

Implementing Deep Learning Neural Networks to Predict Bitcoin Prices

Using LSTM, Bi-LSTM, GRU, and Transformer

Notebook Link:

<https://colab.research.google.com/drive/19ftBA7U0vyEg4xUwUPxynQ8w5GUhwWCy?usp=sharing>

6532510

Adarsh Manoj

Computer Science BSc (hons)

University of Surrey

17 May 2022

Acknowledgments

Before commencing, working, and submitting this dissertation a standing ovation is deserved for my Parents, whose unwavering support and motivation fuelled and enabled me to pursue the complexities of computer sciences. My mother who does her best to ensure that I am following an earnest and rewarding life, and my father whose hard work and success inspires me daily to keep trying my 110%. A big shoutout to my brother and sister who kept me centred with their regular check-ups and entertaining humour. Furthermore, my friends, whose interesting discussions and unique point of views allowed for a memorable time at the University of Surrey. Lastly, to all the staff at the University of Surrey, and in particular Dr. Zhenhua Feng, whose support and availability allowed me to improve, learn and expand my knowledge in the field of Machine Learning.

Abstract

This paper covers four different deep learning neural network techniques, them being Long-Short Term Memory, Gated Recurrent Units, Bi-Directional LSTMs, and RNNs in conjunction with an encoder Transformer. These models were built for the purpose of predicting Bitcoin prices. All the models were fitted on an ever-increasing dataset, 55 thousand samples and ever-increasing at the time of running, and all models made predictions with a future window of 12 hours. The results indicate that from the four models, the GRU is the best with the lowest MAPE and the highest R2 score, the LSTM as well as Bi-LSTM both represent excellent options for time series forecasting. The only model to fail to meet the set objectives was the RNN-Transformer.

Table of Contents

ACKNOWLEDGMENTS.....	2
ABSTRACT.....	3
SECTION 1: INTRODUCTION	7
MOTIVATION.....	7
AIMS	8
STRUCTURE.....	8
SECTION 2: LITERATURE REVIEW.....	9
DISCUSSING CRYPTOCURRENCIES BLOCKCHAINS AND BITCOIN	9
MACHINE LEARNING (ML) AND DEEP LEARNING (DL)	10
ARTIFICIAL NEURAL NETWORKS (ANNs) AND DEEP NEURAL NETWORKS (DNNs).....	11
CONVOLUTIONAL NEURAL NETWORKS (CNNs)	12
RECURRENT NEURAL NETWORKS (RNNs)	13
LONG SHORT-TERM MEMORY (LSTM)	15
BI-LSTM (BI-DIRECTIONAL LONG SHORT-TERM MEMORY)	17
GRU (GATED RECURSIVE UNIT)	18
TRANSFORMER MODEL	19
RELATED WORKS	21
<i>Regression Based Implementations</i>	21
<i>Table of Previous Related Works</i>	22
<i>LSTM</i>	24
<i>Bi-LSTM</i>	24
<i>GRU</i>	24
<i>Transformers</i>	25
SECTION 3: PROBLEM ANALYSIS AND DESIGN	26
DATA COLLECTION.....	26
DATA PRE-PROCESSING	27
<i>Data Normalization</i>	27
<i>Data Optimization</i>	27
<i>Data Splitting</i>	28
DEVELOPMENT ENVIRONMENT	29
<i>Hardware Specifications</i>	29
CRUCIAL LIBRARIES AND DEPENDENCIES.....	29
<i>Data Pre-Processing</i>	29
<i>Model Development</i>	30

SECTION 4: IMPLEMENTATION.....	31
CONSTANTS AND PARAMETERS	31
<i>Activation Function</i>	31
<i>Optimizers</i>	32
<i>Loss Metric or Cost Function</i>	32
<i>Alternative Metrics for Empirical Risk Minimization</i>	33
<i>Call-backs</i>	35
<i>Epochs and Batch Size</i>	35
MODEL IMPLEMENTATION AND TRAINING	35
<i>Input and Data Shape</i>	36
<i>Additional Layers</i>	37
<i>LSTM Approach</i>	39
<i>GRU Approach</i>	40
<i>Bi-LSTM Approach</i>	42
<i>RNN-LSTM Transformer Approach</i>	43
SECTION 5: RESULTS	46
EVALUATION METRIC R2 SCORE.....	46
LSTM APPROACH	46
<i>Graphical Representation</i>	46
<i>Metric Representation</i>	46
GRU APPROACH.....	46
<i>Graphical Representation</i>	47
<i>Metric Representation</i>	47
BI-LSTM APPROACH	47
<i>Graphical Representation</i>	47
<i>Metric Representation</i>	47
RNN-TRANSFORMER APPROACH	48
<i>Graphical Representation</i>	48
<i>Metric Representation</i>	48
SECTION 6: EVALUATION AND OBJECTIVE ANALYSIS	49
OBJECTIVES.....	50
SECTION 7: REVIEW OF LSEP ISSUES	51
LEGAL ISSUES	51
SOCIAL ISSUES	51
ETHICAL ISSUES.....	52

PROFESSIONAL ISSUES	52
SECTION 8: CONCLUSION	53
FUTURE WORKS.....	53
<i>Developing a Platform for the Models</i>	<i>53</i>
<i>Improving the Models</i>	<i>53</i>
SECTION 9: TABLE OF FIGURES	55
SECTION 10: REFERENCES	56
SECTION 11: APPENDIX	61
NOTEBOOK	61
<i>LSTM Model</i>	<i>61</i>
<i>GRU Model</i>	<i>62</i>
<i>Bi-LSTM Model</i>	<i>64</i>
<i>Transformer Model</i>	<i>66</i>
SAGE FORM.....	68

Section 1: Introduction

Motivation

Money makes the world go round, but what is money but a medium of exchange, the existing medium being fiat currencies, but is that all it can be. Cryptocurrencies are a demonstration of alternate currencies. Cryptocurrencies are any form of currency that exists digitally or virtually and uses cryptography to secure transactions (Kraft, 2021), the first implementation being Bitcoin in 2009, which has a market cap of £600 million, a growth by a factor of 6 within the span of 2 years (Hileman, 2017). The opportunity for individuals as well as businesses to capitalise on this opportunity has already been visualised seeing as the distribution of ownership is fairly even between small and large investors (Kraft, 2021). With all forms of investing, there is a risk of losing the investment, a study conducted found that 55% of investors were in the red (Curran, 2022). This fact instigated a personal investigation into solutions for mitigating loss on cryptocurrency investments.

There are several approaches to construct a system that can anticipate the value of a financial asset, including time-series analysis, moving average techniques, various neural network implementations, and more. Long-Short Term Memory (LSTM), Gated Recurrent Unit (GRU), and Bi-LSTM are the three implementations of recursive neural networks (RNN) that will be discussed in this study to accurately develop models to predict bitcoin prices. A Transformer in conjunction with an RNN for time-series forecasting will also be implemented as part of the project.

This paper focusses on implementing deep learning implementations for time series forecasting on cryptocurrency values, for Bitcoin. A more in-depth explanation of cryptocurrencies is found in the Literature Review.

Aims

This report aims to provide a comprehensive analysis on the various machine learning implementations and their abilities to analyse specific cryptocurrencies. Topics to be covered include machine learning, neural networks, transformers, and more. There will be an explanation on the methodology of developing the desired solution as specified. Conclusions will be drawn about the implementation's effectiveness against determined criteria, as well as an exploration into the potential future works.

- Develop a solid base understanding of the core topics to properly evaluate and implement deep learning neural network models.
- Develop a model(s) that adapts daily, that keeps expanding and learning based on constant changes in the market.
- Implement at least three instances of Neural Network models to accurately forecast the price of three different cryptocurrencies. Models will be parameterised to ideally achieve an R2 score of 0.85 or greater – 0.15 greater than industry standard for asset indicators (Fernando, 2021).
- Create functionality that can be visualised for at least 10 hours to act as an indicator if a cryptocurrency should be sold, bought, or held.

Structure

The structure of the paper is as follows:

[Section 2](#) covers the foundational knowledge required to thoroughly comprehend the following parts, it also covers key formulas, and most importantly previous works.

[Section 3](#) covers the design ideologies and constructs that were put in place to develop the four different models.

[Section 4](#) discusses in detail the actual construction and implementation of the model; it also focuses on the parameters and constants required.

[Section 5](#) displays and portrays the results from the four different models, it also covers the performance metrics that were outputted because of the implementation.

[Section 6](#) focusses on understanding the deeper meaning from the results gathered.

[Section 7](#) is an analysis of the LSEP issues with the implementation Legal, Social, Ethical, and Professional.

[Section 8](#) discusses the summation of the entire project and evaluates if the implementation satisfies the objectives set above, it also covers the possible future directions of this project.

[Section 9](#) contains the table of figures.

[Section 10](#) contains the references utilised.

[Section 11](#) contains the appendix with the additional material, as well as a link to the notebook.

Section 2: Literature Review

This paper implements machine learning innovations to analyse cryptocurrency prices, the two key aspects being the different machine learning implementations and understanding cryptocurrencies.

Discussing Cryptocurrencies Blockchains and Bitcoin

Bitcoin is commonly referred to as the grandfather of cryptocurrencies seeing as it's the oldest implementation of blockchain platforms. It was founded in 2009 by an anonymous individual, and it pioneered what the world knows as blockchain platform. Bitcoin allows for day-to-day transactions to be handled by a complex network of nodes and computers to maintain an immutable record of transactions. It possesses the ability to authenticate trades, hold funds, and is currently the primary currency in El Salvador.

Bitcoin as mentioned earlier is a blockchain platform, nodes to maintain the network, timestamping to secure the network, mining for maintaining the network, and wallets for storing the tokens. The blockchain platform is the foundation for many of the major cryptocurrencies including Bitcoin, which is an open-source Distributed Ledger Technology (DLT) (Vallance, 2021).

The blockchain is a distributed ledger of transactions that is spread across honest nodes to form a computer network known as the blockchain network. The blocks in a blockchain are used to store records which are secured using cryptography, each block contains a hash pointer to link the previous block. It's used to host transaction information which is generated when a "transfer of value between wallets" occurs, to verify the integrity of a transaction a private key is used to authorise transactions out of a wallet and a signature is generated to maintain the integrity of the transaction (Vallance, 2021). The hashing algorithm used to maintain security is SHA-256 which digitally imprints an immutable signature on the block (Appel, 2015). The data stored in the blocks cannot be altered without altering all subsequent blocks, the only way to do that is via wide-scale network collusion.

In a cryptocurrency, the nodes act as a support system for the entire network, they are computers that provide two types of functionalities: validating the blockchain, and relaying of transactions. All nodes in a network persists a copy of the blockchain, this allows for historical transparency, as well as allow new nodes to utilise the blockchain data in the existing node. Nodes are a P2P protocol allowing for the relaying of transactions occurs on a network wide level, with each transaction being broadcasted to multiple other nodes in the network. Nodes engage in consensus, communicate transaction and fund information, confirm transactions and store copies of confirmations, and participate in the creation of new blocks in the chain, to ensure the security of the chain a process known as mining is implemented. (Newman, 2021)

Mining is the process of validating transactions on a cryptocurrency network by solving complex cryptographic algorithms. The mining process is a race to verify the transaction the quickest via Proof-of-Work, which is this is accomplished by solving the Proof-of-Work. The Proof-of-Work is a complex math problem known as the Byzantine Generals Problem, the solution to the problem is a newly generated hash, and if it is equal to or less than the target hash then the new hash can be accepted as a solution but won't be accepted by the blockchain. For the solution to be accepted network wide, at least 51% of network miners need to verify the solution. The implementation of mining has solved the problem of double-spending in cryptocurrencies, as the internal sequential nature of the blockchain will identify the time stamps and through the process of mining void the transaction. (Lamport, 1983)

Due to the open-source nature of Bitcoin, and its reliance on a publicly supported network, the cryptocurrency is incentivized to garner stability for the infrastructure that supports the network. Both the nodes and mining machines upon the completion of respective tasks is rewarded in the cryptocurrency, in fact Miners own 9.7% of Bitcoin in circulation. (Kraft, 2021)

There are many determinants of the value of Bitcoin, some even say the determinants are so vast that it's impossible to take all of them into consideration. This paper will focus on adapting technical knowledge - the key parameters of stocks (Open, High, Close, Low, Volume), as the sole factors influencing Bitcoin.

Machine Learning (ML) and Deep Learning (DL)

The root of this paper is implementations of Deep Learning models which are Neural Networks to perform time series analysis tasks. NNs are a subset of Machine Learning, and Machine Learning is a subfield of Artificial Intelligence. Machine Learning is defined as "The field of study that gives computers the capability to learn without being explicitly programmed" (Arthur Samuel et al. 1959), the end goal being an implementation that increases in accuracy. The learning systems in a machine learning algorithm can be broken into three aspects: a decision process, an error function, and a model optimization process. The decision process allows for the algorithm to recognize patterns in the data, the error function aims to assess the accuracy of the model by evaluating the model's predictions, and the optimization process implements weights to reduce the discrepancy between predicted and training values. The entire process will be repeated until a satisfactory threshold for accuracy is met.

Deep learning defers from 'classical' machine learning by possessing the ability to process data with limited human intervention, its widely referred to as a 'scalable machine learning' due to its ability to handle both labelled and unlabelled data. As a result, deep learning can be supervised,

unsupervised, and semi-supervised (Ching et al., 2018). Deep learning branches two NNs, them being Artificial Neural Networks (ANN), and Deep Neural Networks (DNN).

This paper will focus more on various implementations of ANNs and their use in stock price modelling, with the model implementing learnings from both a supervised and unsupervised manner.

Artificial Neural Networks (ANNs) and Deep Neural Networks (DNNs)

Artificial Neural Network are defined as a network of neurons which utilise multi layered non-linear processing units, to handle complex situations in a manner like that of a human brain. These 'units' are spread across various layers; the layers can be of various types, but they all share the functionality of altering the inputs. The outputs tend to be non-linear allowing for the handling of complex implementations. ANNs are comprised of an input layer, a hidden layer, and an output layer. ANNs encompass any form of deep learning, the difference between ANN and DNN is that ANNs can have only one hidden layer, whereas DNNs can have multiple – refer image below. The term ANN and DNN are used interchangeably. (Lancashire, Lemetre and Ball, 2008)

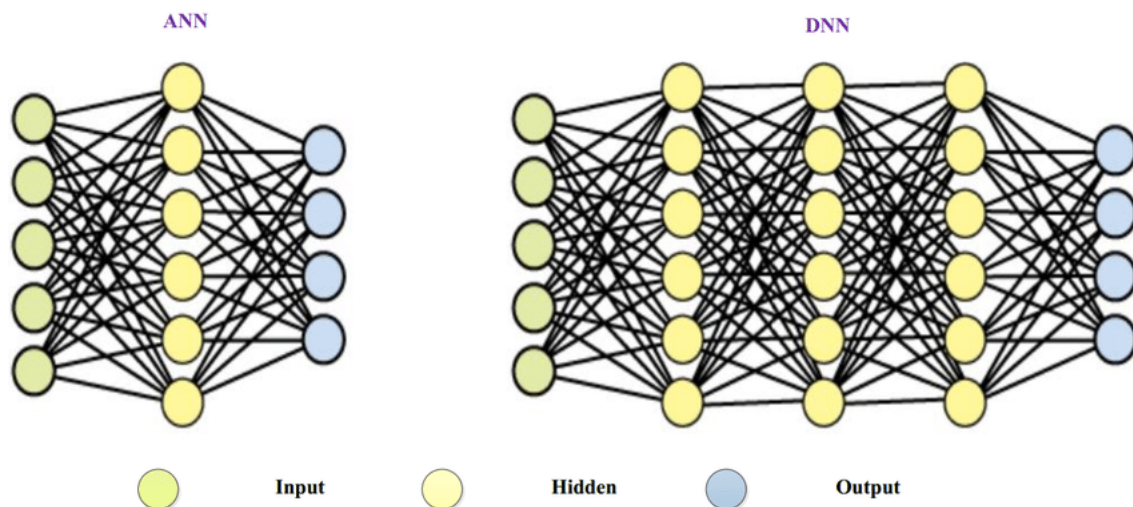


Figure 1: ANN vs DNN high level architecture. (Aslam et al., 2019)

The hidden layer in an ANN is where the mathematical extrapolations of the input are handled with the help of Activation Functions, which define the output of a node in the network given a set of inputs. An activation function is a nonlinear function used to train a neural network by defining the output of a neuron given a set of input (Chigozie et al., 2018), importantly it can add either linearities or non-linearities into the neural network.

The highlight of ANNs is their ability to handle and process non-linear data using its Activation Function(s). The benefits are seen when having to map complex relationships between input and output. One issue with ANN is that it's got limited processing ability when handling large datasets as the number of trainable parameters increase in relation to the complexity of the data. Other

drawbacks include its inability to capture sequential information, and the common prevalence of Vanishing and Exploding gradient. (Aslam et al., 2019)

ANNs and DNNs can be sub-classified into Feed Forward Neural Networks (FFNN), Convolution Neural Networks (CNN), Recurrent Neural Networks (RNN) each tailored to rectify some of the drawbacks mentioned earlier. FFNNs only process information in one direction, forward, and can be broken down into two sub-classes, Single-Layer Perceptron and Multi-Layer Perceptron, however the singular direction of data processing culminates in a neural network that's not tailored to handling the sequential nature of time series data. CNNs share weights between spatial regions, whereas RNNs share across timesteps. RNNs on the other hand are recurrent and possess the ability to understand temporal information.

Convolutional Neural Networks (CNNs)

CNNs are networks designed for processing multi-dimensional data, typically for images, but also on other datasets like time series. This network was pioneered in the 1980s by Dr. Yann LeCun, and is actively discussed and implemented, the reason being the networks' ability to automatically extract features from input data without the need for data pre-processing (Yamashita et al., 2018). CNNs are a type of multi-layered FFNN, and the basic components are three types of layers: convolutional, pooling, and fully connected layers.

The convolutional layer aims to extract the features from the input data and is comprised of multiple convolutional kernels which process feature maps. The feature map is generated by each kernel sharing all spatial locations of the input amongst several kernels. The mathematical representation of each feature value at location (i, j) in the k feature map of the l layer is as shown below:

$$z_{i,j,k}^l = w_k^l * x_{i,j}^l + b_k^l$$

Given that w_k^l is the weight vector of the l layer and k feature, $x_{i,j}^l$ is the input centred at the specified location on the l layer, and b_k^l is the bias term of the l layer and k feature.

The pooling layer is typically sandwiched between two convolutional layers and is implemented with the aims of reducing the dimensions of the feature maps. Like feature maps in convolutional layers, the 'pooled' feature map is connected to the respective feature map in the previous convolutional layer. (Yamashita et al., 2018) Through iterations of the convolutional and pooling layers it's possible to extract high-level feature representation.

The fully connected layer is implemented to perform high level reasoning and does so by connecting all neurons in the previous layer to that in the current layer (Sharma and Mehra, 2019). In doing so, this layer can formulate global semantic information.

CNNs possess great prowess in processing highly complex data structures, its architecture enables for little to no data pre-processing, inherent feature extraction, and weight sharing. However, is typically used in the field of Computer Vision, hence, some adaptations may be necessary for our implementation.

Recurrent Neural Networks (RNNs)

The recurrent neural network (RNN), a type of ANN in which the connections between nodes create a directed graph over time, allowing and enabling it to behave with a temporal aspect. RNNs, which are derived from FFNNs, can process variable length sequences of inputs by using their internal state (memory) (Abiodun et al 2018). RNNs are said to be theoretically Turing complete, which signifies the system's ability to recognise and or decide alternative data-manipulation patterns. RNNs can provide both finite and infinite impulse responses, this provides the network with the ability to have internal additional stored states. The figure below represents an RNN as well as a visualization of how the NN would like if the recurrent aspect was visualised in layers. RNNs typically comprise of four components: an input layer, a hidden state, weights, and an output layer.

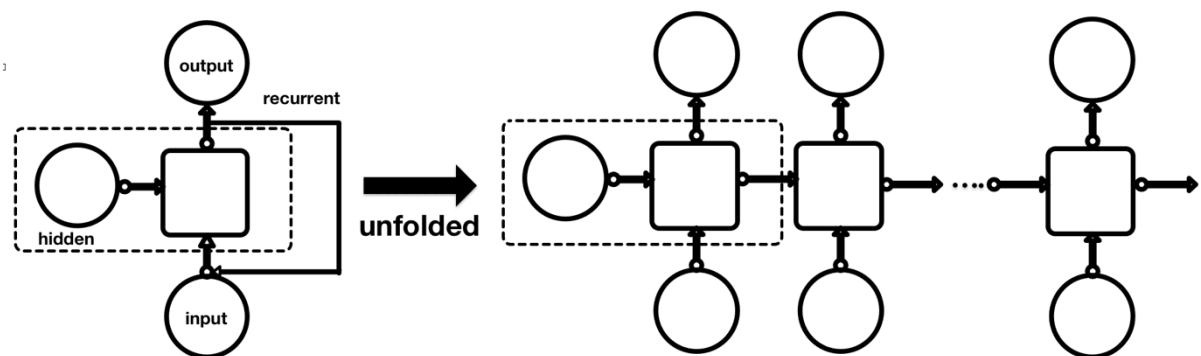


Figure 2: RNN unfolded architecture.

The input layer handles the input data, the input layer is theoretically able to handle an input of any size, seeing as the model size is not correlated to the input parameters. The hidden state $h(t)$ represents the 'memory' at a given time(t), and is mathematically represented as

$$h(t) = f(U * i(t) + W * h(t - 1))$$

where f , is the chosen activation function, U and W are weight matrices. RNN establishes an ability to secure a recurrent connection within a neural networks' hidden state. The weights in an RNN can be broken into three different weight matrices, U, V, W , and are shared throughout the process. U represents the input to hidden parametrized relationship, V represents the hidden to output parametrized relationship, and W represents the hidden-to-hidden parametrized relationship.

The ability to process inputs of any length, the lack of correlation between input size and model size, as well as the “memory” gained from hidden states, hence why there are many interpretations of RNNs tailored for specific tasks (Salehinejad et al., 2018). The newer interpretations aim to rectify some of the challenges faced with RNNs, them being slow computation, complicated access of information for older temporal memory.

Long Short-Term Memory (LSTM)

Long short-term memory (LSTM) was derived from FFNN, a type of ANN by Hochreiter and Schmidhuber. (Hochreiter and Schmidhuber, 1997) Their discovery aimed to rectify the issue of vanishing and exploding gradients, as well as avoid long-term dependency difficulties. The key difference between an LSTM approach and a traditional RNN, is that LSTMs implement 'memory units' commonly referred to as an activation layer, in addition to the pre-existing neurons in the hidden state of an RNN. The concept of the memory unit allows for an LSTM network to learn tasks that depend on events that occurred x steps ago, allowing it to also possess long-term information. Hence, the name Long-Short Term Memory.

The image below represents an LSTMs architecture when unfolded, where X_t is the input at a given time t , and h_t is the hidden state. (Olah, 2015)

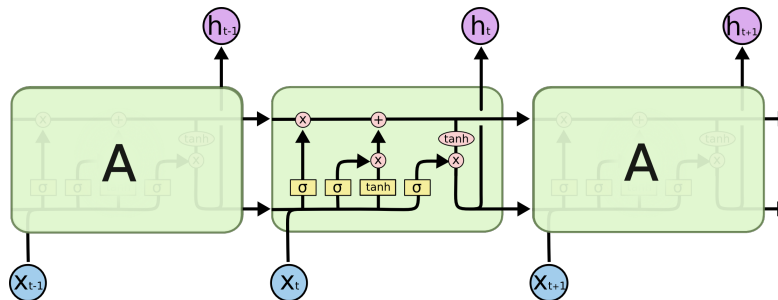
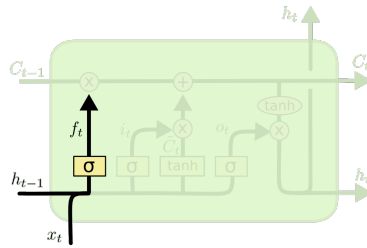


Figure 3: LSTM unfolded architecture with internal aspects. (Olah, 2015)

LSTMs can prioritize data inputs via structures called gates that can add or remove information to the cell state, the gates themselves are comprised of a pointwise multiplication operation and a sigmoid neural net layer (Staudemeyer and Morris, 2019). Sigmoid neural net layers are used for binary classification, in an LSTM the sigmoid weight is either 0 or 1, with 0 indicating no information passes and vice versa.

LSTMs architecture is comprised of 4 different neural network layers - the primary focus is on the 'cell state' in the hidden layer which passes through the entire chain with minimal computational interactions and is mathematically represented below. Note that in the representation below, x_t is the input at time t , W_f and b_f are the weight matrix and coefficient for the forget-gate, W_i and b_i are the weight matrix and coefficient for the input-gate, and W_c and b_c are the weight matrix and coefficient for the cell state.

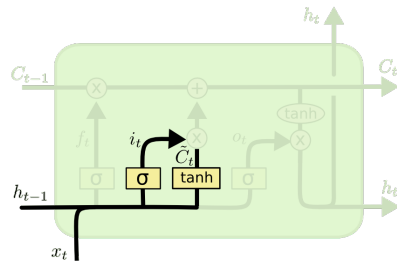
First the LSTM model decides what information is forgotten in the cell state by implementing a forget-gate sigmoid layer (f_t), this variable indicates the information that's left in the cell state.



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

Figure 4: LSTM internal steps, forget-gate.

The next goal of the model is to provide candidate values to retain (\tilde{C}_t), it does so by utilising an input-gate sigmoid layer (i_t). (i_t), reflects the information that will be appended on to the next step. The figure below reflects how the candidate values handles the information generated from the input-gate layer. Note that the activation functions can be changed, tanh is default for Keras, however other options like ReLu and LeakyRelu exist.

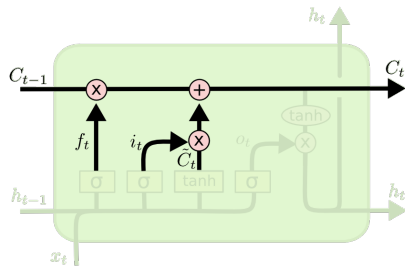


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Figure 5: LSTM internal steps, input gate, and candidate values.

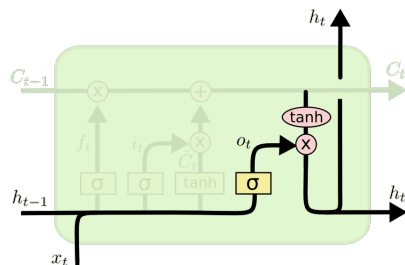
The cell state analyses the candidate values in conjunction with the previous cell state's value to generate a new cell state value.



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

Figure 6: LSTM internal steps, updating Cell state.

The last step is to determine the output, which will be a filtered version of the cell state provided by the “output-gate”. The output is determined by the multiplication of the sigmoid layer and the activation function (i.e., tanh) value of the cell state.



$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Figure 7: LSTM internal steps, hidden layer and output values.

LSTMs are renowned for their ability to tackle the problem of vanishing and exploding gradients, to understand long-term temporal information because of constant error backpropagation amongst memory cells, and its ability to handle noisy data (Sak, Senior and Google, n.d.). Whilst LSTMs do rectify the issue of vanishing and exploding gradients, it's not entirely fixed due to the constant data transfer between cells (Olah, 2015), as well as the complicated nature of the cells. Other issues faced with LSTMs are the complex computational requirements, and whilst the issue of vanishing gradients is addressed it's not entirely rectified. (Staudemeyer and Morris, 2019)

Bi-LSTM (Bi-directional Long Short-Term Memory)

Bi-LSTM is the process of creating a neural network which sequences the information both forwards (future to past) and backwards (past to future). The reasoning behind a Bi-LSTM implementation is due to the evidence that context of the data is interconnected as opposed to solely a linear interpretation, this allows for both “past” and “future” temporal information to come under consideration (Huang, Xu and Yu, 2015). In a Bi-LSTM implementation there will be two LSTM models that will be combined by a Merge Step, which is typically through concatenation, but can be through sum, multiplication, or averaging.

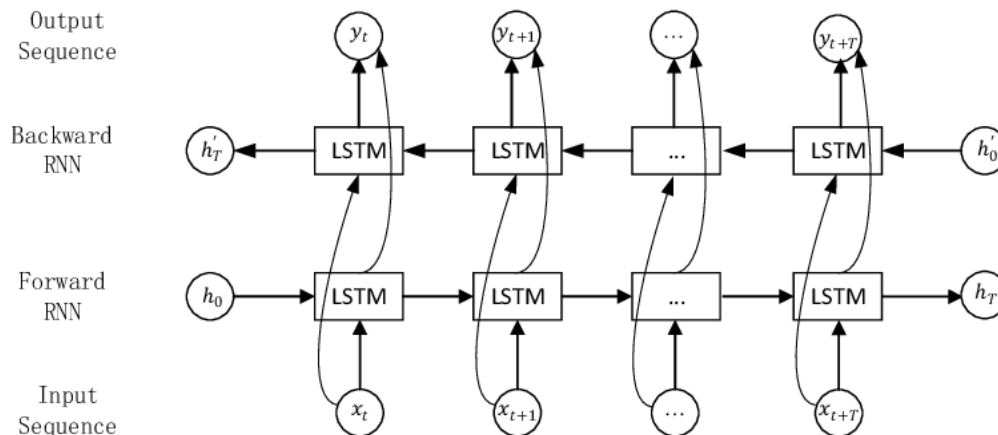


Figure 8: Bi-LSTM high level architecture diagram.(Pintelas et al., 2020)

The architecture of an unfolded Bi-LSTM implementation is displayed above. Where x_t represents the input layer at time t , h_t represents the hidden state, and y_t represents the output layer. The difficulties faced with Bi-LSTMs are very similar, if not the same for LSTMs, hence the issues of vanishing gradients still need to be taken into consideration. (Huang, Xu and Yu, 2015)

GRU (Gated Recursive Unit)

(Chung et al., 2014) reduced the LSTM structure and created GRU, a new deep learning architecture that integrates long-term and short-term memory. GRU solves the problem of gradient disappearance and explosion in classic recurrent neural networks (RNNs) when learning long-term reliance. A GRU implementation contains gating units that moderates the information flow within the unit, unlike an LSTM the GRU can moderate the flow of information in the absence of memory cells. The architecture of a GRU implementation is shown below. The two important aspects of a GRU are the Update and Reset gate.

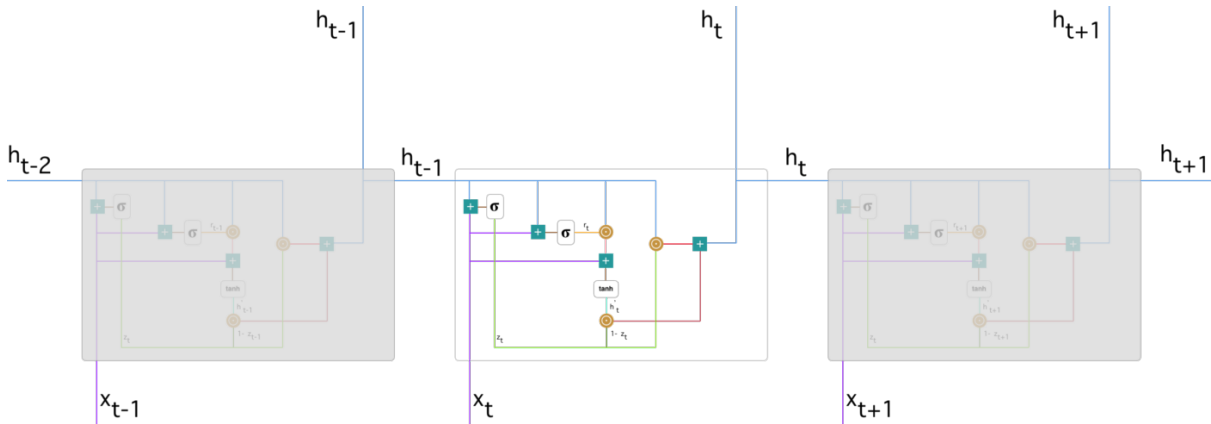


Figure 9: GRU internal steps, unfolded. (Chung et al., 2014)

The first step of the model is the update gate update gate, which aims to the aide the model in determining the amount of information to retain for future use. The weighted input value and the weighted previous value get processed by a sigmoid activation function; the formula looks like:

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

The update gate plays a key role in eliminating the risk of vanishing gradients. The next step is the forget gate, which uses the same formula as the update gate but has differing weights and applications. The forget gate is implemented to determine from the model the amount of information to be forgotten from the past.

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

Next there is the current memory content which utilises the forget gate to determine the necessary information from the past, at this stage a variety of activation functions can be used, the standard implementations use a tanh activation function, and the equation is:

$$h'_t = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

The last step is to determine the final memory at the current timeframe, which utilises the current memory in addition to the previous value with the update gate. The formula looks like:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$

The drawbacks of GRUs and LSTMs are like that of traditional RNNs, with issues regarding its inability to support parallel processing, the long-term learnings of the network are inhibited when faced with large computational data, LSTMs do outperform GRUs in long-term memory (Chung et al., 2014). GRUs outperform LSTMs in resource management, ability to train data more accurately with less data. do require less computational power and tend to be more resource efficient, they also are simpler and allow for easier modification. Another drawback is that there is no distinct modelling of the long- and short-term dependencies, also the distance relationship between position are linear.

Transformer Model

Transformer models are a framework that makes sequence-to-sequence predictions using attentions and weights, to rectify recursion and enables parallel. It follows an encoder-decoder architecture which is a neural network where the decoder operates in reverse of the encoder and culminates in a fully connected point-wise feedforward neural network. (Vaswani et al., 2017)

“... the Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution.” – Attention Is All You Need 2017 (Vaswani et al., 2017)

The Transformer learns an information-passing graph amongst inputs, as it's unable to analyse inputs sequentially. The beauty of Transformers is that it alleviates the pains felt through RNNs (like the ones above), and the primary benefit is the infinite reference window, which enables for the attention mechanism to be fully optimized. The self-attention features enable for the model to capture both long- and short-term context.

The encoder in a Transformer encapsulates two layers, a self-attention layer and a feed forward neural network layer (FNN). The inputs pass through the self-attention layer first before moving on to the FNN. The decoder extends the encoder with a new Encoder-Decoder Attention layer between the self-attention layer and the FNN.

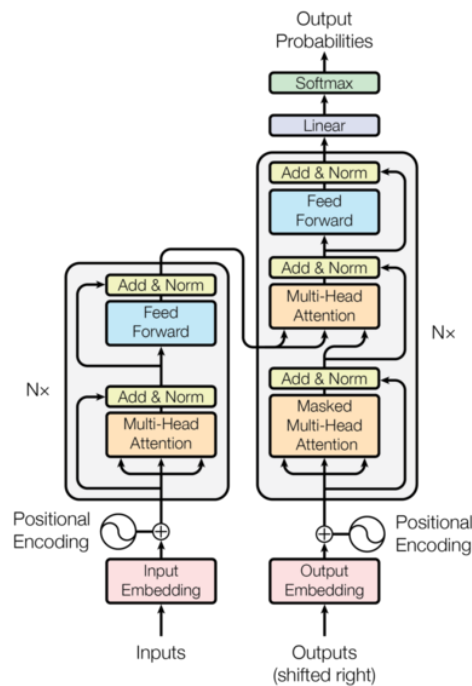


Figure 10: Transformer, encoded-decoder high level architecture. (Vaswani et al., 2017)

The self-attention layer works by splitting inputs into three vectors: query, key, and value. After getting the vectors the next step is to determine a score for each input against the other inputs which determines the focus on other inputs at a given position, the score is determined by taking the dot product of the given inputs query vector with the other inputs key vectors. Next, divide the scores by the square root of the dimension of the key vectors, then utilise a SoftMax function to normalize the scores, this represents the ‘weight’ of each word expressed at the given position. The next step is to sum up the product of each value by the SoftMax score, to shift focus to the important sequences. This completes self-attention.

The benefits with a Transformer model are Non-sequential, and Positional embedding. The non-sequential nature of processing the data allows for Transformer networks to be processed in parallel allowing for the network to be significantly efficient in processing power and time. Positional embedding allows for each inputs ‘position’ to be processed and retained for future use; this allows for the Transformer to identify the location of a given input. Unlike LSTM or RNN, Transformer has no recurrence and no convolution. Instead, it utilizes the positional encoding added in the input embeddings, to model the sequence information. (Khan et al., 2021)

Transformers were initially implemented for Natural Language Processing (NLP), the founding paper, “Attention Is All You Need” backed by Google, focussed their machine on translation tasks. Their findings were significant as their model outperformed the existing benchmark by achieving a score of 28.4 BLEU (Bilingual Evaluation Understudy). (Vaswani et al., 2017)

Related Works

Regression Based Implementations

Time series regression is a series of values of a quantity obtained at successive times, often with equal intervals between them (Pathak, 2021). The difficulty with time series regression is the noise and influence of many factors on the stock's value. The dynamic relationship between the linearity of data and nonlinearity of factors affecting the stock are what prove to be the challenge. The traditional method of time series analysis for price prediction would be ARIMA (Autoregressive Integrated Moving Average). According to Wold's decomposition theorem, ARIMA theoretically can understand a wide-sense stationary time series with seasonality. ARIMA is not in the scope of this paper due to its preference for smaller datasets, difficulty in predicting inflection points, and long-term forecasting is poor. The IEEE paper published in 2018 directly compared ARIMA to an alternate model and ARIMA was found to have 19.6% more error rates obtained. (Siami-Namini, Tavakoli and Siami Namin, 2018)

The table below indicates the methods as well as the R2 score, an R2 score is a coefficient of determination that interprets the proportion of variance in predictability. For financial predictions the accepted target R2 score is 0.7 (Fernando, 2021), by that standard, these models are off from modern day levels.

Multi-linear regression model	R2 score: 44% for LTC and 59% for BTC
Logistic regression and linear	LR: 66%
discriminant analysis	LDA: 65.3%

Figure 11: Past works, analysis of linear methodologies.

Multi-linear regression is defined as a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The multi-learning regression model yields a straight line that determines a quantitative relationship between a dependent variable and two or model independent variables. (Hamayel and Owda, 2021)

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic mode.

Linear discriminant analysis is a generalization of Fisher's linear discriminant, a method used in statistics and other fields, to find a linear combination of features that characterizes or separates two or more classes of objects or events.

The machine learning algorithms that were implemented as an alternative to those discussed in the report which are regression based.

Table of Previous Related Works

ARTICLE TITLE	MEASURED VARIABLE	DATA RATE	DATE RANGE	MODELS	FINDINGS
CRYPTOCURRENCY PORTFOLIO MANAGEMENT WITH DEEP REINFORCEMENT LEARNING (JIANG AND LIANG, 2017)	Bitcoin (USD)	Daily	Jun-15 to Aug-17	CNN with Deep RL	Implemented for portfolio management, with 30-minute trading window can tenfold portfolio in 1.8 months.
PREDICTING THE PRICE OF BITCOIN USING MACHINE LEARNING (MCNALLY, ROCHE AND CATON, 2018)	Bitcoin (USD)	Daily	Aug-13 to Jul-17	Bayesian RNN and LSTM	The ideal temporal duration for LSTMs are 100 units, and 20 for RNN. Attained an accuracy of 52% and RMSE of 8%.
BITCOIN TECHNICAL TRADING WITH ARTIFICIAL NEURAL NETWORKS (NAKANO, TAKAHASHI AND TAKAHASHI, 2018)	Bitcoin (USD)	15-min	Jul-16 to Jan-18	ANN	Strong correlation between lower model performance and higher input data.
A COMPARATIVE STUDY OF BITCOIN PRICE PREDICTION USING DEEP LEARNING	Bitcoin (USD)	Daily	Nov-11 to Dec-18	DNN, LSTM, CNN, ResNet, and more.	For prediction problems LSTMs are the most successful mode, DNNs is best for classification problems. Profitability analysis concluded

(JI, KIM AND IM, 2019)					that classification is better as an indicator for trading.
WHAT DRIVES CRYPTOCURRENCY PRICES? (SMUTS, 2019)	Bitcoin (USD)	Hourly	Dec-17 to Jun-18	LSTM	Input data also considered Google trends, and Telegram chats. Potentially an input parameter to consider.
BITCOIN PRICE PREDICTION USING MACHINE LEARNING: AN APPROACH TO SAMPLE DIMENSION ENGINEERING. (CHEN, LI AND SUN, 2020)	Bitcoin (USD)	Daily	Feb-17 to Feb-19	LR, LDA, RF, SVM, and LSTM	LSTM outperformed LR and LDA by 2% when using accuracy as a metric.
A NOVEL CRYPTOCURRENCY PRICE PREDICTION MODEL (HAMAYEL AND OWDA, 2021)	Bitcoin (USD)	Daily	Jan-18 to Jun-21	LSTM, GRU, Bi-LSTM	GRU was the best model of the three with an MAPE score of 0.24%. Bi-LSTM performed the worst, even on various datasets.

Figure 12: Table of previously related works.

LSTM

In recent years, LSTM has been applied in the field of stock market forecasting in markets and indices all over the globe. (Chen, Li and Sun, 2020) used an LSTM model to predict China's various indices, like the Shenzhen stock market. (Jiahong Li, Hui Bu and Junjie Wu, 2017) introduced the concept of stock indicators in conjunction with investor sentiment based on LSTM architectures to predict the CS1300 indices value. The research results highlighted that the model was better than older support vector machine methodologies, however, this model does not reduce the dimension of stock indicator. (Jiawei and Murata, 2019) worked to identify the weighted determinants for factors affecting stock market trend prediction through the LSTM model, which employed a sentiment analyser to convey financial news and a pre-processing technique to minimise the dimension of stock features for stock trend prediction.

Bi-LSTM

The Bi-LSTM model has been implemented for time series analysis with conflicting results. One study published in the 2019 IEEE Big Data indicated that Bi-LSTM model outperformed the traditional LSTM by 37.78% (Ji, Kim and Im, 2019). This study focussed on a time series analysis of indices and stock prices, however, a paper that focussed on crypto currencies revealed an alternate conclusion, that the Bi-LSTM model underperformed by a factor of 4 when looking at the MAPE (Mean Absolute Percentage Error) (Hamayel and Owda, 2021). This paper also revealed that the model struggled with Bitcoin, as when using the Litecoin dataset the model was outperformed by the LSTM model by 32%.

GRU

GRUs were first introduced by Cho in 2014 as an extension of LSTM. A study that compared the efficacy of machine learning algorithms on various cryptocurrency sets deduced that GRUs have the lowest error rates with a MAPE of 0.24% and a Root Mean Squared Error (RMSE) score that was 41% of the LSTM's value on their Bitcoin dataset. (Chung et al., 2014)

The paper published by UCLA directly compared GRUs against LSTMs and their findings were like that of Hamayel et Al. The table below highlights the differences in the two models' abilities for time series forecasting.

Statistics	RMSE		MAPE	
	LSTM	GRU	LSTM	GRU
mean	57.23	44.77	0.0053	0.0035
min	46.37	40.30	0.0037	0.0028
max	73.65	54.28	0.0076	0.0053
std	7.43	3.30	0.0011	0.00060

Figure 13: GRU related works table.

Transformers

Transformers have shown great modelling ability for long-range dependencies and interactions in sequential data and thus are appealing to time series modelling. Many variants of Transformer have been proposed to address special challenges in time series modelling and have been successfully applied to various time series tasks, such as forecasting, anomaly detection, classification (Zhou et al., 2021). However, for Bitcoin price prediction there doesn't seem to be too many implementations on Bitcoin's dataset, there is however a paper in the 2021 International Conference on Data Mining Workshops (ICDMW) that uses the Dogecoin dataset on a Multi-Head Self-Attention Transformer, their implementation yielded an accuracy of 98.46%, and an R2 value of 0.8616, matching existing state-of-the-art implementations, and beating the industry standard by 23.08%. (Agarwal et al., 2021)

Section 3: Problem Analysis and Design

This section analyses and breaks down the architectural decisions that need to be implemented to properly develop neural networks that perform time series regression. This section is structured with the ideology of preparing the environment to handle the model development, as such the necessary initial steps are Data Collection, Data Pre-Processing, Data Optimisation, Data Splitting, Model Development, Model Training, Model Evaluation, and the necessary dependencies and packages. The end goal being to utilise the model's developed and their results as a 5-day indicator to investors regarding the volatility and change in price of Bitcoin for the specified outlook period.

Data Collection

The first part of creating any type of model is to establish the data that is going to be utilised. Seeing as the task at hand is to perform time series analysis on Bitcoin data, the dataset utilised will need to be temporal, as well as provide metrics on Bitcoin's performance. One of the key decisions that needed to be made was regarding the temporal range between Bitcoin values, having options for minute-by-minute, hourly, and daily. Ultimately the range selected was hourly due to the frequent changes of Bitcoin's value and the window for prediction being 24 hours, over a 120-hour data size.

The dataset is collected from, an open-access API that provides the necessary data (www.cryptodatadownload.com, n.d.). The recorded prices in the dataset were collected from 1 January 2018 till present day, which signifies that the dataset used is expanding hourly. The table below shows the structure of the data that is being received from the API.

Variable Name	Variable Description	Data Type
Date	Date and hour of observation.	Date
Open	Opening price at the given date.	Number
High	High price at the given date.	Number
Low	Low price at the given date.	Number
Close	Close price at the given date.	Number
Volume	Volume of trades at the given date	Number

Figure 14: Table of technical values taken into consideration for model development.

Data Pre-Processing

The initial step to accomplish Data Pre-Processing is to determine key variables for the implementation which are the range of temporal data to be fed into the RNN (SEQ_{LEN}), the future window to predict ($FUTURE_PERIOD$), and the aspect of the data to be predicted ($RATIO_PREDICT$).

$$\begin{aligned} SEQ_{LEN} &= 60 \\ FUTURE_{PERIOD} &= 12 \\ RATIO_{TOPREDICT} &= 'Close' \end{aligned}$$

After establishing these variables, the raw data is then grouped into train(*train_df*), test(*test_df*), and valid (*valid_df*) data frames where the *train_df* holds up to the last 20% of available data, *test_df* holds the last 10% of available data, and *valid_df* holds the last 10 to 20% of available data. These data frames will then be normalised and formatted into *train_*, *test_*, and *valid_*.

Data Normalization

“Data normalization is one of the pre-processing approaches where the data is either scaled and or transformed to make an equal contribution of each feature.” (Singh and Singh, 2019). In our situation the normalization method used was MinMaxScaler which utilises a scaled transformation of the data such that all values fit between the specified minimum and maximum values.

Data Optimization

The data optimization portion only applies to the *train_* values as the other values are for testing purposes and is also a part for the data preparation step. The reason for implementing data optimization is to minimize errors, risks, and allow the model to handle the input data efficiently. There are many options for data optimization, one option considered was the addition of gaussian noise however, upon research in the financial field the standardised choice was Exponential Moving Average (EMA) (Harper, 2019).

EMA is used to smooth noisy data in time series datasets, and is referred to as “smoothing”, it’s an interpretation of weighted averages with the ability to emphasis more recent temporal data. The function implemented to do so is displayed below:

```
length = len(train_)

# Now perform exponential moving average smoothing for smooth curve of data
EMA = 0.0
gamma = 0.1
for ti in range(length):
    EMA = gamma*train_[ti] + (1-gamma)*EMA
    train_[ti] = EMA
```

Figure 15: Function to perform Exponential Moving Average

Data Splitting

The next step is to split the raw data into data suitable for training the model with. To do so a custom splitting method is implemented to take into consideration the variables declared earlier. The function is displayed below:

This function is applied on the *train_*, *test_*, and *valid_* datasets, which then become into *X_train*, *y_train*, *X_test*, *y_test*, *X_valid*, *y_valid*, respectively. Upon splitting the data into the necessary subsets, the pre-processed data can then be visualised by the target values from the data. The data is then formatted in such a way that (*sample_size*, *window_size*, *sample_values*). Where *sample_size* follows the sizes of *train_df*, *test_df*, and *valid_df*. *window_size* is equivalent to the *SEQ_LEN*, and *sample_values* are the parameters specified to be trained on, in this case being Open, Close, High, Low, and Volume (BTC). The *X_train* and *y_train* are then shuffled using a package.

The split of data is graphically represented in the figure below.

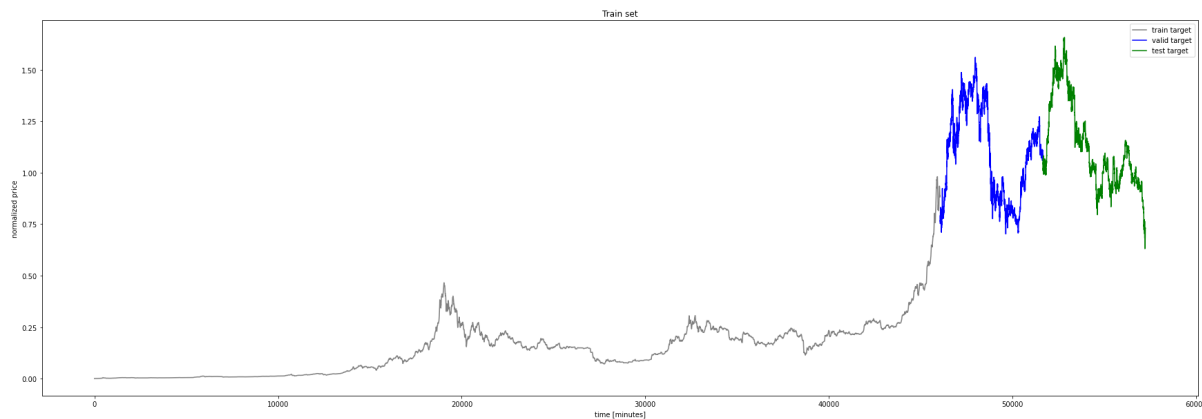


Figure 16: Graphical representation of split up raw data.

Development Environment

As will be discussed later, a key asset to this project is the ability to run the code with just one click, to do so the right development environment needs to be selected. The key criteria are that Python3 needs to be run, as well as any and all dependencies need to be fulfilled, and last but not least enough GPU power is required due to the algorithmic complexity to efficiently train a model. The python notebook options, were between Jupyter notebook and Google Colab, and Colab was selected due to its frequent use within the faculty.

Hardware Specifications

GPU: Nvidia K80 or Nvidia T4 or Nvidia P100

CPU: 2 * 2.30 GHz Haswell vCPU

RAM: 24 GB

Runtime: Unallocated

Subscription: Colab Pro (£8.49)

Crucial Libraries and Dependencies

Data Pre-Processing

```
#Data Processing
import numpy as np
import pandas as pd
from keras import backend as K
from sklearn.preprocessing import MinMaxScaler
from sklearn.utils import shuffle
```

Figure 17: Libraries and packages needed for data pre-processing.

The figure above demonstrates the different packages that were utilised in the data pre-processing aspect of the project. The key libraries here being NumPy, Pandas, and SKLearn. NumPy is a mathematical library available for python that is utilised in the Data Pre-Processing to reshape, organise, and reformat the data (Harris et al., 2020). The Pandas library is a tool that can be utilised for data manipulation, and it provides reliable data structures to house the data for Data Pre-Processing (Mckinney, n.d.). SKLearn is a library provided by Sci-Kit Learn, which houses regression, clustering, normalising, and other mathematical functions required for machine learning (Pedregosa et al., 2011).

Model Development

```
#Model Development
import numpy as np
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import *
from tensorflow.keras.initializers import *
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.python.keras.layers import Layer
```

Figure 18: Libraries and packages needed for model development.

The figure above represents the different packages necessary for proceeding with model development. The key library being TensorFlow which is an open-source library developed by Google for machine learning and artificial intelligence (Abadi et al., n.d.). Keras, which is another library that interacts with TensorFlow that allow for easy instantiation of the various aspects of a model. More details about the specifics within each import are discussed in each model's approach.

Section 4: Implementation

The following sections cover the configuration and architectural decisions of 4 different models and the chosen configuration for the models. The models being LSTM, Bi-LSTM, GRU, and an Encoder Transformer. This section will also discuss the various implementations that were attempted using these specified models, as well as discuss the implementations that didn't achieve or perform as desired.

Constants and Parameters

This portion covers the various constants and parameters that were utilised in the implementation of the models. These parameters are kept constant and were done so for the purpose of identifying the performance of the individual layers as opposed to which model could be tuned to the optimum performance, so long as the objective was met. The chosen library as mentioned earlier is Keras, within Keras contains many libraries that allow for the creation of models, call-backs, and other parameters for tuning a machine learning neural network. The following section discusses the parameters utilised for the creation of the models.

Activation Function

The Keras layer contains an activation function parameter that allows for the activation function to be specified by the user, the considered options as well as a breakdown of the activation functions are displayed on the table below:

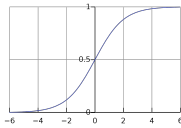
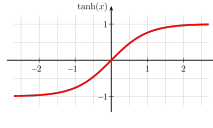
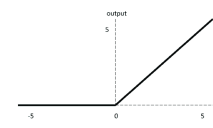
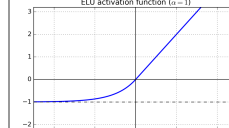
	Activation Function (Chigozie et al., 2018)			
	<u>Sigmoid</u>	<u>Tanh</u>	<u>ReLu</u>	<u>ELU</u>
Range	0 to 1	-1 to 1	0 to ∞	$-\infty$ to ∞
Functional Representation	$\sigma(x) = \frac{1}{1+e^{-x}}$	$\frac{2}{1+e^{-2x}} - 1$	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Graphical Representation				

Figure 19: Breakdown of different activation functions and their parameters.

The chosen option for implementation being the ReLu, referred to as rectifier and standing for Rectified Linear unit. The ReLu as an activation function is the most popular activation function utilised in deep neural network (Agarap, Abien Fred, 2018). The advantages of the ReLu function are its ability to handle the problem of vanishing gradients, efficient computation, sparse activation. The key advantage of a ReLu activation function is its ability to tackle the problem of vanishing gradients, which other activation functions like sigmoid and hyperbolic tangent struggle to do. The risks of this implementation is that the function's not zero-centred nor is it bounded, another issue is that of a dying ReLu. The neuron's activated by the function can potentially be pushed into inactive states

reducing them unresponsive to any inputs, this is an issue like that of vanishing gradients. (Agarap, Abien Fred, 2018) However, by adjusting the learning rate and tuning the optimizer it's possible to rectify some of these issues.

Optimizers

Optimizers in deep learning are algorithms utilized to find the ideal parameters and weights to minimize errors when mapping inputs to outputs. Some studies indicate that optimizer enhancement is one of the, if not the, greatest factor in the tuning protocol of a deep learning pipeline (Choi et al., n.d.). One key finding was that popular adaptive gradient methods never underperform momentum or gradient descent, when properly tuned. Keeping this in mind, the considered options for compatible adaptive gradient methods were RMSProp or Adam.

The Adam optimizer employs a hybrid of two gradient descent techniques: Momentum: This approach is used to speed up the gradient descent algorithm by considering the gradients “exponentially weighted average.” The benefits of an Adam optimizer are, the ability to add parameters to tune the optimizer, minimal computational power required, and typically require minimal tuning (Kingma and Ba, 2014). One key parameter available for tuning is the learning rate.

Learning Rate

The learning rate is a parameter in an optimization method that defines the step size at each iteration while advancing toward the loss function's minimum (Wu et al., 2019). The default learning rate is set to 10^{-3} , however this parameter was tuned and adjusted for each respective model. The range for the learning rate in all implementations ranges from 1×10^{-5} to 5×10^{-3} . The means for deciding this value followed a naïve approach wherein a high learning rate of 10^{-1} is initially set, and then is exponentially lowered until a suitable objective is met.

Loss Metric or Cost Function

The cost function is utilised for backpropagation and is a metrics of a neural network's performance. Keras's library provides multiple options for the cost function, due to unique scenarios requiring specific functions to accurately train the model. Seeing as the implementation aims to perform time series regression, the cost function needs to be tailored for regression. Research pointed towards Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Percentage (MAPE), Huber loss, and pinball loss (Chigozie et al., 2018). The table below identifies the popular metrics based on appearances in academic surveys.

Metrics	C&A, 1982	M&K, 1995	M et al, 2006	F&G, 2007
Mean square error (MSE) or Root MSE (RMSE)	34	10	6	9
Mean absolute error (MAE)	18	25	20	36
Mean absolute percentage error (MAPE)	15	52	45	44

Note:

C&A, 1982: (Carbone and Armstrong, 1982); M&K, 1995: (Mentzer and Kahn, 1995); M et al, 2006: (McCarthy et al., 2006); F&G, 2007: (Fildes and Goodwin, 2007).

Figure 20: Survey of loss metrics and their usages in academic papers. (Botchkarev, n.d.)

The chosen metric for evaluating loss is MSE due to its widespread applications in regression (Hyndman and Koehler, 2006), and due to its ability to measure the quality of an estimator. The function is defined mathematically below:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Given that y_i corresponds to the scalar value of the i^{th} input, \hat{y}_i equates to the scalar value of the predicted output, and n is the number of samples in a set. MSE simply put is the average squared difference between a model's predictions outputs when in comparison to the actual output value. In MSE the value is computed using the Euclidean distance between the y_i and \hat{y}_i . The MSE's value will always be positive and approaches zero if the model's accuracy improves, an MSE close to zero is more desirable as it means the variance between predicted and real values are low.

Alternative Metrics for Empirical Risk Minimization

As mentioned earlier, there are many options when considering the cost function of a model, Keras's library provided the functionality to apply additional metrics onto the model to identify and track performance. The additional metrics utilised are: Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Root Mean Squared Error (RMSE).

Mean Absolute Error (MAE)

The statistical definition of MAE is a measure of errors between paired observations expressing the same phenomenon (Willmott and Matsuura, 2005). The paired observations in this case being the predicted values and the true values.

$$MAE = \frac{\sum_{i=1}^n |\hat{y}_i - y_i|}{n}$$

Given that y_i corresponds to the scalar value of the i^{th} input, \hat{y}_i equates to the scalar value of the predicted output, and n is the number of samples in a set.

Some studies indicate that MAE is a better metric for model evaluation due to its unambiguous nature which extends its hand in ensuring that MAE results are a more natural way of evaluating error (Willmott and Matsuura, 2005). There exists an inherent difficulty with MAE due to the output value's scaling being data value dependant, resulting in a value that is difficult to propagate meaning from (Hyndman and Koehler, 2006).

Mean Absolute Percentage Error (MAPE)

MAPE is defined as a percentile metric for error between paired observations in relation to the expected value. The MAPE is defined mathematically below, given that y_i corresponds to the scalar value of the i th input, \hat{y}_i equates to the scalar value of the predicted output, and n is the number of samples in a set.

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

One paper discussing minimizing empirical risks evaluated that the MAPE can be used as a loss function in regression analysis both practically and theoretically, due to the existence of an optimal model and the consistency of empirical risk minimization can be demonstrated. The table below indicates various MAPE values and the interpretations that can be drawn from the data.

MAPE	Interpretation
< 10%	Highly Accurate Forecasting
10% < 20%	Good Forecasting
20% < 50%	Reasonable Forecasting
50% <	Inaccurate Forecasting

Figure 21: MAPE breakdown with corresponding ability to forecast.(Lewis, 1982, p.42)

Root Mean Squared Error (RMSE)

RMSE is an extension of MSE as the RMSE is the square root of the MSE value. Like MSE, the RMSE value is utilised for comparing paired observations from estimators or forecasters. (Chai and Draxler, 2014) The RMSE function can be mathematically represented as shown below:

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}}$$

Given that y_i corresponds to the scalar value of the i th input, \hat{y}_i equates to the scalar value of the predicted output, and n is the number of samples in a set.

The benefit of RMSE as opposed to MSE is that the values are always non-negative and indicates that a value closer to zero indicates less deviance from the predictions and real values. An inherent flaw

with RMSE is its sensitivity to outliers in the data (common in Bitcoin), this is due to an increase in difference $y_i - \hat{y}_i$ resulting in a proportionally squared increase in RMSE value.

Call-backs

Another key aspect of developing and running deep learning neural network models with Keras is the ability to implement a concept known as Call-backs. “A call-back is an object that can perform actions at various stages of training (e.g., at the start or end of an epoch, before or after a single batch, etc).” (Keras Team, n.d.). The call-backs API can handle a list of parameters, however the key aspect for this implementation was Early Stopping.

Early Stopping was implemented to maximise training time and ensure that the model doesn’t waste time training when the measured parameter isn’t indicative to an improving model. The code exactly monitors the validation loss while fitting, allows the model 12 epochs of minimal improvement before cancelling, and upon cancelling restores the best weights of the model. The code snippet below visualises the parameters set:

```
callback = EarlyStopping(monitor='val_loss',  
                        patience=12,  
                        restore_best_weights=True)
```

Figure 22: Code written for call-back declaration.

Epochs and Batch Size

The epochs and batch size are two key hyperparameters in training a model. The epoch is defined as the number of times the entire dataset will be passed through the model (Brownlee, 2018). The value chosen for the `EPOCHS` is slightly irrelevant due to the presence of early stopping which stops the training process earlier, if possible, hence why `EPOCHS` is a relatively high number of iterations at 100.

```
EPOCHS = 100  
BATCH_SIZE = 256
```

Figure 23: Epoch and batch_size breakdown.

The batch size is the number of samples the model iterates through before updating the internal weights and carrying on (Brownlee, 2018). The batch size was determined through naïve investigation, by initially instantiating a size of 32, and progressively working up by powers of 2 to 256. This batch size allows for suitable training times, model dependant, and allows for resources to be allocated affectively.

Model Implementation and Training

Now that the parameters and various configurations are setup and in place, the next plan of action is to architect and implement the neural networks themselves. This section is structured with the

following four sections covering LSTM, GRU, Bi-LSTM, and a LSTM-Transformer model implementation. Be aware that the units within the hidden layer's RNNs are kept constant.

Note that the entire notebook is available to be seen in the [appendix](#).

Input and Data Shape

Training Data

The training data contains up to the last 20% of available data.

```
X train shape (46222, 72, 5)
y train shape (46222,)
```

Figure 24: Training data, x and y shape.

The *X train* dataset contains 46,222 date samples, with a 72-hour temporal period, and 5 parameters. The *y train* contains the same length of sample, but only the labelled data, the normalized Bitcoin prices.

Validation Data

The validation data contains from the last 20% up to the latest 10 % of available data.

```
X valid shape (5705, 72, 5)
y valid shape (5705,)
```

Figure 25: Validation data, x and y shape.

The *X valid* dataset contains 5,705 samples, with the same period and parameters. The *y valid* contains the same length of sample, but only the labels.

Testing Data

The testing data contains the latest 10 % of available data.

```
X test shape (5705, 72, 5)
y test shape (5705,)
```

Figure 26: Testing data, x and y shape.

The *X test* dataset contains 5,705 samples, with the same period and parameters. The *y test* contains the same length of sample, but only the labelled data.

Refer to earlier section for data visualization.

Additional Layers

Whilst it may be possible to build a model utilising purely the RNN layers, there are other model architectural decisions that can be made to standardize and overall improve model performance. The two key layers that are utilised are the Dense and Dropout layers.

Dense Layer

The dense layer is defined as a layer that is deeply connected with its previous layer – signifying that each neuron in the dense layer is connected to each neuron in the previous (Huang, Liu and Weinberger, Kilian Q, 2016). Within the neurons a mathematical function is applied to generate the output, using Matrix-Vector multiplication. The mathematical representation is shown below:

$$Ax = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$
$$= \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \end{bmatrix}.$$

Figure 27: Matrix vector multiplication diagram.

Given that Ax is the output of the multiplication of $A * x$, A is a matrix of dimensions $(M \times N)$, and x is another matrix with $(1 \times N)$ dimensions. It is understood that the output, Ax will always be an N dimensional vector.

Keras provides a Dense layer that allows for easy implementation of the functionality, the options for parameters are plenty, and are shown in the code below, however, there is only one required parameter, `units` (Team, n.d.). This parameter is equivalent to the N dimensions, as mentioned the $Ratio_{to_{predict}}$ is 'close' and of only 1 dimension, hence our Dense layer has only 1 unit.

Default Implementation	This Implementation
------------------------	---------------------

<pre>tf.keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer="glorot_uniform", bias_initializer="zeros", kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None, **kwargs)</pre>	
--	--

	<pre>tf.keras.layers.Dense(units=1)</pre>
--	---

Dropout Layer

The Dropout layer also known as the Dilution layer, is defined by its implementation of regularization techniques to minimize overfitting by avoiding complicated co-adaptations from the training data. The Dropout layer essentially disregards and drops weights out from the model by implementing a regularization algorithm shown in the figure below. The main reason for implementing a dropout layer is to prevent the common problem of overfitting a training dataset on a model (Srivastava et al., 2014).

Keras's Dropout layer contains three parameters: *rate*, *noise_shape*, and *seed*. The only mandatory parameter being the rate, which ranges between 0 and 1. The layer itself will randomly edit input values to 0 with a frequency of the rate specified, this directly aids in the prevention of overfitting. The input values that aren't randomly dropped to 0 are then scaled up by $\frac{1}{1-rate}$ such that $\sum_{i=1}^n Previous_{outputs}(x_i) = \sum_{i=1}^n Dropout(x_i)$. Given that n is the number of inputs, *Previous_ouputs* is the output values of the previous layer to the Dropout, and *Droupout* is the dropout layer applied on the inputs.

LSTM Approach

There are many ways to structure an LSTM model, single-layered, stacked, not to mention the parameters within the LSTM layer. Ultimately a bi-layered LSTM (*lstm_model*) was implemented with 60 units in the hidden layer for the first LSTM, and 120 units in the hidden layer of the second LSTM. The image below visualises a summary of the model:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 72, 60)	15840
dropout (Dropout)	(None, 72, 60)	0
lstm_1 (LSTM)	(None, 120)	86880
dropout_1 (Dropout)	(None, 120)	0
dense (Dense)	(None, 1)	121
Total params: 102,841		
Trainable params: 102,841		
Non-trainable params: 0		

Figure 28: LSTM model summary diagram.

Training the Model

Upon building the model, *lstm_model* needs to be fitted on the training data, this is accomplished using an inbuilt function that Keras provides to fit the model. The exact lines of code for fitting the model are shown below:

```
lstm_history = lstm_model.fit(x=X_train,
                             y=y_train,
                             batch_size=BATCH_SIZE,
                             epochs=EPOCHS,
                             validation_data=(X_valid, y_valid),
                             callbacks=[callback])
```

Figure 29: LSTM model fitment onto training data.

The *fit()* method takes in key parameters to successfully fit a model to a given dataset. The *x* equates to the input training data, *y* equates to the labelled training data, *batch_size* is the specified batch size, *epochs* is the specified number of epochs, *validation_data* is a tuple of the input validation data (*X_valid*), and labelled validation data (*y_valid*), and lastly *callbacks* equates the previously mentioned call-backs.

Upon running this line of code, the entire model starts fit to the data, and will in parallel display the ongoing loss performance, as well as the alternative metrics specified. The figure below is a sample from the LSTM model.

```
Epoch 1/60
181/181 [=====] - 49s 245ms/step - loss: 0.0073 -
mean_absolute_error: 0.0467 - mean_absolute_percentage_error: 195.6753 -
root_mean_squared_error: 0.0852 - val_loss: 0.0574 - val_mean_absolute_error:
0.1803 - val_mean_absolute_percentage_error: 14.6120 -
val_root_mean_squared_error: 0.2397
```

Figure 30: LSTM model output whilst training.

Upon completion, the *lstm_history*, contains a Keras object known as History, which contains the historical data for the training of the *lstm_model*, this also means that *lstm_model*'s layers have all been trained and the necessary weights applied to it from its fitment to the training data. This means two things, *lstm_history* contains the necessary data to analyse if the model has successfully comprehended the training data, and more importantly, the model should be primed and ready to predict the values for the testing data.

Predicting Using the Model

With the model updated with the weights from the fitment, the next step is to predict using the model. To do so, Keras provides an inbuilt function called *predict()*, which returns output predictions for a given input data sample.

```
pred_lstm = lstm_model.predict(X_test)
```

Figure 31: Code to predict using LSTM model.

The newly created variable, *pred_lstm*, contains the predicted values for the corresponding temporal sequences available in *X_test*. The output is of array type with (5702,1) shape. This output can be processed and needs to be formatted to reveal the results of the prediction. The results will be discussed in more detail in the Results section.

GRU Approach

The architectural decisions taken between the approaches are constant, and hence the GRU model is bilayered as well (*gru_model*). There are 60 units in the hidden layer for the first GRU, and 120 units in the hidden layer of the second GRU. The image below visualises a summary of the model:

Model: "sequential_2"

Layer (type)	Output Shape	Param #
gru (GRU)	(None, 72, 60)	12060
dropout_2 (Dropout)	(None, 72, 60)	0
gru_1 (GRU)	(None, 120)	65520
dropout_3 (Dropout)	(None, 120)	0
dense_1 (Dense)	(None, 1)	121

=====
Total params: 77,701
Trainable params: 77,701
Non-trainable params: 0
=====

Figure 32: GRU model summary diagram.

Training the Model

At this stage the model has already been instantiated. *gru_model* needs to be fitted on the training data, using the same inbuilt Keras function to fit the model. The exact lines of code for fitting the model are shown below:

```
gru_history = gru_model.fit(x=X_train,
                           y=y_train,
                           batch_size=128,
                           epochs=EPOCHS,
                           validation_data=(X_valid, y_valid),
                           callbacks=[callback])
```

Figure 33: GRU model fitment to training data code.

The figure below is a sample output when fitting the GRU model on the training data.

```
Epoch 1/60
361/361 [=====] - 144s 391ms/step - loss: 0.0012 -
mean_absolute_error: 0.0191 - mean_absolute_percentage_error: 74.0005 -
root_mean_squared_error: 0.0351 - val_loss: 0.0017 - val_mean_absolute_error:
0.0303 - val_mean_absolute_percentage_error: 2.8206 -
val_root_mean_squared_error: 0.0410
```

Figure 34: GRU output whilst fitting data.

gru_model is now trained and the necessary weights applied to it from its fitment to the training data. *gru_history*, contains the History object for the *gru_model*, the results are displayed in the [appendix](#). The model should now be primed and ready to predict the values for the testing data.

Predicting Using the Model

The next step is to predict using the `predict()` function available from Keras, which returns output predictions for a given input data sample.

```
pred_gru = gru_model.predict(X_test)
```

Figure 35: Code to predict using the GRU model.

The newly created variable, `pred_gru`, contains the predicted values for the corresponding temporal sequences available in `X_test`. The output is of array type with (5702,1) shape. This output can be processed and needs to be formatted to reveal the results of the prediction. The results will be discussed in more detail in the Results section.

Bi-LSTM Approach

The architectural decisions taken between the approaches are constant, and hence the Bi-LSTM model is bi-layered as well (`bi_model`). There are 60 units in the first hidden layer, and 120 units in the second hidden layer. The image below visualises a summary of the model:

Model: "sequential_3"

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)	(None, 72, 120)	31680
dropout_4 (Dropout)	(None, 72, 120)	0
bidirectional_1 (Bidirectional)	(None, 240)	231360
dropout_5 (Dropout)	(None, 240)	0
dense_2 (Dense)	(None, 1)	241
Total params: 263,281		
Trainable params: 263,281		
Non-trainable params: 0		

Figure 36: Bi-LSTM model summary diagram.

Training the Model

At this stage the model has already been instantiated. `bi_model` needs to be fitted on the training data, using the same inbuilt Keras function to fit the model. The exact lines of code for fitting the model are shown below:

```
bi_history = bi_model.fit(x=X_train,
                        y=y_train,
                        batch_size= BATCH_SIZE,
                        epochs=EPOCHS,
                        validation_data=(X_valid, y_valid),
                        callbacks=[callback])
```

Figure 37: Bi-LSTM fitment on to X_{train} data.

The figure below is a sample output when fitting the Bi-directional LSTM model on the training data.

```
Epoch 1/60
181/181 [=====] - 114s 603ms/step - loss: 0.0017 -
mean_absolute_error: 0.0203 - mean_absolute_percentage_error: 84.2943 -
root_mean_squared_error: 0.0416 - val_loss: 0.0086 - val_mean_absolute_error:
0.0740 - val_mean_absolute_percentage_error: 6.6226 -
val_root_mean_squared_error: 0.0930
```

Figure 38: Bi-LSTM model output whilst fitting data.

bi_model is now trained and the necessary weights applied to it from its fitment to the training data. *bi_history*, contains the History object for the *bi_model*, the results are displayed in the [appendix](#). The model should now be primed and ready to predict the values for the testing data.

Predicting Using the Model

The next step is to predict using the *predict()* function available from Keras, which returns output predictions for a given input data sample.

```
pred_bi = bi_model.predict(X_test)
```

Figure 39: Code to predict using the Bi-LSTM model.

The newly created variable, *pred_bi*, contains the predicted values for the corresponding temporal sequences available in *X_test*. The output is of array type with (5702,1) shape. This output can be processed and needs to be formatted to reveal the results of the prediction. The results will be discussed in more detail in the Results section.

RNN-LSTM Transformer Approach

Transformer's are a new technology for deep learning neural networks, and the approach implemented extends the knowledge put forth from a Kaggle competition which implemented LSTMs in conjunction with transformers for classification. The encoder-decoder architecture is an extension of the model put forth in the repository and paper "Attention Is All You Need" (Vaswani et al., 2017), the key difference being adapting the activation functions, the cost function, the optimizer, and the LSTM model initially proposed. The cost function was altered to utilise "*mse*" as it's loss metric, the optimizer utilised and Adam with a custom learning rate, and the LSTM model proposed followed the Bi-Directional LSTM architecture. The full model summary is provided in the [appendix](#) due to the length.

Training the Model

At this stage the transformer model, *trans_model*, has already been instantiated. *trans_model* needs to follow the same steps as the other models and hence need to be fitted on the training data, using the same inbuilt Keras function to fit the model. The exact lines of code for fitting the model are shown below:

```
trans_history = trans_model.fit(x=X_train,
                                y=y_train,
                                batch_size= BATCH_SIZE,
                                epochs=EPOCHS,
                                validation_data=(X_valid, y_valid),
                                callbacks=[callback])
```

Figure 40: Transformer fitment onto training data.

The figure below is a sample output when fitting the Bi-directional LSTM Transformer model on the training data.

```
Epoch 1/60
181/181 [=====] - 119s 624ms/step - loss: 0.0079 -
mean_absolute_error: 0.0358 - mean_absolute_percentage_error: 74.5950 -
root_mean_squared_error: 0.0890 - val_loss: 0.0709 - val_mean_absolute_error:
0.2102 - val_mean_absolute_percentage_error: 17.0744 -
val_root_mean_squared_error: 0.2663
```

Figure 41: Transformer fitment output.

trans_model is now trained and the necessary weights applied to it from its fitment to the training data. *trans_history*, contains the History object for the *trans_model*, the results are displayed in the [appendix](#). The model should now be primed and ready to predict the values for the testing data.

Predicting Using the Model

The next step is to predict using the *predict()* function available from Keras, which returns output predictions for a given input data sample.

```
pred_trans = trans_model.predict(X_test)
```

Figure 42: Code to use the transformer to predict on X_test.

The newly created variable, *pred_bi*, contains the predicted values for the corresponding temporal sequences available in *X_test*. The output is like that of all other outputs, being an array of (5702,1) dimensions. This output can be processed and needs to be formatted to reveal the results of the prediction. The results will be discussed in more detail in the Results section.

Section 5: Results

Evaluation Metric R2 Score

R2 score, commonly known as the coefficient of determination, is the fraction of the dependent variable's variance that can be predicted by the independent variable. In finance, it's determined as the performance of an asset in relation to a given benchmark. The function to calculate the R2 score is:

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}}$$

Given that SS_{RES} is the sum of squared errors in the prediction, and SS_{TOT} is the sum of total squared errors. The main usage of an R2 score in this implementation is to use said value and understand if the coefficient is satisfactory to utilise for predictions. For financial analysis a desirable R2 score is 0.7, which indicates a high correlation (Fernando, 2021).

Thankfully, SKLearn, provides a metrics package to easily evaluate the performances, using this methodology is how the R2 score, final MAPE score, as well as a variance score are visualised.

LSTM Approach

Below is the graphical representation as well as the performance metrics for the LSTM model.

Graphical Representation

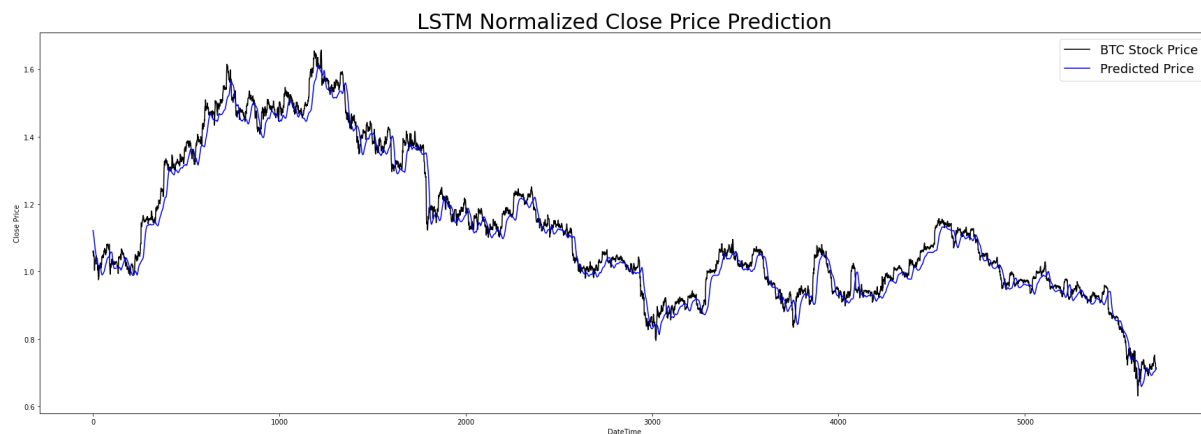


Figure 43: Normalised graphical represenetation of LSTM predictions.

Metric Representation

```
Explained Variance Score : 0.9764412386431712
Mean Absolute Percentage Error : 0.024583315162298402
R2 Score : 0.971785004401008
```

GRU Approach

Below is the graphical representation as well as the performance metrics for the GRU model.

Graphical Representation

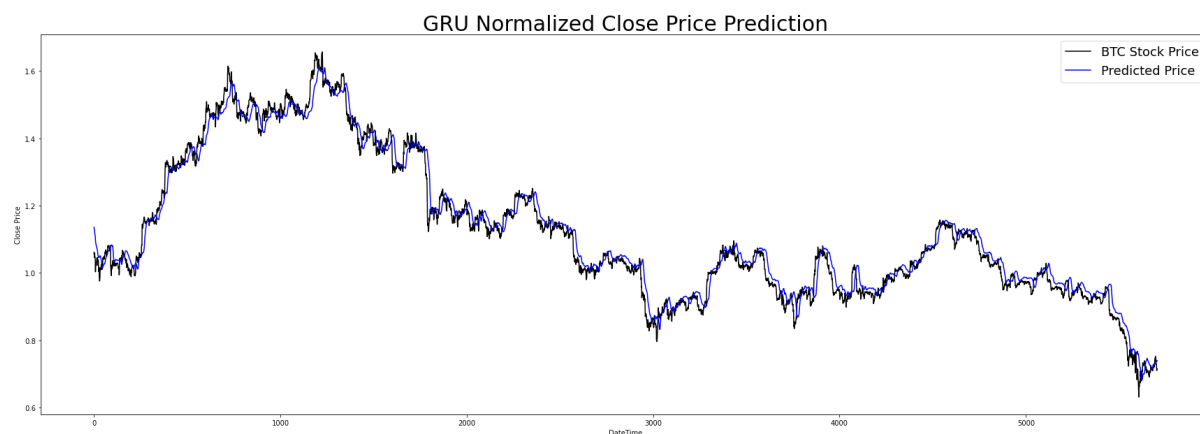


Figure 44: GRU graphical represenetaion of predicted outputs.

Metric Representation

```
Explained Variance Score : 0.9783305226606047  
Mean Absolute Percentage Error : 0.0220535538576336  
R2 Score : 0.9771994979226912
```

Bi-LSTM Approach

Below is the graphical representation as well as the performance metrics for the Bi-LSTM model.

Graphical Representation

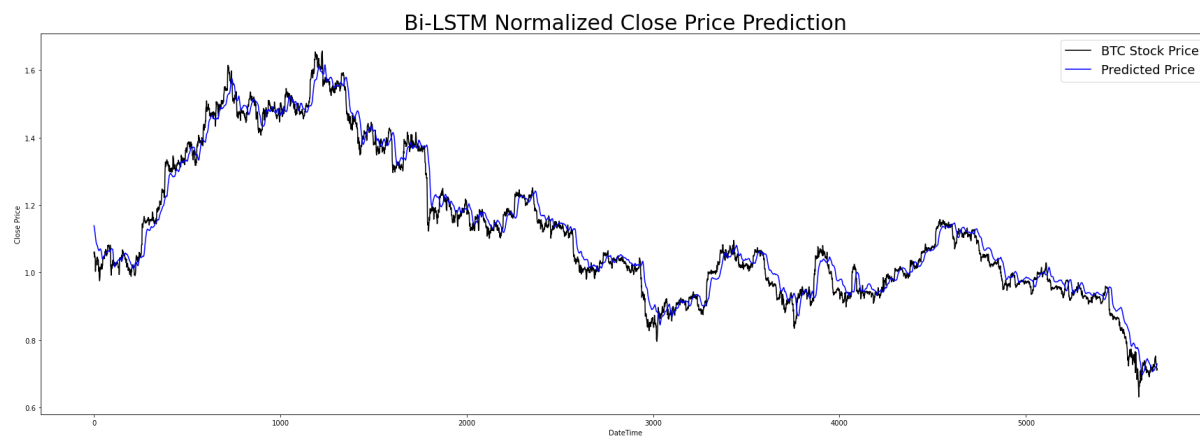


Figure 45: Graphical representation of Bi-LSTM prediction outputs.

Metric Representation

```
Explained Variance Score : 0.9756784974506815  
Mean Absolute Percentage Error : 0.02366633701625547  
R2 Score : 0.9745046810905619
```

RNN-Transformer Approach

Graphical Representation

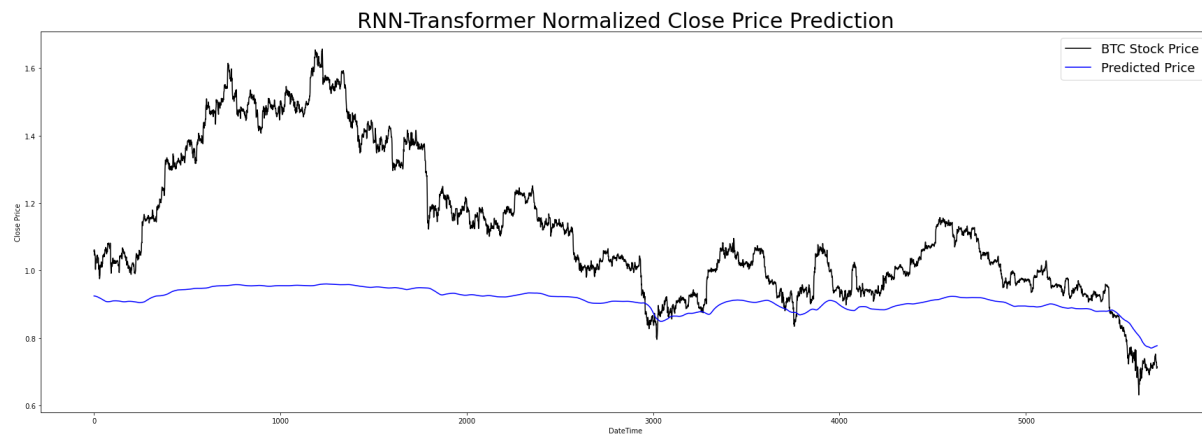


Figure 46: Transformer predicted outputs graphical representation.

Metric Representation

Explained Variance Score : 0.25522234640864305
Mean Absolute Percentage Error : 0.17195781254853773
R2 Score : -0.7017785723228909

Section 6: Evaluation and Objective Analysis

The performances of the four various models are tabulated in the table below, and the graphical representations are demonstrated in the figure below as well. The MAPE and the R2 score of the GRU is 2.2053% and 0.9772, surpassing all other instantiated models, making the GRU implementation the implementation to suggest. The Bi-Directional LSTM came a close second with an R2 score of 0.9745 and a 2.3664% MAPE, this beat the LSTM model which had an R2 score of 0.97179 and an MAPE value of 2.4583%.

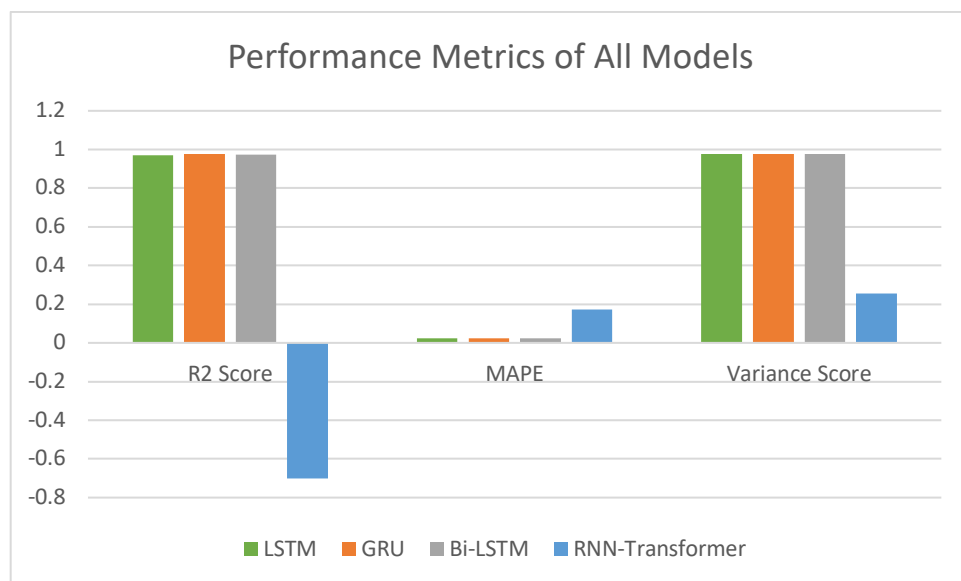


Figure 47: Graphical representation of analysis metrics on predicted outputs of all models.

The clear outlier from the four models, was the RNN-Transformer, which had a negative R2 score of -0.70178 and an MAPE of 17.196%. It's clear to understand that the Transformer model, either wasn't implemented correctly, wasn't tuned correctly, or it doesn't work for this type of problem. This aspect is a feature to think about for the future.

Name	R2 Score	MAPE	Variance Score
LSTM	0.97179	2.4583%	0.97644
GRU	0.97720	2.2053%	0.97833
Bi-LSTM	0.97450	2.3664%	0.97568
RNN-Transformer	-0.70178	17.196%	0.25522

Figure 48: Data representation of performance metrics for all implementations.

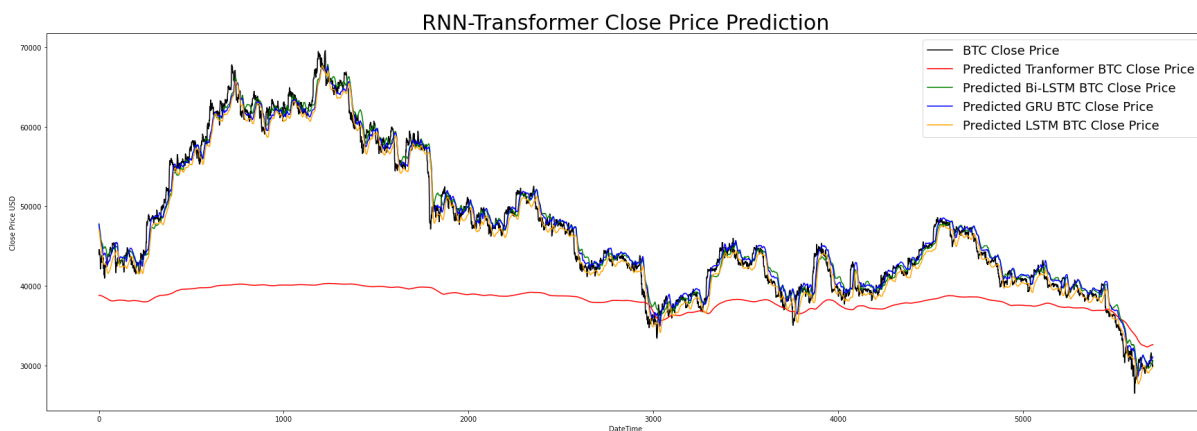


Figure 49: Graphical representation of all outputs on actual Bitcoin close prices.

Objectives

Develop a solid base understanding of the core topics to properly evaluate and implement deep learning neural network models.

- The key objective of any dissertation is to thoroughly comprehend and understand the topic of discussion. To do so for this project, countless articles, papers, and journals were scoured, from them knowledge extracted and implemented. One difficulty faced was the presence of inaccurate and irregular information, hence why multiple sources needed to be covered for any single topic.

Develop a model(s) that adapts daily, that keeps expanding and learning based on constant changes in the market.

- The way the notebook is designed is to dynamically get all the latest hourly Bitcoin data fixed from 2016. When the project initially started the dataset size was about 10,000 samples smaller than it is currently, the model can also be adapted to adjust for these new weights. The weightings can either be recreated, updated, or ignored depending on the training parameters set.

Implement at least three instances of Neural Network models to accurately forecast the price of three different cryptocurrencies. Models will be parameterised to ideally achieve an R2 score of 85% or greater – 15% greater than industry standard for asset indicators (Fernando, 2021).

- R2 needing to be greater than 85%, and a forecast longer than 6 hours, an analysis as to whether it's been achieved or not needs to take place. All four models were predicting for a future period of 12 hours; hence the second objective is fulfilled to its entirety. However, seeing as the metric for evaluation is the 85% R2 score, the RNN-Transformer model is the only model to fail its objective, and looking at the graphical representation it's fair to see why. The three other RNN models – LSTM, GRU, and Bi-LSTM, all had R2 scores of at least 12% greater than the evaluation metric.

Create functionality that can visualise at least 10 hours in advance for a positive or negative gradient (will the price go up or down) – this can help decided if a cryptocurrency should be sold, bought, or held.

- This objective is fulfilled by the decelerations specified in the code for the future window to predict, by declaring a 10-hour window, and then using the model to predict on a given temporal sequence will result in predictions of that length as well.

Section 7: Review of LSEP Issues

This section covers the LSEP Issues with this dissertation and discusses in detail with reference to the British Computer Society Code of Conducts and Good Practices (www.bcs.org, n.d.), the necessary legal institutions like IEEE's Code of Ethics, and the University of Surrey's Governance and Ethics (research.surrey.ac.uk, n.d.).

Legal Issues

The common legal issues that are discussed in the field of computer science are to do with Contract law, Intellectual Property law, Data Protection law, Computer Misuse Act (Gov.uk, 1990), and computer evidence. Note that only Intellectual Property law, and Computer Misuse Act, will be discussed seeing as there is no implementation of contracts nor is there personal data being collected.

Intellectual property (IP) refers to a person's or company's innovations and creative efforts, and IP Law refers to the administration and facilitation of these efforts under a legal setting (World Intellectual Property Organization, 2016). In the entire project, the only potential infringement of IP law would relate to the implementation of the RNN-transformer model which utilised pre-existing works to reach its final design. However, this is not an issue seeing as the utilised as the open-source code has an MIT License, which allows all individuals access for both private and commercial reasons but doesn't provide liability (Opensource.org, 2019).

Criminal Misuse Act of 1990 (CMA) is the law defining how individuals can properly access data on a machine, as well as defining the law if and when any unauthorised access to data and the practise of making changes to stored information are done without the owner's authorization. The data utilised in this implementation is free and open-source, most importantly the providers, CryptoDataDownload, openly discuss the free use of the software – as well as its limitations (www.cryptodatadownload.com, n.d.). Seeing as the author has permitted use of the data, and the data is not being modified directly to the author, this dissertation doesn't infringe the CMA.

Social Issues

The social issues that arise from this implementation are those affecting the end user, the one's either running the notebook, or using the information provided from the predictions. The issue arises primarily if and when the users choose to act financially on the sole discretion of the prediction. However, for this reason similar platforms like eToro, openly advocate that any information provided

is merely an indicator and should not be taken as the whole truth (Rudolf, Ajour El Zein and Lansdowne, 2021). For this reason, a disclaimer is put that covers to a limited extent of the social issues discussed and most importantly declares that the use of the software is solely as an indicator and no liability is held for any decisions made using the notebook.

Ethical Issues

These ethical issues section focuses on the issues specified in the Governance and Ethics form for the University of Surrey, any other ethical issues with this implementation extend that of the social issues. The primary concern when regards to governance and ethics is the SAGE form that's required to understand the ethicality, as well as feasibility of the project (research.surrey.ac.uk, n.d.). The completed SAGE form is in the [appendix](#).

Professional Issues

The professional issues pertaining to this project are done in referral to the Code of Conduct provided by the British Computer Society (BCS), as mentioned in their document there are four key principles. You make IT for everyone. Show what you know, learn what you don't. Respect the organization or individual you work for. Keep IT Real, Keep IT Professional, and Pass IT On. At a high-level, it's fair to assume that the BCS Code of Conduct hasn't been violated but has been kept as a cornerstone to this project. The overarching theme from the Code of Conduct is to make more information public to better educate and inform the general population (www.bcs.org, n.d.). Seeing as the implementations discussed in this paper is available on a Google CoLab notebook with proper formatting, and only requires one click to receive the predictions. For this exact reason, it's fair to say that the implementation is highly public as well as educational.

Section 8: Conclusion

This paper covered four different deep learning neural network techniques, it covered various iterations of RNNs, tuned the numerous hyper parameters, and discussed in detail how to construct these AI methodologies to predict Bitcoin's value. The different models implemented are Long-Short Term Memory, Gated Recurrent Units, Bi-Directional LSTMs, and RNNs in conjunction with an encoder Transformer. All these models were fitted on an ever-increasing dataset, 55 thousand samples and ever-increasing at the time of running, and all models made predictions with a future window of 12 hours. The performances of these individual models and the accuracy and depth as to what knowledge was extracted can be comprehended in detail in the section above. The results indicate that from the four models, the GRU is the best with the lowest MAPE of 2.2053%, and the highest R2 score of 0.97745, the LSTM as well as Bi-LSTM both represent excellent options for time series forecasting. The only model to fail to meet the set objectives was the RNN-Transformer, the reasons for this could be overfitting of the model, too many units within the model, and because the initial architecture was tailored for time-series classification and not regression. Keeping in mind that three of the four models passed exceptionally, the opportunity for future works needs to be evaluated.

Future Works

The scope for future works stemming from this implementation can be broken down into two avenues; one is improving the models themselves or developing a platform so the models can be utilised. are very broad due to the many variables that can influence a time series forecasting, them being the dataset, the factors of influence, the model architectures, and many more.

Developing a Platform for the Models

To go about developing a platform for the model wouldn't be too complicated from the given implementation, the key technical hurdle here would be generating the APIs to use the model in a web application. For the actual implementation the usage of FLASK in conjunction with python to develop the interface between the model and UI. The technology stack would be ideally implemented using cross platform web applications, like Flutter from Google, or React Native from Facebook. It's also possible to utilise a more traditional stack like Ruby on Rails, or a MERN stack for traditional web applications.

Improving the Models

To improve the models there are multiple paths that can be undertaken, new architectures could be investigated, more complex optimisations of hyperparameters could be done, as well as considering more determinants of bitcoin.

The current implementation took technical data (Open, High, Close, Low, Volume), and conducted analysis under the assumption that these were the sole factors of influence for Bitcoin. Deeper studies into the determinants of price for Bitcoin indicate a strong correlation between Google

searches, and value, as well as Twitter's sentiment analysis on posts affiliated to Bitcoin to affect the value. Some studies indicate a 91% correlation between Google searches and Bitcoin prices, hence, using these factors and reapplying a new model architecture could result in further improvement of Bitcoin value forecasting. (Arratia and López-Barrantes, 2021).

One potential new architecture that could be implemented could be the implementation of Dynamic Time Warping (DTW), which is used under the guise of Time Series Classification. DTW is a form of time series analysis, which converts time-based values into sequences to measure similarities and differences between sequences (Hsu et al., 2015). The aim of implementing DTW and potentially even clustering algorithms would be to perform classification on the predicted prices to indicate when Bitcoin prices are good for either holding, selling, or buying. The training data would be sequences of historical Bitcoin values.

Section 9: Table of Figures

Figure 1: ANN vs DNN high level architecture. (Aslam et al., 2019).....	11
Figure 2: RNN unfolded architecture.	13
Figure 3:LSTM unfolded architecture with internal aspects. (Olah, 2015)	15
Figure 4:LSTM internal steps, forget-gate.....	16
Figure 5: LSTM internal steps, input gate, and candidate values.....	16
Figure 6: LSTM internal steps, updating Cell state.	16
Figure 7: LSTM internal steps, hidden layer and output values.....	16
Figure 8: Bi-LSTM high level architecture diagram.(Pintelas et al., 2020).....	17
Figure 9: GRU internal steps, unfolded. (Chung et al., 2014)	18
Figure 10: Transformer, encoded-decoder high level architecture. (Vaswani et al., 2017).....	20
Figure 11: Past works, analysis of linear methodologies.	21
Figure 12: Table of previously related works.....	23
Figure 13: GRU related works table.	24
Figure 14: Table of technical values taken into consideration for model development.	26
Figure 15: Function to perform Exponential Moving Average	27
Figure 16: Graphical representation of split up raw data.....	28
Figure 17: Libraries and packages needed for data pre-processing.	29
Figure 18: Libraries and packages needed for model development.	30
Figure 19: Breakdown of different activation functions and their parameters.....	31
Figure 20: Survey of loss metrics and their usages in academic papers. (Botchkarev, n.d.)	33
Figure 21: MAPE breakdown with corresponding ability to forecast.(Lewis, 1982, p.42)	34
Figure 22: Code written for call-back declaration.	35
Figure 23: Epoch and batch_size breakdown.	35
Figure 24: Training data, x and y shape.....	36
Figure 25: Validation data, x and y shape.	36
Figure 26: Testing data, x and y shape.	36
Figure 27: Matrix vector multiplication diagram.	37
Figure 28: LSTM model summary diagram.....	39
Figure 29: LSTM model fitment onto training data.	39
Figure 30: LSTM model output whilst training.....	40
Figure 31: Code to predict using LSTM model.	40
Figure 32: GRU model summary diagram.....	41
Figure 33: GRU model fitment to training data code.	41
Figure 34: GRU output whilst fitting data.....	41
Figure 35: Code to predict using the GRU model.....	42
Figure 36: Bi-LSTM model summary diagram.	42
Figure 37: Bi-LSTM fitment on to X_train data.	43
Figure 38: Bi-LSTM model ouput whilst fitting data.	43
Figure 39: Code to predict using the Bi-LSTM model.....	43

Figure 40: Transformer fitment onto training data.	44
Figure 41: Transformer fitment output.	44
Figure 42: Code to use the transformer to predict on X_test.	44
Figure 43: Normalised graphical represnetation of LSTM predictions.	46
Figure 44: GRU graphical represnetaion of predicted outputs.	47
Figure 45: Graphical representation of Bi-LSTM prediction outputs.	47
Figure 46: Transformer predicted outputs graphical representation.	48
Figure 47: Graphical representation of analysis metrics on predicted outputs of all models.	49
Figure 48: Data representation of performance metrics for all implementations.	49
Figure 49: Graphical represnation of all outputs on actual Bitcoin close prices.	50
Figure 50:LSTM model code	61
Figure 51: LSTM normalized predictions.	61
Figure 52: LSTM predictions with normal values.	62
Figure 53: Val_loss and Loss diagram for LSTM	62
Figure 54: Code to build GRU model.....	63
Figure 55: Normalized predicted values for GRU model.	63
Figure 56: Real predicted values GRU model.....	63
Figure 57: Loss and validation loss diagram for GRU model.	64
Figure 58: Code to build the Bi-LSTM model.	65
Figure 59: Normalized predicted values for Bi-LSTM.....	65
Figure 60: Real predicted prices Bi-LSTM model.	65
Figure 61: Loss and validation loss diagram for Bi-LSTM model.	65
Figure 62: Code to build the RNN-Transformer.....	66
Figure 63: Model summary for Transfomer model.	67
Figure 64: Transformer predicted values graphical representation on normalized values.	67
Figure 65: Transformer predictions using real BTC values.....	68
Figure 66: Training and validation loss for transformer model.....	68

Section 10: References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D., Steiner, B., Tucker, P., Vasudevan, V., Warden, P. and Wicke, M. (n.d.). *TensorFlow: A System for Large-Scale Machine Learning This paper is included in the Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). TensorFlow: A system for large-scale machine learning.* [online] Available at: <https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf>.

Agarap, Abien Fred (2018). *Deep Learning using Rectified Linear Units (ReLU)*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1803.08375>.

Agarwal, B., Harjule, P., Chouhan, L., Saraswat, U., Airan, H. and Agarwal, P. (2021). Prediction of dogecoin price using deep learning and social media trends. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, 8(29), p.171188. doi:10.4108/eai.29-9-2021.171188.

Appel, A.W. (2015). Verification of a cryptographic primitive: SHA-256 (abstract). *ACM SIGPLAN Notices*, 50(6), pp.153–153. doi:10.1145/2813885.2774972.

- Arratia, A. and López-Barrantes, A.X. (2021). Do Google Trends forecast bitcoins? Stylized facts and statistical evidence. *Journal of Banking and Financial Technology*. [online] doi:10.1007/s42786-021-00027-4.
- Aslam, S., Herodotou, H., Ayub, N. and Mohsin, S.M. (2019). *Deep Learning Based Techniques to Enhance the Performance of Microgrids: A Review*. [online] IEEE Xplore. doi:10.1109/FIT47737.2019.00031.
- Botchkarev, A. (n.d.). *Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology*. [online] Available at: <https://arxiv.org/pdf/1809.03006.pdf>.
- Brownlee, J. (2018). *Difference Between a Batch and an Epoch in a Neural Network*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>.
- Chai, T. and Draxler, R.R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature. *Geoscientific Model Development*, [online] 7(3), pp.1247–1250. doi:10.5194/gmd-7-1247-2014.
- Chen, Z., Li, C. and Sun, W. (2020). Bitcoin price prediction using machine learning: An approach to sample dimension engineering. *Journal of Computational and Applied Mathematics*, [online] 365, p.112395. doi:10.1016/j.cam.2019.112395.
- Chigozie, E., Nwankpa, W., Ijomah, A., Gachagan, S. and Marshall (2018). *Activation Functions: Comparison of Trends in Practice and Research for Deep Learning*. [online] Available at: <https://arxiv.org/pdf/1811.03378.pdf>.
- Ching, T., Himmelstein, D.S., Beaulieu-Jones, B.K., Kalinin, A.A., Do, B.T., Way, G.P., Ferrero, E., Agapow, P.-M., Zietz, M., Hoffman, M.M., Xie, W., Rosen, G.L., Lengerich, B.J., Israeli, J., Lanchantin, J., Woloszynek, S., Carpenter, A.E., Shrikumar, A., Xu, J. and Cofer, E.M. (2018). Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141), p.20170387. doi:10.1098/rsif.2017.0387.
- Choi, D., Shallue, C., Nado, Z., Lee, J., Maddison, C. and Dahl, G. (n.d.). *On Empirical Comparisons of Optimizers for Deep Learning*. [online] Available at: <https://arxiv.org/pdf/1910.05446.pdf>.
- Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1412.3555>.
- Curran, R. (2022). *More Than Half of Bitcoin Investors Are in the Red, Study Says*. [online] Money. Available at: <https://money.com/bitcoin-crypto-losses-2022/>.
- Fernando, J. (2021). *What Is R-Squared?* [online] Investopedia. Available at: <https://www.investopedia.com/terms/r/r-squared.asp#:~:text=In%20finance%2C%20an%20R%2DSquared> [Accessed 17 May 2022].
- Gov.uk (1990). *Computer Misuse Act 1990*. [online] Legislation.gov.uk. Available at: <https://www.legislation.gov.uk/ukpga/1990/18/contents>.
- Hamayel, M.J. and Owda, A.Y. (2021). A Novel Cryptocurrency Price Prediction Model Using GRU, LSTM and bi-LSTM Machine Learning Algorithms. *AI*, 2(4), pp.477–496. doi:10.3390/ai2040030.
- Harper, D. (2019). *Exploring the Exponentially Weighted Moving Average*. [online] Investopedia. Available at: <https://www.investopedia.com/articles/07/ewma.asp>.
- Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P. and Gérard-Marchant, P. (2020). Array programming with NumPy. *Nature*, 585(7825), pp.357–362. doi:10.1038/s41586-020-2649-2.
- Hileman, Dr.G. (2017). *GLOBAL CRYPTOCURRENCY BENCHMARKING STUDY*. [online] Available at: <https://www.jbs.cam.ac.uk/wp-content/uploads/2020/08/2017-04-20-global-cryptocurrency-benchmarking-study.pdf>.

- Hsu, C.-J., Huang, K.-S., Yang, C.-B. and Guo, Y.-P. (2015). Flexible Dynamic Time Warping for Time Series Classification. *Procedia Computer Science*, 51, pp.2838–2842. doi:10.1016/j.procs.2015.05.444.
- Huang, G., Liu, Z. and Weinberger, Kilian Q (2016). *Densely Connected Convolutional Networks*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1608.06993>.
- Huang, Z., Xu, W. and Yu, K. (2015). Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv:1508.01991 [cs]*. [online] Available at: <https://arxiv.org/abs/1508.01991>.
- Hyndman, R.J. and Koehler, A.B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, [online] 22(4), pp.679–688. doi:10.1016/j.ijforecast.2006.03.001.
- Ji, S., Kim, J. and Im, H. (2019). A Comparative Study of Bitcoin Price Prediction Using Deep Learning. *Mathematics*, 7(10), p.898. doi:10.3390/math7100898.
- Jiahong Li, Hui Bu and Junjie Wu (2017). Sentiment-aware stock market prediction: A deep learning method. *2017 International Conference on Service Systems and Service Management*. doi:10.1109/icsssm.2017.7996306.
- Jiang, Z. and Liang, J. (2017). Cryptocurrency Portfolio Management with Deep Reinforcement Learning. *arXiv:1612.01277 [cs]*. [online] Available at: <https://arxiv.org/abs/1612.01277>.
- Jiawei, X. and Murata, T. (2019). *Stock market trend prediction with sentiment analysis based on LSTM neural network*. [online] waseda.pure.elsevier.com. Available at: <https://waseda.pure.elsevier.com/en/publications/stock-market-trend-prediction-with-sentiment-analysis-based-on-ls> [Accessed 17 May 2022].
- Khan, S., Naseer, M., Hayat, M., Zamir, S.W., Khan, F.S. and Shah, M. (2021). Transformers in Vision: A Survey. *arXiv:2101.01169 [cs]*. [online] Available at: <https://arxiv.org/abs/2101.01169>.
- Kingma, D.P. and Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1412.6980>.
- Kraft, R. (2021). *No, Bitcoin Ownership is not Highly Concentrated – But Whales are Accumulating*. [online] Glassnode Insights - On-Chain Market Intelligence. Available at: <https://insights.glassnode.com/bitcoin-supply-distribution/>.
- Lamport, L. (1983). The Weak Byzantine Generals Problem. *Journal of the ACM*, [online] 30(3), pp.668–676. doi:10.1145/2402.322398.
- Lancashire, L.J., Lemetre, C. and Ball, G.R. (2008). An introduction to artificial neural networks in bioinformatics—application to complex microarray and mass spectrometry datasets in cancer studies. *Briefings in Bioinformatics*, [online] 10(3), pp.315–329. doi:10.1093/bib/bbp012.
- Lewis, C.D. (1982). *Industrial and business forecasting methods : a practical guide to exponential smoothing and curve fitting*. London: Butterworths Scientific, p.42.
- Li, K. (2019). *The Blockchain Scalability Problem & the Race for Visa-Like Transaction Speed*. [online] Medium. Available at: <https://towardsdatascience.com/the-blockchain-scalability-problem-the-race-for-visa-like-transaction-speed-5cce48f9d44>.
- McKinney, W. (n.d.). *pandas: a Foundational Python Library for Data Analysis and Statistics*. [online] Available at: https://www.dlr.de/sc/en/Portaldata/15/Resources/dokumente/pyhpc2011/submissions/pyhpc2011_submission_9.pdf.
- McNally, S., Roche, J. and Caton, S. (2018). Predicting the Price of Bitcoin Using Machine Learning. *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. doi:10.1109/pdp2018.2018.00060.
- Nakano, M., Takahashi, A. and Takahashi, S. (2018). Bitcoin technical trading with artificial neural network. *Physica A: Statistical Mechanics and its Applications*, [online] 510, pp.587–609. doi:10.1016/j.physa.2018.07.017.

- Newman, D. (2021). *Blockchain Node Providers and How They Work*. [online] InfoQ. Available at: <https://www.infoq.com/articles/blockchain-as-a-service-get-block/>.
- Olah, C. (2015). *Understanding LSTM Networks – colah’s blog*. [online] Github.io. Available at: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- Opensource.org. (2019). *The MIT License | Open Source Initiative*. [online] Available at: <https://opensource.org/licenses/MIT>.
- Pathak, P.P. (2021). *Time Series Forecasting – A Complete Guide*. [online] Analytics Vidhya. Available at: <https://medium.com/analytics-vidhya/time-series-forecasting-a-complete-guide-d963142da33f>.
- Pedregosa, F., Pedregosa@inria, F., Fr, Org, G., Michel, V., Fr, B., Grisel, O., Grisel@ensta, O., Blondel, M., Prettenhofer, P., Weiss, R., Com, V., Vanderplas, J., Com, A., Cournapeau, D., Varoquaux, G., Gramfort, A., Thirion, B., Dubourg, V. and Passos, A. (2011). Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot Edouard Duchesnay. *Journal of Machine Learning Research*, [online] 12, pp.2825–2830. Available at: <https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf>.
- Pintelas, E., Livieris, I.E., Stavroyiannis, S., Kotsilieris, T. and Pintelas, P. (2020). Investigating the Problem of Cryptocurrency Price Prediction: A Deep Learning Approach. *Artificial Intelligence Applications and Innovations*, [online] 584, pp.99–110. doi:10.1007/978-3-030-49186-4_9.
- research.surrey.ac.uk. (n.d.). *University Of Surrey | MySurrey*. [online] Available at: <https://research.surrey.ac.uk/ethics>.
- Rudolf, K.O., Ajour El Zein, S. and Lansdowne, N.J. (2021). Bitcoin as an Investment and Hedge Alternative. A DCC MGARCH Model Analysis. *Risks*, 9(9), p.154. doi:10.3390/risks9090154.
- Sak, H., Senior, A. and Google, B. (n.d.). *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*. [online] Available at: <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43905.pdf>.
- Salehinejad, H., Sankar, S., Barfett, J., Colak, E. and Valaee, S. (2018). *Recent Advances in Recurrent Neural Networks*. [online] Available at: <https://arxiv.org/pdf/1801.01078.pdf>.
- Sharma, S. and Mehra, R. (2019). Implications of Pooling Strategies in Convolutional Neural Networks: A Deep Insight. *Foundations of Computing and Decision Sciences*, 44(3), pp.303–330. doi:10.2478/fcds-2019-0016.
- Siarni-Namini, S., Tavakoli, N. and Siarni Namin, A. (2018). A Comparison of ARIMA and LSTM in Forecasting Time Series. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. doi:10.1109/icmla.2018.00227.
- Singh, D. and Singh, B. (2019). Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, p.105524. doi:10.1016/j.asoc.2019.105524.
- Smuts, N. (2019). What Drives Cryptocurrency Prices? *ACM SIGMETRICS Performance Evaluation Review*, 46(3), pp.131–134. doi:10.1145/3308897.3308955.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, [online] 15(56), pp.1929–1958. Available at: <https://jmlr.org/papers/v15/srivastava14a.html>.
- Staudemeyer, R.C. and Morris, E.R. (2019). Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. *arXiv:1909.09586 [cs]*. [online] Available at: <https://arxiv.org/abs/1909.09586>.
- Team, K. (n.d.). *Keras documentation: Callbacks API*. [online] keras.io. Available at: <https://keras.io/api/callbacks/>.
- Team, K. (n.d.). *Keras documentation: Dense layer*. [online] keras.io. Available at: https://keras.io/api/layers/core_layers/dense/.

Vallance, S.P. (2021). *Distributed Ledger Technology: beyond block chain A report by the UK Government Chief Scientific Adviser*. [online] Government Office For Science. Available at: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, Aidan N, Kaiser, L. and Polosukhin, I. (2017). *Attention Is All You Need*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1706.03762>.

Willmott, C.J. and Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, [online] 30(1), pp.79–82. Available at: <https://www.jstor.org/stable/24869236>.

World Intellectual Property Organization (2016). *About IP*. [online] Wipo.int. Available at: <https://www.wipo.int/about-ip/en/>.

Wu, Y., Liu, L., Bae, J., Chow, K.-H., Iyengar, A., Pu, C., Wei, W., Yu, L. and Zhang, Q. (2019). Demystifying Learning Rate Policies for High Accuracy Training of Deep Neural Networks. *arXiv:1908.06477 [cs, stat]*. [online] Available at: <https://arxiv.org/abs/1908.06477> [Accessed 4 May 2022].

www.bcs.org. (n.d.). *BCS Code of Conduct | BCS*. [online] Available at: <https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/#:~:text=respect%20and%20value%20alternative%20viewpoints>.

www.cryptodatadownload.com. (n.d.). *Disclaimer*. [online] Available at: <https://www.cryptodatadownload.com/disclaimer/> [Accessed 17 May 2022].

Yamashita, R., Nishio, M., Do, R.K.G. and Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4), pp.611–629. doi:10.1007/s13244-018-0639-9.

Zhou, D., Kang, B., Jin, X., Yang, L., Lian, X., Jiang, Z., Hou, Q. and Feng, J. (2021). DeepViT: Towards Deeper Vision Transformer. *arXiv:2103.11886 [cs]*. [online] Available at: <https://arxiv.org/abs/2103.11886> [Accessed 17 May 2022].

Section 11: Appendix

Notebook

Link to notebook : [link](#)

LSTM Model

```
def build_lstm():
    model = Sequential()
    #56 and 102
    model.add(LSTM(units = 60, activation = 'relu', return_sequences = True, input_shape =
(X_train.shape[1], 5)))
    model.add(Dropout(0.2))
    model.add(LSTM(units = 120, activation = 'relu', return_sequences = False))
    model.add(Dropout(0.3))
    model.add(Dense(units =1))

    start = time.time()

    opt = tf.keras.optimizers.Adam(learning_rate=0.0001, )

    model.compile(
        loss='mse',
        metrics=[
            #MeanSquaredError(),
            MeanAbsoluteError(),
            MeanAbsolutePercentageError(),
            #tf.keras.metrics.AUC(),
            RootMeanSquaredError(),
        ],
        optimizer=opt)

    print("> Compilation Time : ", time.time() - start)
    return model
```

Figure 50:LSTM model code

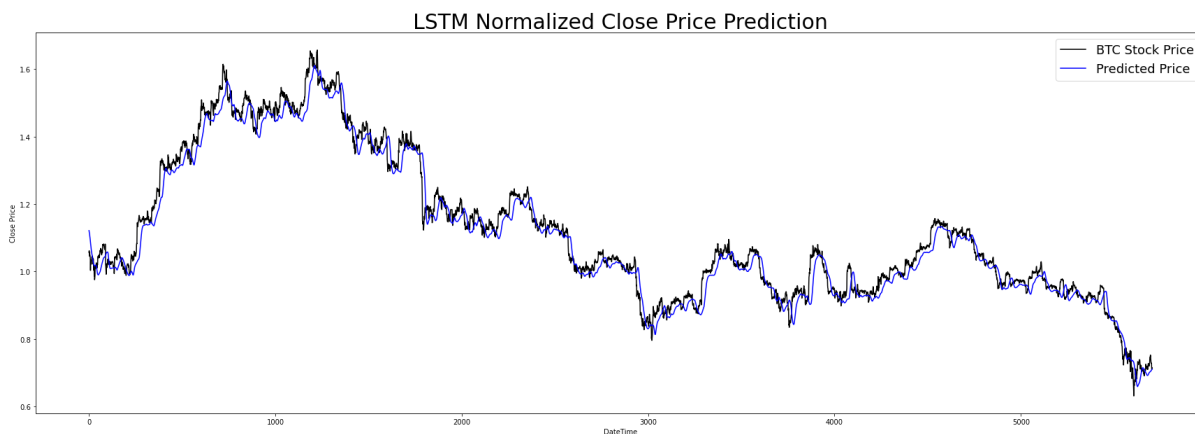


Figure 51: LSTM normalized predictions.

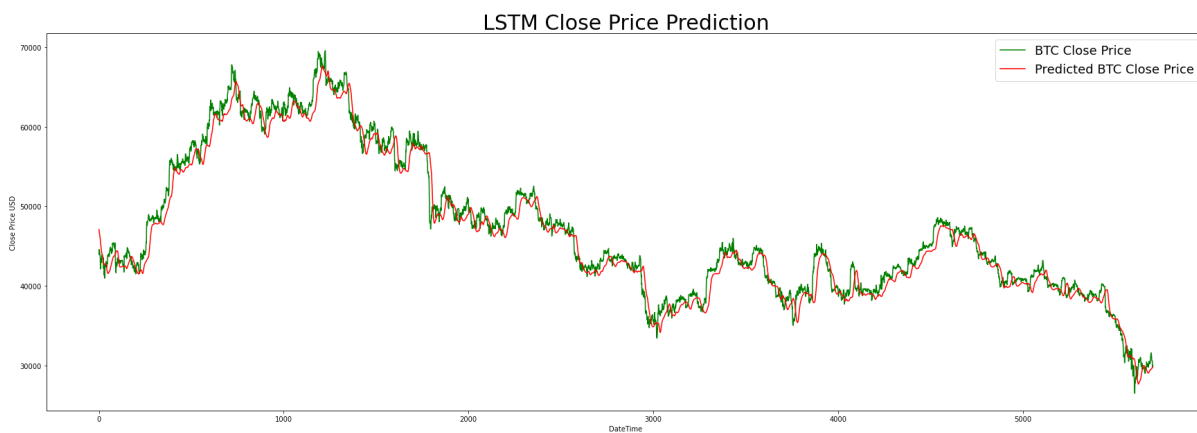


Figure 52: LSTM predictions with normal values.

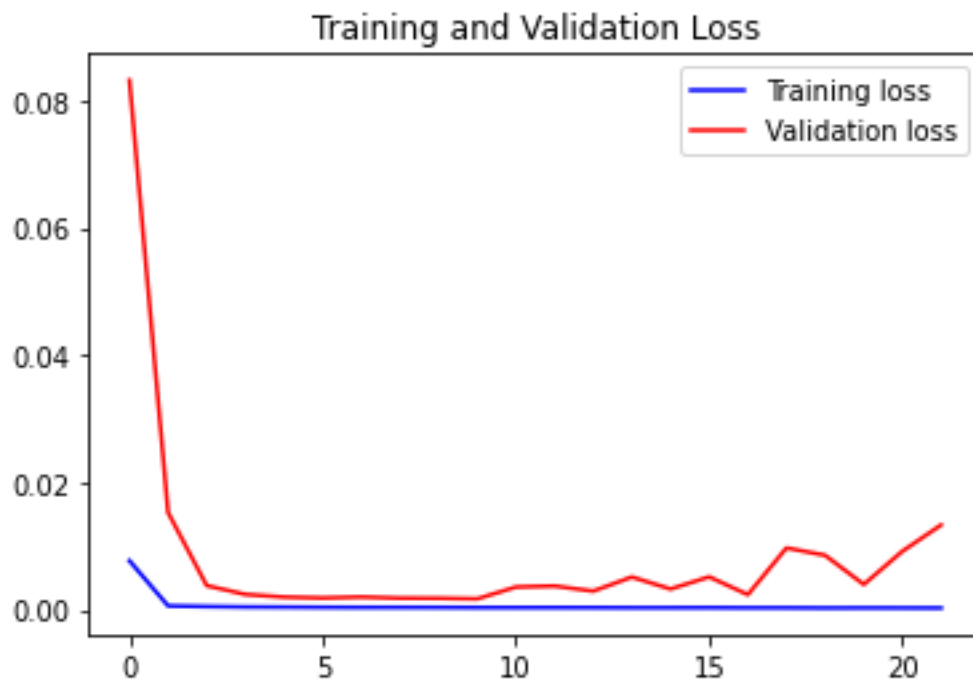


Figure 53: Val_loss and Loss diagram for LSTM

GRU Model

```
def build_gru():
    model = Sequential()
    model.add(GRU(units = 60, activation = 'relu', return_sequences = True, input_shape =
(X_train.shape[1], 5)))
    model.add(Dropout(0.2))
    model.add(GRU(units = 120, activation = 'relu', return_sequences = False))
    model.add(Dropout(0.3))
    model.add(Dense(units =1))

    start = time.time()

    opt = tf.keras.optimizers.Adam(learning_rate=0.0005, )
```

```

model.compile(
    loss='mse',
    metrics=[
        #MeanSquaredError(),
        MeanAbsoluteError(),
        MeanAbsolutePercentageError(),
        #tf.keras.metrics.AUC(),
        RootMeanSquaredError(),
    ],
    optimizer=opt)

print("> Compilation Time : ", time.time() - start)
return model

```

Figure 54: Code to build GRU model.

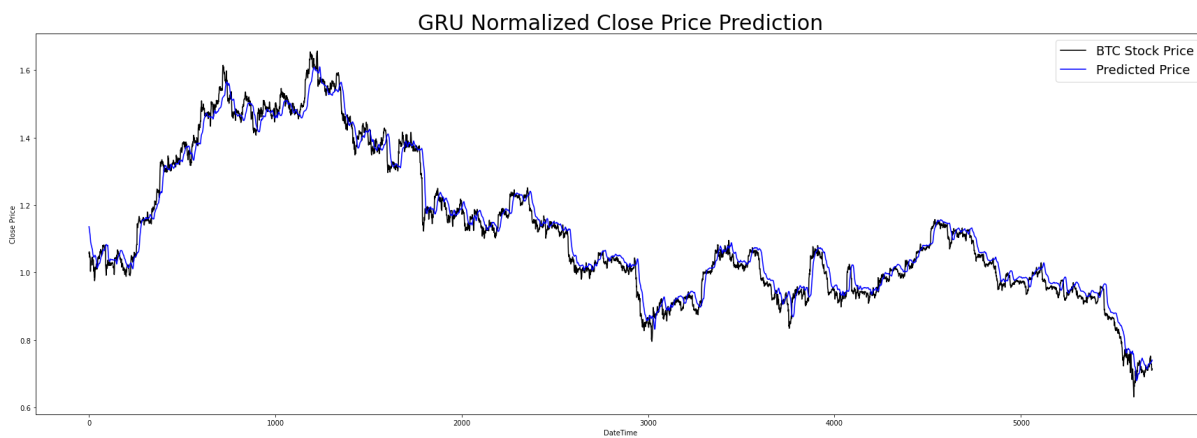


Figure 55: Normalized predicted values for GRU model.

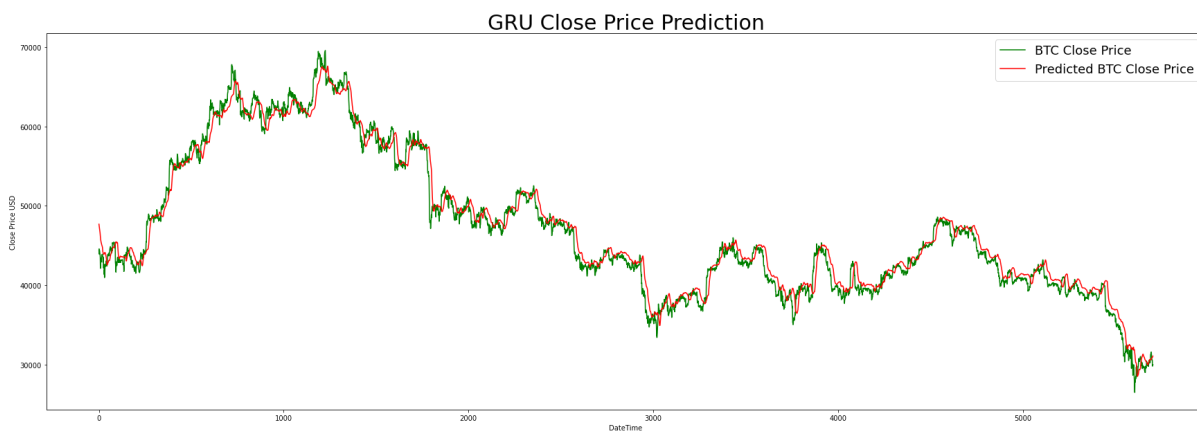


Figure 56: Real predicted values GRU model.

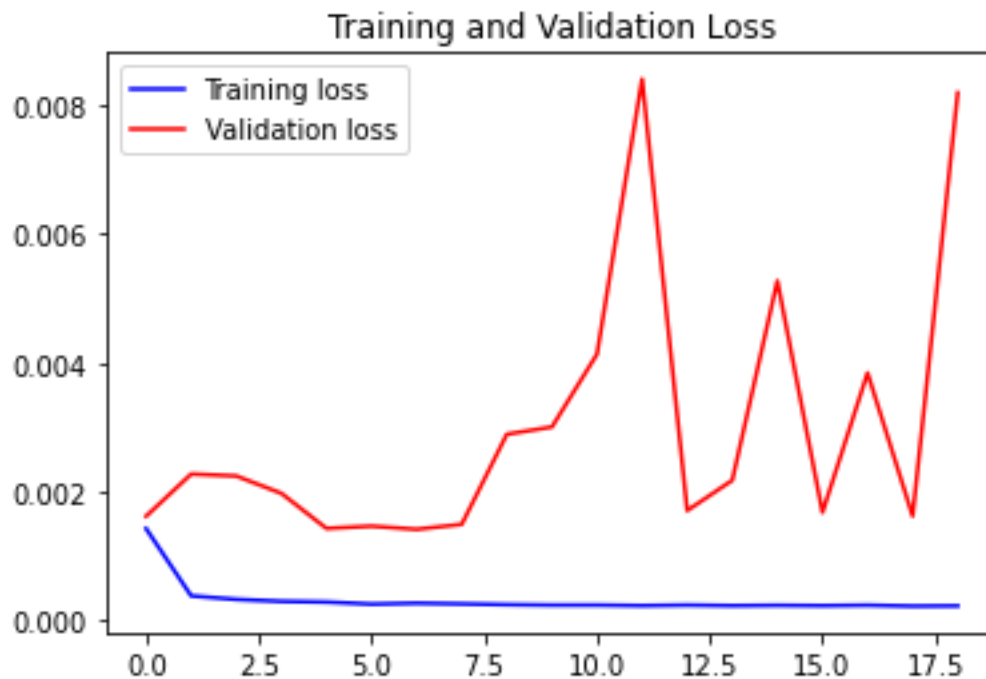


Figure 57: Loss and validation loss diagram for GRU model.

Bi-LSTM Model

```
def build_bilstm():
    model = Sequential()
    model.add(Bidirectional(LSTM(units = 60, activation = 'relu', return_sequences = True,
input_shape = (X_train.shape[1], 5))))
    model.add(Dropout(0.2))
    model.add(Bidirectional(LSTM(units = 120, activation = 'relu', return_sequences = False)))
    model.add(Dropout(0.3))
    model.add(Dense(units =1))

    start = time.time()

    opt = tf.keras.optimizers.Adam(learning_rate=0.0005, )

    model.compile(
        loss='mse',
        metrics=[
            #MeanSquaredError(),
            MeanAbsoluteError(),
            MeanAbsolutePercentageError(),
            #tf.keras.metrics.AUC(),
            RootMeanSquaredError(),
        ],
        optimizer=opt)

    print("> Compilation Time : ", time.time() - start)
    return model
```


Figure 58: Code to build the Bi-LSTM model.

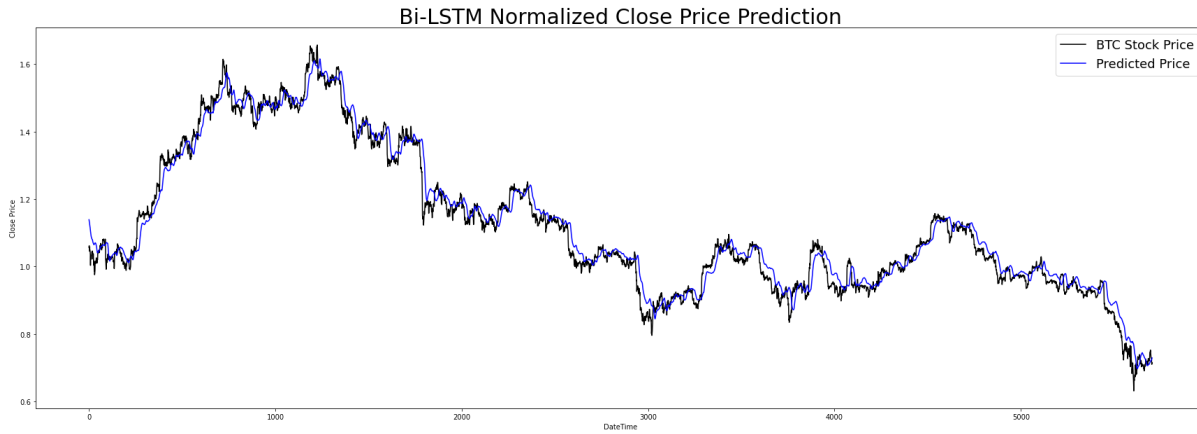


Figure 59: Normalized predicted values for Bi-LSTM

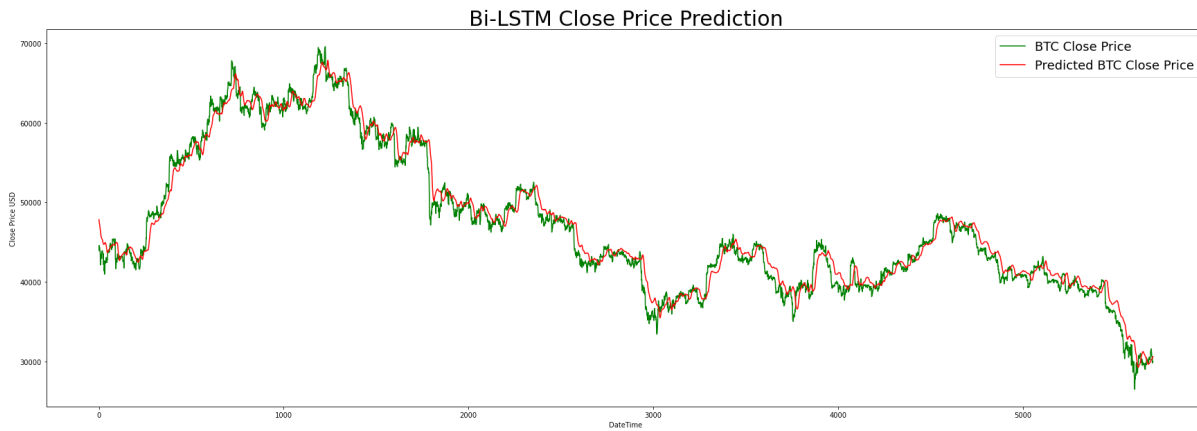


Figure 60: Real predicted prices Bi-LSTM model.

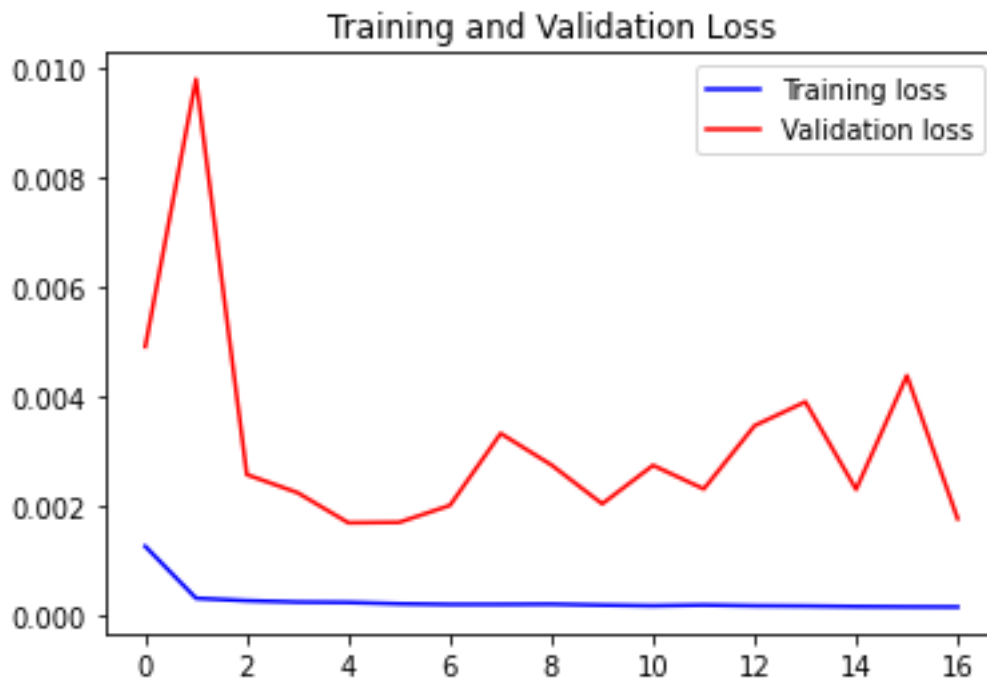


Figure 61: Loss and validation loss diagram for Bi-LSTM model.

Transformer Model

```
def build_model():
    inp = Input(shape = (SEQ_LEN, 5))

    # LSTM before attention layers
    x = Bidirectional(LSTM(60, return_sequences=True, activation = 'relu'))(inp)
    x = Bidirectional(LSTM(120, activation = 'relu', return_sequences=True))(x)

    x, slf_attn = MultiHeadAttention(n_head=3, d_model=150, d_k=32, d_v=32, dropout=0.1)(x, x,
x)

    avg_pool = GlobalAveragePooling1D()(x)
    max_pool = GlobalMaxPooling1D()(x)
    conc = concatenate([avg_pool, max_pool])
    conc = Dense(32, activation="relu")(conc)

    x = Dense(1, activation="sigmoid")(conc)

    model = Model(inputs = inp, outputs = x)

    opt = tf.keras.optimizers.Adam(learning_rate=0.0001, )

    model.compile(
        loss = "mean_squared_error",
        metrics=[
            #MeanSquaredError(),
            MeanAbsoluteError(),
            MeanAbsolutePercentageError(),
            #tf.keras.metrics.AUC(),
            RootMeanSquaredError(),
        ],
        #optimizer = Adam(lr = config["lr"], decay = config["lr_d"]),
        optimizer = opt)

    # Save entire model to a HDF5 file
    #model.save('my_model.h5')

    return model
```

Figure 62: Code to build the RNN-Transformer.

Model: "model_3"

Layer (type)	Output Shape	Param #	Connected to
input_6 (InputLayer)	[(None, 72, 5)]	0	[]
bidirectional_12 (BidirectionalL)	(None, 72, 120)	31680	['input_6[0][0]']
bidirectional_13 (BidirectionalL)	(None, 72, 240)	231360	['bidirectional_12[0][0]']
dense_21 (Dense)	(None, 72, 96)	23040	['bidirectional_13[0][0]']
dense_22 (Dense)	(None, 72, 96)	23040	['bidirectional_13[0][0]']
lambda_18 (Lambda)	(None, 72, 32)	0	['dense_21[0][0]']
lambda_19 (Lambda)	(None, 72, 32)	0	['dense_22[0][0]']
lambda_21 (Lambda)	(None, 72, 72)	0	['lambda_18[0][0]', 'lambda_19[0][0]']
activation_3 (Activation)	(None, 72, 72)	0	['lambda_21[0][0]']
...			
Total params: 356,375			
Trainable params: 356,375			
Non-trainable params: 0			

Figure 63: Model summary for Transformer model.

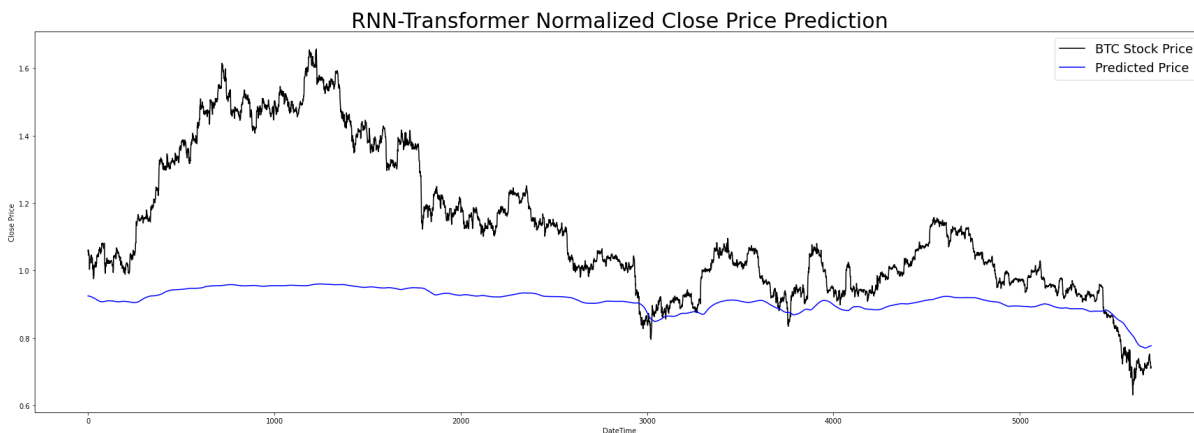


Figure 64: Transformer predicted values graphical representation on normalized values.

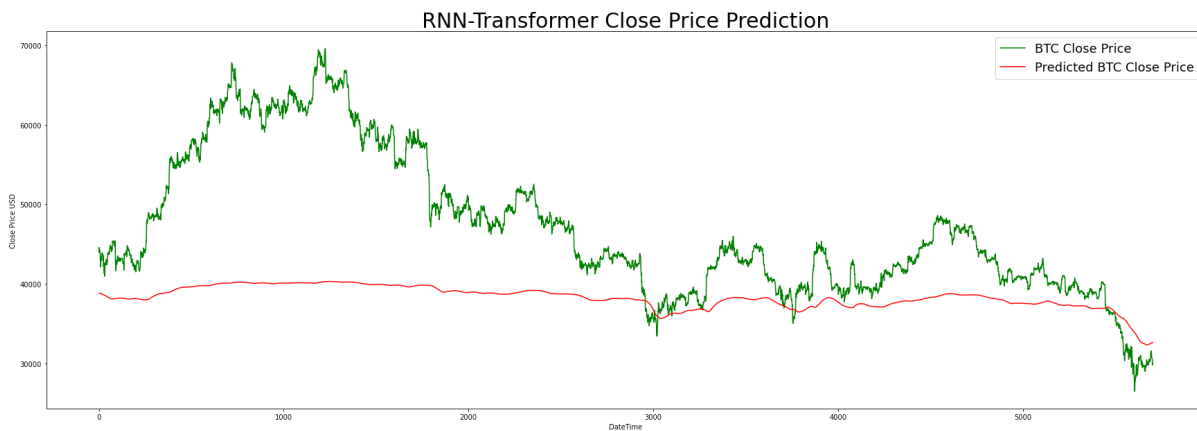


Figure 65: Transformer predictions using real BTC values.

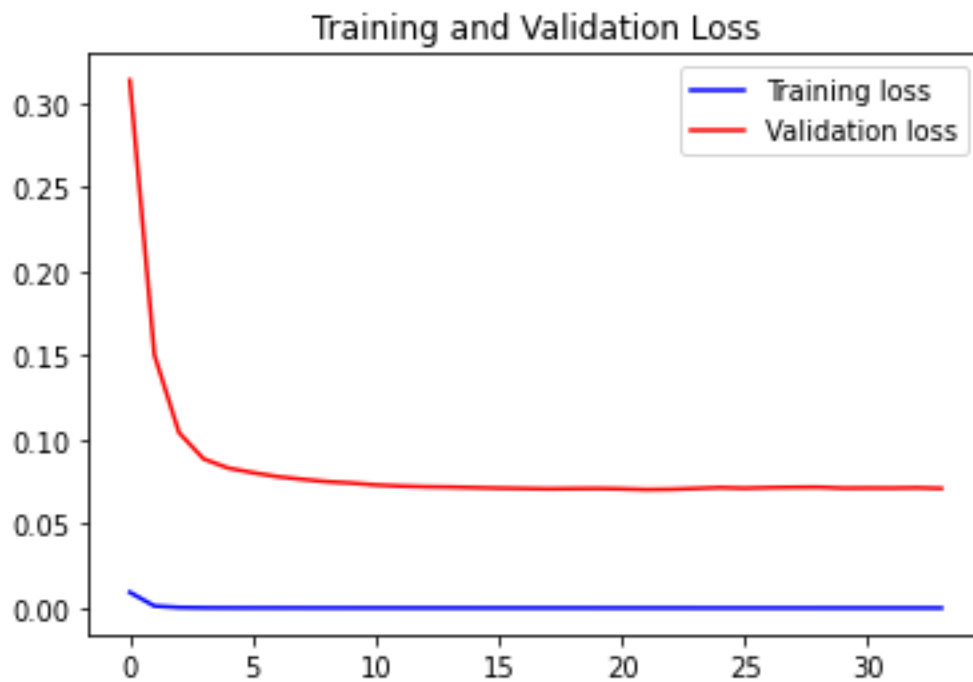


Figure 66: Training and validation loss for transformer model.

SAGE Form

SAGE-HDR (v3.3 23/02/22)

Response ID	Completion date
858429-858411-95029790	16 May 2022, 13:12 (BST)

1	Applicant Name	Adarsh Manoj
1.a	University of Surrey email address	am02434@surrey.ac.uk
1.b	Level of research	Undergraduate
1.b.i	Please enter your University of Surrey supervisor's name. If you have more than one supervisor, enter the details of the individual who will check this submission.	Feng Zhenhua
1.b.ii	Please enter your supervisor's University of Surrey email address. If you have more than one supervisor, enter the details of the supervisor who will check this submission.	z.feng@surrey.ac.uk
1.c	School or Department	Computer Science
1.d	Faculty	FEPS - Faculty of Engineering and Physical Sciences

2	Project title	Using Artificial Neural Networks to Predict Bitcoin Cryptocurrency Values
3	Please enter a brief summary of your project and its methodology in 250 words. Please include information such as your research method/s, sample, where your research will be conducted and an overview of the aims and objectives of your research.	This paper aims to implement machine learning implementations for time series forecasting on cryptocurrency values, for Bitcoin. A more in-depth explanation of cryptocurrencies is found in the Literature Review. Using publicly available open source data.
4	Are you planning to join on to an existing Standard Study Protocol (SSP)? SSPs are overarching pre-approved protocols that can be used by multiple researchers investigating a similar topic area using identical methodologies. Please note, SSPs are only being used by one school currently and cannot be used by other schools. Using an SSP requires permission and sign-off from the SSP owner	NO

5	Are you making an amendment to a project with a current University of Surrey favourable ethical opinion or approval in place?	NO
----------	--	----

6	Does your research involve any animals, animal data or animal derived tissue, including cell lines?	NO
----------	--	----

8	Does your project involve human participants (including human data and/or any human tissue*)?	NO
----------	--	----

9	<p>Does your project involve any type of human tissue research? This includes Human Tissue Authority (HTA) relevant, or non-relevant tissue (e.g. non-cellular such as plasma or serum), any genetic material, samples that have been previously collected, samples being collected directly from the donor or obtained from another researcher, organisation or commercial source.</p>	NO
---	--	----

10	<p>Does your research involve exposure of participants to any hazardous materials e.g. chemicals, pathogens, biological agents or does it involve any activities or locations that may pose a risk of harm to the researcher or participant?</p>	NO
----	---	----

11	Will you be importing or exporting any samples (including human, animal, plant or microbial/pathogen samples) to or from the UK?	NO
----	---	----

12	Will any participant visits be taking place in the Clinical Research Building (CRB)? (involving clinical procedures; if only visiting the CRB to collect/drop-off equipment or to meet with the research team (i.e. for informed consent/discussion) select 'NO').	NO
----	---	----

13	<p>Will you be accessing any organisations, facilities or areas that may require prior permission? This includes organisations such as schools (Headteacher authorisation), care homes (manager permission), military facilities, closed online forums, private social media pages etc. If you are unsure, please contact ethics@surrey.ac.uk.</p>	NO
----	--	----

14	<p>Will you be working with any collaborators or third parties to deliver any aspect of the research project?</p>	NO
----	--	----

15	<p>Are you conducting a service evaluation or an audit? Or using data from a service evaluation or audit?</p>	NO
----	--	----

16	Does your funder, collaborator or other stakeholder require a mandatory ethics review to take place at the University of Surrey?	NO
----	--	----

17	Does your research involve accessing students' results or performance data? For example, accessing SITS data.	NO
----	---	----

18	Will ANY research activity take place outside of the UK?	NO
----	--	----

19	Are you undertaking security-sensitive research, as defined in the text below?	NO
----	--	----

20	Does your project require the processing of special category ¹ data?	NO
----	---	----

21	Have you selected YES to one or more of the above governance risk questions on this page (Q9-Q20)?	NO
----	--	----

22	<p>Does your project process personal data? Processing covers any activity performed with personal data, whether digitally or using other formats, and includes contacting, collecting, recording, organising, viewing, structuring, storing, adapting, transferring, altering, retrieving, consulting, marketing, using, disclosing, transmitting, communicating, disseminating, making available, aligning, analysing, combining, restricting, erasing, archiving, destroying.</p>	NO
----	---	----

23	<p>Are you using a platform, system or server external to the University approved platforms (Outside of Microsoft Office programs, Sharepoint or OneDrive)?</p>	NO
----	--	----

24	Does your research involve any of the above statements? If yes, your study may require external ethical review or regulatory approval	NO
-----------	--	----

25	Does your research involve any of the above? If yes, your study may require external ethical review or regulatory approval	NO
-----------	---	----

26	Does your project require ethics review from another institution? (For example: collaborative research with the NHS REC, the Ministry of Defence, the Ministry of Justice and/or other universities in the UK or abroad)	NO
-----------	---	----

27	<p>Does your research involve any of the following individuals or higher-risk methodologies? Select all that apply or select 'not applicable' if no options apply to your research. Please note: the UEC reviewers may deem the nature of the research of certain high risk projects unsuitable to be undertaken by undergraduate students</p>	<p>NOT APPLICABLE - none of the above high-risk options apply to my research.</p>
----	---	---

28	<p>Does your research involve any of the following individuals or medium-risk methodologies? Select all that apply or select 'not applicable' if no options apply to your research.</p>	<p>NOT APPLICABLE - none of the above medium-risk options apply to my research.</p>
----	--	---

29	<p>Does your research involve any of the following individuals or lower-risk methodologies? Select all that apply or select 'not applicable' if no options apply to your research.</p>	<p>NOT APPLICABLE - none of the above lower-risk options apply to my research.</p>
----	---	--

--	--	--

- I confirm that I have read the University's Code on Good Research Practice and ethics policy and all relevant professional and regulatory guidelines applicable to my research and that I will conduct my research in accordance with these.
- I confirm that I have provided accurate and complete information regarding my research project
- I understand that a false declaration or providing misleading information will be considered potential research misconduct resulting in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies
- I understand that if my answers to this form have indicated that I must submit an ethics and governance application, that I will NOT commence my research until a Favourable Ethical Opinion is issued and governance checks are cleared. If I do so, this will be considered research misconduct and result in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies.
- I understand that if I have selected 'YES' on any governance risk questions and/or have selected any options on the higher, medium or lower risk criteria then I MUST submit an ethics and governance application (EGA) for review before conducting any research. If I have NOT selected any governance risks or selected any of the higher, medium or lower ethical risk criteria, I understand I can proceed with my research without review and

		<p>acknowledge that my SAGE answers and research project will be subject to audit and inspection by the RIGO team at a later date to check compliance.</p>
--	--	--

31	<p>If I am conducting research as a student:</p>	<p>•</p> <p>I confirm that I have discussed my responses to the questions on this form with my supervisor to ensure they are correct.</p> <p>I confirm that if I am handling any information that can identify people, such as names, email addresses or audio/video recordings and images, I will adhere to the security requirements set out in the relevant Data Protection Policy</p>
----	---	---