the moving average. (the size of the average movable window).

And now I will start with the steps I have used for implementing the model:

1. Building the ARIMA Model

Firstly, I have fitted an ARIMA (2,1,1) model. Where:

- the lag value (p) = 2.
- the difference order = 1. (This made the time series stationary).
- the average model = 1.

To fit the ARIMA model I have used the statsmodels library. This library created the ARIMA model as follows:

1. Called ARIMA () and showed up the p, d, and q parameters.
2. Called fit () function.
3. Called predict () function. This function has made predictions and also it has gotten the index of the time steps and made predictions as arguments.

ARIMA Model Results

Dep. Variable:	D.Bitcoin Core (BTC) Price	No. Observations:	3332
Model:	ARIMA(2, 1, 1)	Log Likelihood	-22368.206
Method:	css-mle	S.D. of innovations	199.179
Date:	Mon, 01 Jun 2020	AIC	44746.412
Time:	17:09:00	BIC	44776.969
Sample:	1	HQIC	44757.345

	coef	std err	z	P>|z|	[0.025	0.975]
const	1.8549	3.735	0.497	0.619	-5.466	9.176
ar.L1.D.Bitcoin Core (BTC) Price	-0.6685	0.081	-8.292	0.000	-0.827	-0.511
ar.L2.D.Bitcoin Core (BTC) Price	0.0357	0.021	1.674	0.094	-0.006	0.077
ma.L1.D.Bitcoin Core (BTC) Price	0.7675	0.079	9.777	0.000	0.614	0.921

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-1.3924	+0.0000j	1.3924	0.5000
AR.2	20.1342	+0.0000j	20.1342	0.0000
MA.1	-1.3029	+0.0000j	1.3029	0.5000

Figure 30. Arima Model Results

2. Rolling forecast ARIMA model

- I had also used the forecast () function which with the use of the model performs a one-step forecast.
- I divided the training dataset into train and testing sets, where the train set is used to fit the model and the test set is used for generating a prediction for each of his elements.

- In view of the dependence of observations in earlier differentiation phases, the AR model has required a rolling forecast. I have done that by re-creating the ARIMA model is provided after every new observation.
- History is a list that I manually kept track of all observations. This list is seeded with the training data.

3. Predict and Compute Evaluation Metrics

After that I got prediction and then I have made some transformation to be able to calculate MSE (mean squared error) properly in USD.

CHAPTER 4

SETUP AND ANALYSIS OF THE EXPERIMENTAL STUDY

4.1 Architectures

As I have mentioned earlier, in this thesis are used different models to predict bitcoin price, for this reason, I will explain the architecture for each of them.

4.1.1 MLP Regression of Scikit-Learn and MLP for Keras

Since the two first models are multilayer perceptron; I have used the same architecture in both of them. The architecture is made of one input layer, two hidden layers with 25 neurons each, and one output layer.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 25)                5275
_________________________________________________________________
dropout_1 (Dropout)          (None, 25)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 26
=================================================================
Total params: 5,301
Trainable params: 5,301
Non-trainable params: 0
_________________________________________________________________
```

Figure 31. Summary of MLP from Keras

4.1.2 Long Short-Term Memory

For the LSTM model, I have used an LSTM layer with 50 neurons and a dense layer.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 50)                11600
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 11,651
Trainable params: 11,651
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 32. Summary of LSTM model

4.1.3 GRU with recurrent dropout Neural Network

In the first GRU model, I have used a GRU layer with 50 neurons and a dense layer.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 50)                8850
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 51
=================================================================
Total params: 8,901
Trainable params: 8,901
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 33. Summary of GRU model

4.1.4 GRU with 2 layers

For the GRU with two layers model, I have used in the first GRU layer 50 neurons, in the second GRU 10 neurons and a dense layer.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_1 (GRU)                  (None, 30, 50)            8850
_________________________________________________________________
gru_2 (GRU)                  (None, 10)                1860
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 10,721
Trainable params: 10,721
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 34. Summary of GRU with two-layers model

4.1.5 Convolutional Neural Network (CNN)

The CNN model is a little bit more complicated. Here I have used two convolutional layers with 112 neurons and 512 neurons, respectively. Two max-pooling layers with 112 neurons and 512 neurons, respectively. Two dropout layers with 112 neurons and 512 neurons, respectively. One flatten layer with 512 neurons and one dense layer.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 16, 7)             4816
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 16, 7)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 7)             0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 16, 32)            2272
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 16, 32)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 32)            0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 513
=================================================================
Total params: 7,601
Trainable params: 7,601
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 35. Summary of CNN model

4.2 Running the experiments

For predicting the bitcoin price, I have used TensorFlow. The time that I have spent running all the models has been 1 month. During this month, I have fixed a lot of errors. Most of those errors were because of the libraries. Sometimes I had forgotten to declare them, and some other times, some libraries had changed the syntax of declaration.

Another problem was the part of downloading all the libraries that I have used in my code.

A difficult part of this work was to find a suitable dataset. To found it, I have emailed a lot of people that work in Bitcoin web pages.

About the computational time, the MLP models was faster than other models.

CHAPTER 5

RESULTS

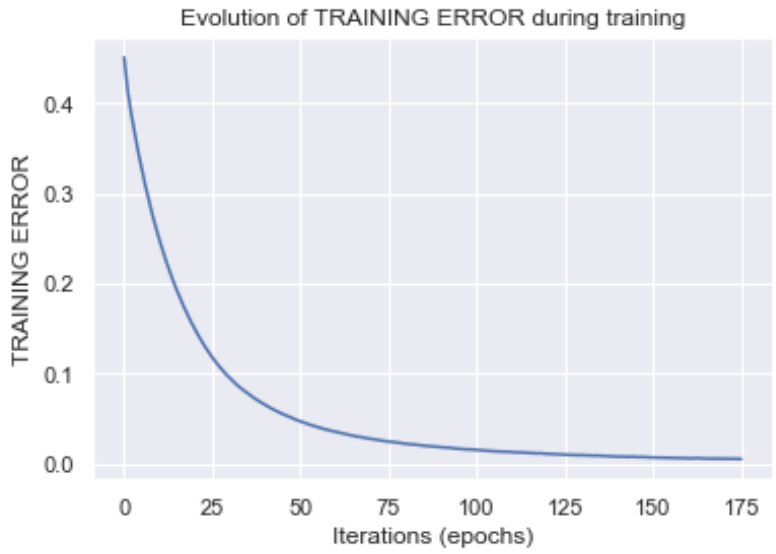
This chapter will give the results for all the models I have used in chapter 3. The first section gives the results of train and test loss for all the models and a comparison between them. The second section gives the result for the timely implementation of all models, the mean squared error and, also gives a comparison between all the results for different models. The third part gives a comparison of the actual price and the price predicted from our models.

5.1 Train and test loss

At all models that I have used in this thesis, during the iteration I have done an early stopping. This is shown in figures 20, 24, 28, 31, and 34. But what is early stopping and why is it done? If we see at these figures I just mentioned, the training error decreases over time while validation error increases. To do the regularization in deep learning I have used that early stopping strategy. In this strategy, the best parameter setting with the lowest validation error is reserved, and the model associated with the parameter setting with the minimum validation error is selected (not the latest one) when training is completed (with the lowest training error). After doing the early stop I have plotted the result of train and test loss for each model. (figure 36)

Let us examine the plots. If training loss is greater than validation loss the data is underfitting. If training loss is smaller than validation loss the data is overfitting and if both losses are roughly the same the data is perfect fitting. MLP from Keras model seems to have a perfect fitting to data. LSTM at the 17 first epochs has overfitting to data but later the validation loss is decreased and has made a fitting to data. The GRU with recurrent dropout

neural network seems to have a fitting to data and when we add another layer the difference between two losses has increased. And the CNN model has overfitting of the data.



a)

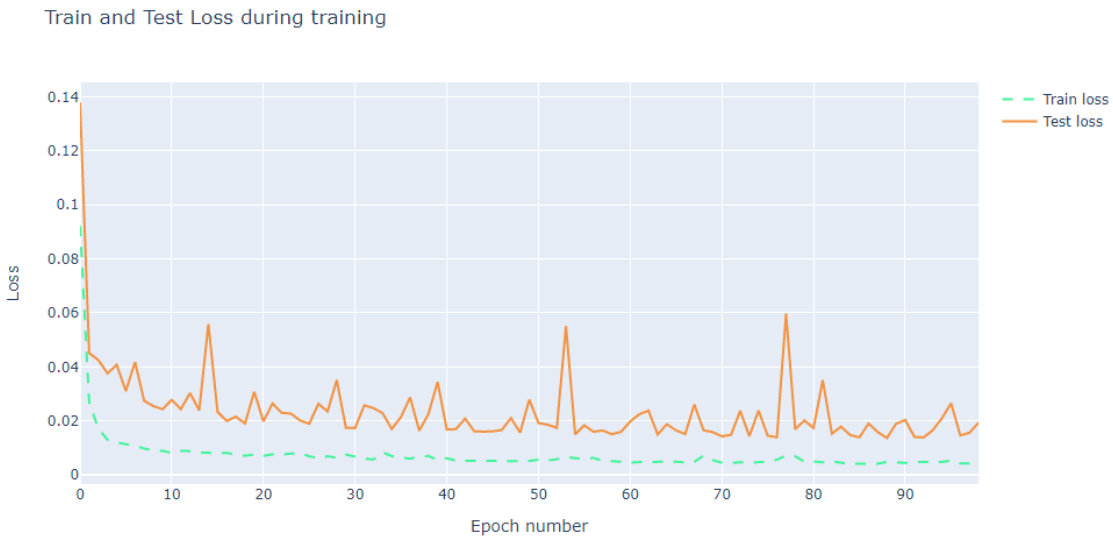
b)


Train and Test Loss during training
Train loss
Test loss
Loss
Epoch number
0
0.2
0.4
0.6
0.8
1
1.2
0
20
40
60
80
100

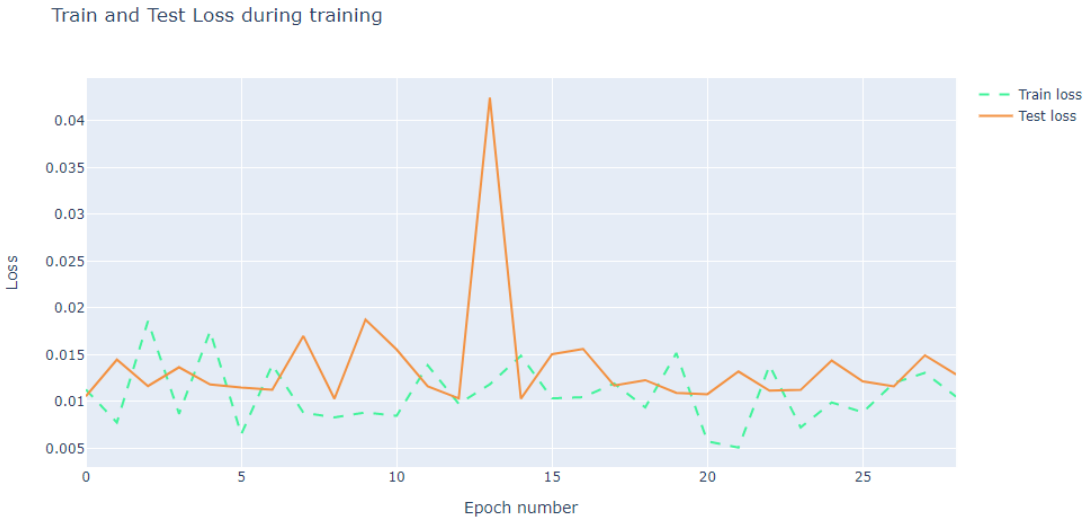
c)


Train and Test Loss during training
Train loss
Test loss
Loss
Epoch number
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0
20
40
60
80
100
120

d)



e)



f)

Figure 36. Train and Test Loss during training of a) MLP regression of Scikit-Learn b) MLP of Keras c) LSTM d) GRU recurrent with dropout neural network e) GRU with two layers f) CNN

5.2 Comparison of time and mean squared error

In that part of this chapter, I made a comparison for execution time of iterations between each model. As is shown in figure 37 the best performance on this topic is from MLP Scikit-Learn with a time of execution 7.261 seconds. MLP for Keras has iterated with a time of 17.4753 seconds which means that it is the second model faster than others. According to that, MLP models are faster than other models. For this topic, the worst performance was from GRU with a recurrent dropout neural network with a time of 433.333 seconds. Another topic that I will discuss here is the mean squared error. The mean squared error represents the square average of the difference between a variable's observed and predicted values.

A larger MSE means that the data values are widely distributed around its central mo ment (mean), while a smaller MSE means otherwise and it is certainly the chosen and/or des ired option because indicates that the data values are scattered near to its central moment(mean), which is generally fantastic.

In our case, the model with the lowest MSE is CNN while the highest MSE error has reached by MLP Scikit-Learn. This means that CNN performs better than other models.

	Mean Squared Error	Time
ARIMA	0.107162	135.728895
MLP Scikit Learn	0.212657	7.261529
MLP Keras	0.165907	17.475357
LSTM	0.179165	265.514015
GRU with recurrent Dropout	0.100899	433.333039
GRU with 2 layers	0.141355	173.664032
CNN with multiple Conv1D	0.089971	79.444069

Figure 37. Mean Squared Error and Time of implementation of each mode

5.3 Comparison of true prices with prices our model predicted

The below graphs show the actual prices and predicted ones from six different models during the 120 coming days. As you can see, there are two lines on the graphs. The actual price is represented by the green line. The orange line represents the predicted prices. To understand which model has made the best prediction I will interpret the six graphs.

The first graph that is shown in figure 38 represents the prediction made by the MLP of Scikit-Learn. During 43 first days, this model has predicted an increase in Bitcoin price. This changes in the interval of 44 to 81 days where the predicted price will decrease compared with the actual one. In the upcoming 28 days this model has assumed that the predicted price will have almost the same value as the actual one. In the last days, the predicted price has reached the maximum growth of these 120 days.

Figure 38. Predicted price by MLP of Scikit-Learn model

The second graph that is shown in figure 39 represents the prediction made by the MLP for Keras model. As we can see, most of the days have a lower predicted price compared to the actual one. In almost 20 days the MLP for Keras model has predicted that the price will

be the same as the current one. The largest increase of predicted price is recorded from day 105 to 117 and in the last three days, the price will decrease again.

Figure 39. Predicted price by MLP for Keras model

The third graph that is shown in figure 40 represents the prediction made by the LSTM model. Most of the days, this model has predicted a very small decrease in predicted price compared to the current one. In almost 30 days this predicted price has no change from the actual price. In the last 15 days, the predicted price has reached the largest increase.

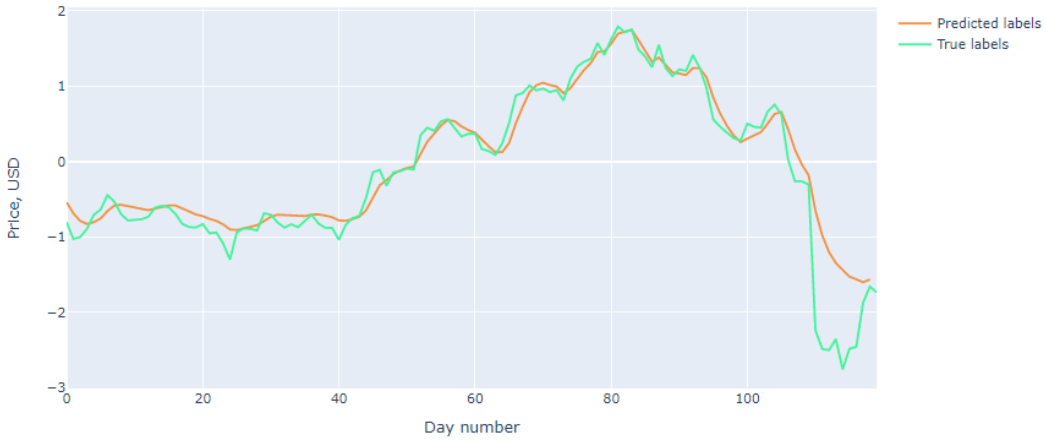


Figure 40. Predicted price by LSTM model

The fourth graph that is shown in figure 41 represents the prediction made by the GRU with a recurrent dropout model. As we can see this model has predicted no decrease in bitcoin price during the upcoming 120 days. Most of the days the predicted price has an increase compared to the current one. The largest increase in the predicted price has reached in the last 10 days. A large number have the days where the predicted price has no change compared to the current one.

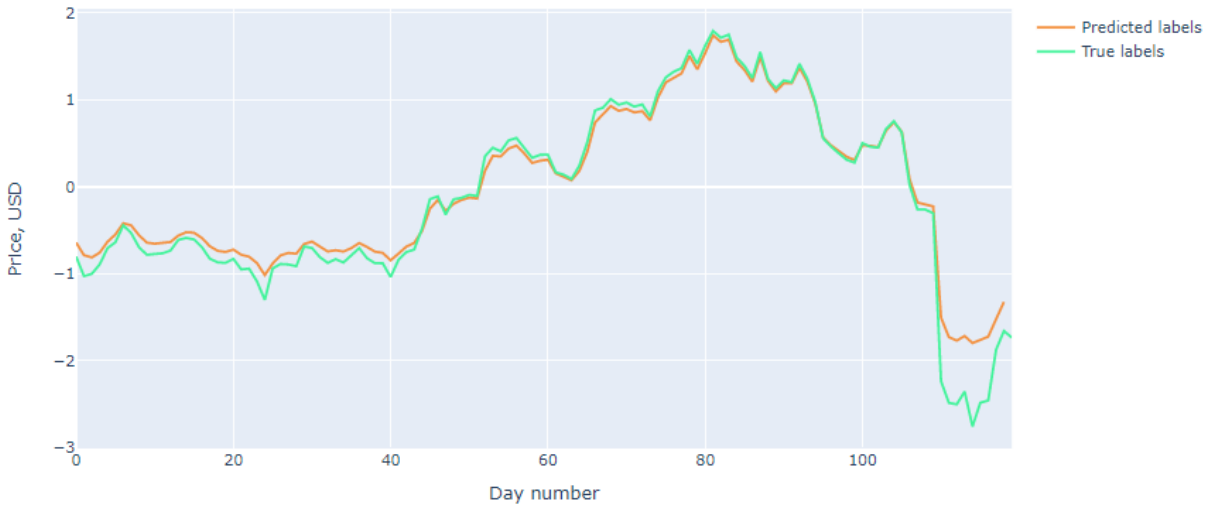


Figure 41. Predicted price by GRU with recurrent dropout model

The fifth graph that is shown in figure 42 represents the prediction made by GRU with two layers model. This model has predicted almost the same prices as the fourth model. Their similarity is that neither of those two models has predicted a decrease in Bitcoin price. According to GRU with two layers model, in 43 first days, the Bitcoin price will increase. From 44 to 106 days the price of Bitcoin will have no change. The largest increase in the predicted price has reached is in the last 14 days.

Figure 42. Predicted price by GRU with two layers model

The sixth graph that is shown in figure 43 represents the prediction made by the CNN model. This model has some similarities with the two GRU models according to the lack of price reduction. In most days, this model has predicted an increase in Bitcoin price. The largest higher values the prediction price has reached in the last 10 days. In the remaining days, the prices have no change.

Figure 43. Predicted price by CNN model

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

The last chapter gives the conclusions about this thesis and the future work that other researchers can make in this topic. The first section is about the conclusions. This part I have split in three different parts. The second section is about future work.

6.1 Conclusions

First, I will give the conclusions about the fitting of the data. According to the results MLP from Keras model seems to have a perfect fitting to data. LSTM at the 17 first epochs has overfitting to data but later the validation loss is decreased and has made a fitting to data. The GRU with recurrent dropout neural network seems to have a fitting to data and when we add another layer the difference between two losses has increased. And the CNN model has overfitting of the data.

The second part is a conclusion based on the MSE results. The model with the lowest MSE is CNN while the highest MSE error has reached by MLP Scikit-Learn. This means that CNN performs better than other models.

The third part of the conclusions is about Bitcoin price prediction. A better prediction is that one where the actual price is very close to the predicted one. In our cases three are the models that has this kind of information. GRU with recurrent dropout, GRU with two layers and CNN have predicted better than three other models.

6.2 Future Work

Future studies can focus on some different models to see if there are other models than perform better than these three, I concluded in this paper.

Another future study can be about GRU with two layers model since it is a model that is rarely used. To see its architecture more carefully, how it works and to understand why adding a new layer makes a model better.

REFERENCES

[1] «Techburst,» [Online]. Available: https://techburst.io/short-guide-on-how-deep-learning-really-works-81a588541b24.

[2] S. A. a. J. P. John Mern, «Using Bitcoin Ledger Network Data to Predict the Price of Bitcoin».

[3] E. S. a. L. Wang, «"Bitcoin Price Prediction Using Ensembles of Neural Networks"».

[4] J. K. a. H. I. Suhwan Ji, «"A Comparative Study of Bitcoin Price Prediction Using Deep Learning",» 25 September 2019.

[5] D. S. Virk, «"Prediction of Bitcoin Price using Data Mining",» in *School of Computing National College of Ireland*.

[6] S. a. P. Mehta, «"Prediction of Bitcoin using Recurrent Neural Network",» *Blue Eyes Intelligence Engineering & Sciences ,* March 2020.

[7] R. O. E. D.-M.-H. a. F. C. Leonardo Felizardo, «"Comparative study of Bitcoin price prediction using WaveNets, Recurrent Neural Networks, and other Machine Learning Methods"».

[8] K. S. Olti Qirici, «"Bitcoin Price Prediction with Neural Networks"».

[9] S. McNally, «"Predicting the price of Bitcoin using Machine Learning",» in *The School of Computing National College of Ireland,* 9th September 2016.

[10] M. Appel, «"Bitcoin price prediction using Deep Neural Networks",» in *The Artificial Intelligence University of Amsterdam,* June 2016.

[11] M. H. S. a. I. A. Ziaul Haque Munim, «“Next-Day Bitcoin Price Forecast”,» in *Risk and Financial Management*, 20 June 2019.

[12] «Bitcoin,» [Online]. Available: https://www.bitcoin.com/.

[13] «Elite Data Science,» [Online]. Available: https://elitedatascience.com/python-machine-learning-tutorial-scikit-learn.

[14] «GitHub,» [Online]. Available: https://github.com/keras-team/keras.

[15] «PLURALSIGHT,» [Online]. Available: https://www.pluralsight.com/guides/regression-keras/.

[16] «towards data science,» [Online]. Available: https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47.

[17] «toward data science,» [Online]. Available: https://towardsdatascience.com/implementation-of-rnn-lstm-and-gru-a4250bf6c090.

[18] «Tensorflow,» [Online]. Available: https://www.tensorflow.org/tutorials/images/cnn.

[19] «Analytics Vidhya,» [Online]. Available: https://www.analyticsvidhya.com/blog/2018/08/auto-arima-time-series-modeling-python-r/.

[20] «Machine Learning Mastery,» [Online]. Available: https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/.

Appendix A

Definitions

Nr.	Features	Definitions
1.	CoinMarketCap	The most-referenced price-tracking website in the fast-growing cryptocurrency space for crypto assets worldwide.
2.	ConvNet	A subset of deep neural networks, most commonly used for visual imaging studies.
3.	WaveNets	A deep neural network to raw audio generation.
4.	CPU	When you run your deep nets on a GPU, the CPU does little computation.
5.	GPU	This is a single chip processor used for intensive graphical and mathematical computations that releases CPU cycles for other functions.
6.	Bitcoin Core Price (BTC)	Price of Bitcoin.
7.	Money Supply	The Bitcoin Core amount in circulation.
8.	Daily Transactions	The number of transactions included in the everyday blockchain.
9.	Inflation rate	The federal fund rate will decide the shape of the economy's future interest rate.

10.	Seasonal Decomposition	The Seasonal Decomposition procedure breaks down a sequence into a seasonal component, a combined trend and cycle component, and a component called "error."
11.	look_back	The number of previous time phases to be used as input variables for predicting the next time cycle
12.	MinMaxScalar	A scikit-learn library pre-processing package for standardizing the dataset.
13.	tanH function	A rescaling of the logistic sigmoid, so that its outputs vary between -1 and 1.
14.	Sigmoid function	Sigmoid removes the issue of gradients in the model of machine learning when training.
15.	ReLU	An activation function that does not simultaneously activate all neurons.
16.	Keras	An open-source library of neural networks written in Python.
17.	Scikit-Learn	Python 's free machine-learning library. It features various algorithms such as supporting vector machines, random forests, and k-neighbors, as well as supporting numerical and science libraries such as NumPy and SciPy.
18.	Grid search	The process of data scanning to configure optimal parameters for a particular model

19.	Hyperparameters	A parameter whose value is set prior to starting the learning process.
20.	Cross-Validation	A methodology used to test ML models by training and evaluating several ML models on subsets of available input data on the complementary data subsets.
21.	epochs	An epoch refers to a single period through a full training data collection.
22.	hidden layer number	The number of hidden layers.
23.	learning rate initial	A configurable hyperparameter used in neural network training that has a small positive value, often within a range of 0.0 to 1.0.
24.	alpha	The parameter that determines the size of a step/iteration.
25.	momentum	A set of tricks and techniques designed to accelerate convergence of methods of first order optimization, such as gradient descent (and its many variants).
26.	hidden layer size	The size of hidden layers.
27.	batch size	Number of training examples used in one iteration.
28.	early stop	A type of regularization used to prevent overfitting when training an iterative learner, such as gradient descent.
29.	tol	Tolerance to the stop criteria.
30.	adam optimizer	An adaptive learning rate optimization algorithm specifically

		designed for training deep neural networks.
31.	sequential model	The easiest way to construct a model at Keras. It lets you build a layer-by - layer model.
32.	dense layer	A layer of neurons in a neural network.
33.	dropout	Simply placed dropout refers to missing units (i.e neurons) that are selected at random during the training process of any group of neurons.
34.	mean squared error	It is the sum of the square of the difference between the predicted and the actual target variables, divided by the number of data points, over all the data points.
35.	convolutional layer	The major building blocks used in convolutional neural networks.
36.	feature map	The output of one filter added to the layer above.
37.	pooling layer (downsampling)	The layer that is used to gradually reduce the representation 's spatial size to reduce the amount of parameters and computation in the network, and thus also to control the overfitting.
38.	fully connected layer	The layer where all inputs are connected to each activation unit of the next layer from one layer to another.

39.	flatten	The function which converts the pooled feature map to a single column passed to the fully connected layer.
40.	Maxpooling	A sample-based discretization method that downsamples an input representation (image, hidden-layer output matrix, etc.), reduces its dimensionality and allows for assumptions about features found in binned sub-regions.
41.	Autoregression	A method of regressing the variable by itself on past values.

XH, K BITCOIN PRICE PREDICTION USING DEEP LEARNING 2020