University of Sussex

# Exploring the Application of Neural Networks in Option Pricing: A Black-Scholes Model Perspective

Departments of Mathematics
Msc Data Science

Candidate Number: 260771
Supervisor: Antony Lewis

# Acknowledgement

I would like to express my sincere gratitude and thank my supervisor Prof Antony Lewis, for supervising this project and for guiding me throughout. His valuable guidance, support, and patience at every stage of my project were highly appreciated. I'd also like to thank Enrico Scalas for introducing me to the Black-Scholes methodology and providing me with the necessary knowledge to head start with this project.

# Abstract

The financial landscape is rapidly evolving, driven by technological advancements and the need for precision in pricing models. This dissertation explores the intriguing possibility of effectively learning and applying the Black-Scholes model, a fundamental component of financial mathematics, through neural networks. The primary aim is to leverage the power of artificial intelligence to gain deeper insights into option pricing.

This research embarks on a dual exploration. Firstly, it aims to train a neural network to understand the complexities of the Black-Scholes model, a sophisticated mathematical framework. This endeavour has the potential to unlock a new dimension in option valuation, allowing for more flexible and nuanced pricing. The second aspect of this study involves a practical examination of the predictive capabilities of the trained neural network. By comparing its Black-Scholes call price predictions with actual call prices from the dynamic S&P500 market, we aim to assess the model's accuracy and reliability. These insights can provide buyers with a clearer understanding of whether an option is fairly valued, offering a competitive edge in decision-making.

# Contents

# List of Figures

# List of Tables

# Chapter 1
# Introduction

In the realm of modern finance, option pricing holds the utmost importance as it facilitates risk management and decision-making processes for investors, corporations, and financial institutions. The Black-Scholes Model, introduced in 1973 by Fischer Black, Myron Scholes, and Robert Merton, revolutionized the field of options pricing by providing a groundbreaking formula for valuing European-style options. Despite its remarkable success, the Black-Scholes Model has its limitations, especially when dealing with complex market conditions and financial instruments.

In recent years, advancements in artificial intelligence and machine learning have significantly impacted various industries, and finance is no exception. Neural Networks, a class of artificial intelligence algorithms inspired by the human brain's neural architecture, have emerged as powerful tools for solving intricate problems in finance. Their ability to learn from data and adapt to changing market conditions presents a compelling opportunity for enhancing option pricing models.

The aim of this dissertation is to explore the application of Neural Networks in option pricing from a Black-Scholes Model perspective. By blending the traditional framework of the Black-Scholes Model with the innovative capabilities of Neural Networks, we seek to address the model's limitations and create a model that learns the Black-Scholes formula. We will explore the current state-of-the-art methods in applying Neural Networks to option pricing. This involves examining various neural network architectures, such as MLP, LSTM, and GRU, to identify their strengths and weaknesses. Additionally, we will investigate the role of input features, data preprocessing techniques, and hyperparameter tuning in optimizing the performance of these models.

A substantial empirical analysis will be conducted to validate the proposed approach's effectiveness. Historical financial data, including option prices, underlying asset prices, and market volatility, will be utilized to train, test, and validate the neural network-based option pricing model. A comparative analysis with the traditional Black-Scholes Model and other existing pricing models will be performed to gauge the accuracy and efficiency of the neural network approach.

Moreover, this research aims to shed light on the practical implications and potential limitations of adopting neural network-based option pricing models in real-world financial decision-making scenarios. Factors such as computational complexity, data requirements, and interpretability will be discussed to offer insights into these models' practical viability and implementation challenges in the financial industry.

As the digital revolution continues to reshape financial markets, the power of Neural Networks to augment traditional option pricing models represents a significant step forward in quantitative finance. This dissertation strives to contribute to the growing body of knowledge in this domain and provide valuable guidance for financial professionals, traders, and researchers seeking to embrace the potential of artificial intelligence in their quest for more precise and effective option pricing methodologies.

## 1.1 Motivation

I'm excited to learn about option pricing and the world of derivatives. Understanding how options are valued is key for managing risk and making smart investments – something that really interests me. I want to learn more about the famous Black-Scholes formula, which has been used for a long time to price options. It's like a powerful tool that financial experts still use today. I'm curious to see how this formula has stayed useful over the years, even as the financial world keeps changing. I'm also curious about neural networks, a type of computer trick that can find patterns and make predictions. I want to see if we can use these networks to learn the Black-Scholes model and predict how markets might move. It's like using a crystal ball, but based on lots of data and math. But I'm not just stopping at theories. I want to see if this model actually works in the real world. Markets can be wild and messy, so I'm excited to test how well our neural network-infused model holds up in real-life situations. This way, we can find out if all this learning and predicting can actually help us make better decisions when it comes to options and investments. In a nutshell, I'm eager to explore option pricing, figure out what makes the Black-Scholes formula tick, see if neural networks can give it a boost, and find out if this all adds up to practical success in the exciting and unpredictable world of finance.

## 1.2 Objectives

1. Generate a simulated dataset of stock values employing Geometric Brownian Motion, and subsequently compute call option prices through the utilization of the Black-Scholes model.
2. Construct Multi-Layer Perceptron (MLP), Long Short-Term Memory (LSTM), and Gated Recurring Unit (GRU) models, and conduct training to facilitate the assimilation of the Black-Scholes model for the creation of a predictive framework.
3. Execute web scraping procedures to extract data from Yahoo Finance, thereby acquiring real-world figures from the S&P500 equity options market.
4. Apply the previously trained neural network (NN) model to prognosticate call option prices within the collated dataset, subsequently assessing its efficacy in the context of real-world data sets.

## 1.3 Structure of Dissertation

An outline of the dissertation is presented below.

1. Chapter 2 presents an exhaustive overview of prior research pertaining to the subject, encompassing various algorithms whilst also briefing about the options market.
2. Chapter 3 explores different facets of the dissertation, encompassing the Black-Scholes model, the Efficient market hypothesis (EMH) and the Neural Network (NN) models slated for employment.
3. Chapter 4 furnishes an intricate exposition of the devised experimental framework tailored for prediction, encompassing an in-depth evaluation of model performance.
4. Chapter 5 details the executed experiments while visually representing the outcomes of the predictions.
5. Chapter 6 furnishes a synthesis of the study, elucidates the conclusions drawn from the conducted research, and propounds avenues for future research endeavours to undertake.

# Chapter 2
# Background Research

This segment provides an overview of the topics we will discuss. I'll begin by discussing the derivatives market, specifically focusing on options trading within the stock market. Additionally, we'll delve into past efforts involving the integration of Machine Learning into the derivatives market, aiming to develop a reliable model for predicting future option prices with precision. We'll explore a range of conducted experiments in this field, outlining their results and performance. Most of this literature review concentrates on this approach, as it highlights those that have achieved the most impressive outcomes.

## 2.1 Option Pricing

Options trading is a strategy used in the stock market that allows investors to speculate on the future direction of share prices. It provides a unique way to potentially profit from both rising and falling markets. In options trading, you're essentially buying the right to either buy (call option) or sell (put option) a certain number of shares of a particular company's stock at a predetermined price, known as the strike price, within a specific time frame, called the expiration date. This gives you the flexibility to benefit from price movements without actually owning the shares themselves.

Here's a closer look at how options work:

1. **Call Options**: Suppose you anticipate that the shares of Company X, currently trading at £50, will increase in value. You can purchase a call option for a premium (the cost of the option), let's say £5. This gives you the right, but not the obligation, to buy the shares at the agreed-upon £50 price before the option's expiration date. If the share price rises to, say, £60, you can still buy at the lower £50 price, making a profit of £5 after deducting the option cost.
2. **Put Options**: On the other hand, if you predict that Company Y's shares, currently valued at £50, will decline, you can buy a put option. This option allows you to sell the shares at the strike price, even if the market price falls below that. For instance, if the share price drops to £40, you can still sell at £50, gaining a profit of £5 after considering the option expense.

Options trading can offer significant advantages, including the potential for substantial returns with a relatively small upfront investment. However, it's important to note that options come with higher risks compared to traditional stock trading. If the market moves in the opposite direction of your prediction, you might lose the entire premium you paid for the option.

In options trading, the terms "in the money" (ITM), "out of the money" (OTM), and the integration of machine learning play pivotal roles in enhancing trading strategies and outcomes.

1. **In the Money (ITM)**: When an option is "in the money," it means that the option has intrinsic value. For call options, this occurs when the market price of the underlying stock is above the option's strike price. In the case of put options, being in the money happens when the market price is below the strike price. ITM options are valuable because they offer the

opportunity for immediate profit if exercised. As a result, ITM options typically have higher premiums.

2. **Out of the Money (OTM)**: Conversely, an option is "out of the money" if it lacks intrinsic value. For call options, this happens when the market price is below the strike price, and for put options, it occurs when the market price is above the strike price. OTM options are considered riskier since they require the market to move substantially in the predicted direction to become profitable. As a result, OTM options usually have lower premiums.

Machine learning (ML) has emerged as a game-changing technology in options trading. ML algorithms can analyse vast amounts of historical market data to identify patterns, trends, and correlations that might not be evident to human traders. By processing these complex data sets, ML models can help predict price movements and make more informed trading decisions. For example, ML can assist in the development of predictive models for options pricing. These models consider factors like historical price movements, market volatility, interest rates, and more. By training on historical data and continuously learning from new data, ML algorithms can refine their predictions over time. Moreover, machine learning techniques can be employed for risk management and portfolio optimization. ML algorithms can help traders assess the potential risks associated with different options strategies and provide insights into constructing balanced portfolios.

## 2.2 Efficient Market Hypothesis (EMH)

The efficient-market hypothesis (EMH) formulated by Eugene Fama in 1970 is a financial theory that claims that asset prices reflect all available information. According to this theory, it is impossible to beat the market consistently by using technical or fundamental analysis.

However, the EMH has faced challenges from empirical research, which has found evidence of return predictability. This means that some factors can help forecast future prices, even with public information. Despite these challenges, the EMH remains an important theory in finance, as it provides a framework for understanding how asset prices are formed [20].

It has three forms:

1. **Weak Form of Efficient Market:** This form states that current security prices include all the information from the market, such as historical prices, return rates, trading volume, and other market data. This means that past returns cannot predict future returns. Therefore, it is impossible to beat the market by using technical analysis. Technical analysis is the study of historical patterns in stock charts.
2 **Semi-Strong Form of Efficient Market:** This form states that current security prices include all the public information, both market-related and non-market information. This means that new information cannot lead to profits above the market average. Therefore, it is impossible to beat the market by using fundamental analysis. Fundamental analysis is the study of financial statements and other non-market data.
3. **Strong Form of Efficient Market:** This form states that current security prices include all the information, both public and private information. This means that no group of investors can achieve returns above the market average. Moreover, it assumes that transaction costs are minimal.

The EMH relies on the idea of market efficiency, where prices incorporate all the information at hand. This implies that all investors have equal access to information and act rationally, taking advantage of any market inefficiencies. As a result, prices adjust quickly to new information, making it impossible to outperform the market.

The EMH is not without its critics. Some argue that the EMH is too simplistic and ignores the complex factors that affect asset prices. Others question its realism, as it assumes that all investors are rational and have the same information. Despite its critics, the EMH remains an influential theory in finance. It offers a useful framework for understanding how asset prices are determined, contributing significantly to the field of finance [20].

## 2.3 Geometric Brownian Motion

Geometric Brownian Motion (GBM), also known as exponential Brownian motion or Wiener process, represents a continuous-time stochastic phenomenon. Within this mechanism, the logarithm of a variable undergoing random changes follows a Brownian motion pattern, incorporating a drift element. This notion holds notable importance in mathematical finance, primarily employed for emulating stock prices in the context of the Black–Scholes model. It is a landmark of option pricing and risk management in finance. GBM models how financial assets, especially stock prices, change over continuous time.

Stochastic process $S_t$, is considered to adhere to a Geometric Brownian Motion (GBM) if it meets the criteria of the subsequent stochastic differential equation (SDE) [22]:

$$dS_t = \mu S_t d_t + \sigma S_t dW_t$$

Where $W_t$ is a Wiener process or Brownian motion and μ, $\sigma$ are percentage drift and percentage volatility respectively [22].

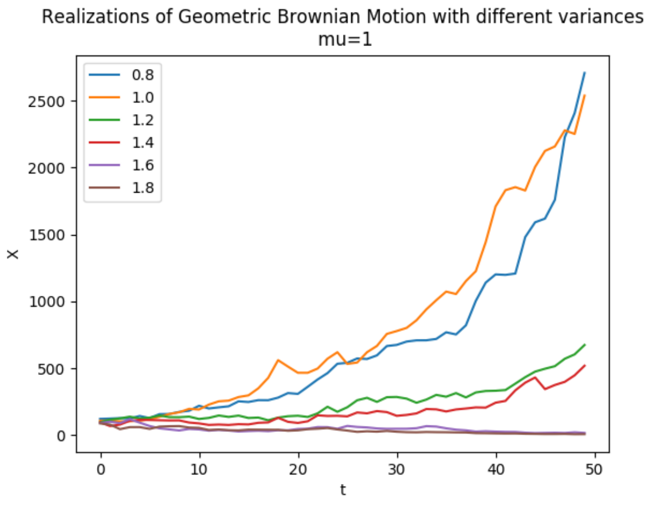


Fig 1: Brownian motion for different variance [22]

The Black-Scholes model uses GBM to describe the dynamic behaviour of underlying assets, enabling it to estimate the value of options with remarkable accuracy. The logarithmic nature of GBM matches well with the assumptions of the Black-Scholes model, which assumes constant volatility and normally distributed returns. This link makes it easier to calculate option prices by considering the asset's current price, strike price, time to expiration, risk-free rate, and volatility.

GBM's integration into the Black-Scholes model helps traders and investors to find fair option prices and make informed decisions about options trading, hedging, and risk management. This model not only offers a way to price options but also insights into the factors that affect option values, such as time decay and implied volatility. However, it's important to recognize that both GBM and the Black-Scholes model have limitations. GBM assumes constant volatility, which might not be realistic in real-world situations, and the Black-Scholes model has been challenged for its simplifications, especially during times of extreme market conditions.

## 2.4 Previous Work

Hutchinson, Lo, and Poggio (2023) [24] proposed a nonparametric method for estimating the pricing formula of a derivative security using learning networks. They suggested using learning networks, which are mathematical models that can learn from data, to approximate the pricing formula of a derivative security without relying on any specific assumptions about the underlying asset or the no-arbitrage condition. Hutchinson, Lo, and Poggio used four types of learning networks: ordinary least squares (OLS), radial basis function (RBF) networks, multilayer perceptron (MLP) networks, and projection pursuit (PP) networks. They trained these networks on a two-year training set of daily options prices and then used them to price and delta-hedge options out-of-sample. They compared their results with the Black-Scholes formula and found that learning networks can recover the Black-Scholes formula from the training data with high accuracy and that they can also price and delta-hedge options out-of-sample with low errors. They show that learning networks outperform the other methods in terms of pricing and hedging performance, especially when the underlying asset exhibits volatility changes or jumps. They also show that learning networks can handle different types of options, such as American options, Asian options, and barrier options.

İltüzer, Z. (2021) [25] compares the performances of neural network and Black-Scholes models in pricing BIST 30 index options, which are options based on the Turkish stock market index. The paper uses different methods to forecast the volatility of the underlying asset, which is a key parameter in option pricing. The paper uses GARCH (Generalized Autoregressive Conditional Heteroskedasticity), implied volatility, historical volatility, and implied volatility index (VBI) to determine the best volatility approach for pricing options according to moneyness and time-to-maturity dimensions. It also analyses how the models perform during tranquil and turbulent periods in the market. They find that neural network and Black-Scholes models have different advantages and disadvantages depending on the type of option, the volatility method, and the market condition. It also discusses the implications of the results for options traders and investors. İltüzer argues that neural networks can provide more accurate and flexible option pricing models than Black-Scholes models, especially when the market is volatile and uncertain. Also suggests that neural networks can capture the nonlinear and complex relationships between option prices and various factors, such as the underlying asset price, the strike price, the time-to-maturity, and the volatility. İltüzer concluded that neural networks can be a valuable tool for option pricing and hedging in the Turkish stock market.

The paper by Malliaris and Salchenberger (1993) [26] is about using an ANN model to estimate the market price of options at closing. Their paper compares the neural network model with the Black-Scholes model. The paper uses the same financial input data for both models, such as the underlying asset price, the strike price, the time-to-maturity, the interest rate, and the dividend yield. It trains the ANN model on historical data of OEX options, which are options based on the Standard and Poor's 100 index. It tests the performance of both models on out-of-sample data and measures the mean squared error between the estimated prices and the actual prices reported by the Chicago Board of Options Exchange. The findings were that the ANN has a lower mean squared error than the Black-Scholes model in about half of the cases examined. It also discusses the differences and similarities between the two models and argues that the neural network model is a robust and flexible model that does not require any distribution assumptions and can learn from data, while the Black-Scholes model is an equilibrium model that relies on certain assumptions about the price dynamics and the volatility of the underlying asset whilst concluding that neural networks can be a useful tool for estimating option prices in complex and uncertain markets.

Robert Culkin and Sanjiv R. Das (2017) [27] published a paper about using deep learning to price options which compares the performance of deep learning models with the Black-Scholes model. The study uses historical data of S&P 500 index options from 1996 to 2015 to train and test deep learning models such as Artificial Neural Networks (ANNs), also referred to as multilayer perceptrons (MLPs). It finds that deep learning models can outperform the Black-Scholes model in terms of pricing accuracy and computational efficiency whilst also discussing the advantages and limitations of deep learning models for option pricing. The study suggests that deep learning models can capture the complex and nonlinear relationships between option prices and various factors, such as the underlying asset price, the strike price, the time-to-maturity, the interest rate, and the volatility.

# Chapter 3
# Preliminaries

The aspiration to forecast financial prices has always captivated investors and traders, constituting an elusive goal. Despite its continued attraction, the task is everything but simple. A paradigm shift has been brought about by the increasing significance of machine learning techniques in this complex endeavour. Like oil, data drives the modern financial landscape and, when used skilfully, may produce dramatic transformations. In this era, as it develops, the neural network shows itself to be a formidable force with untapped potential. The Black-Scholes model, a steadfast cornerstone of financial research, is still relevant today. However, the idea of combining this tested model with the powerful capabilities of the neural network inspires wonder and excitement.

In the era defined by the ascendance of machine learning, the significance of data as a valuable asset amplifies the reshaping of financial analysis. The crossroads where traditional methodologies intersect with state-of-the-art neural network technology invites us to explore uncharted domains. In this quest, the blend of old financial wisdom and modern computing power gives us not just practical insights but also a peek into the future of predicting financial outcomes.

In this dissertation, my aim is to delve into and enhance the comprehension of the Black-Scholes model by examining its viability when combined with a Neural Network. This preliminary section offers an overview of the subjects that will be subsequently addressed in the dissertation, laying the foundation for the forthcoming chapters that will delve into the methodology, findings, and conclusions.

## 3.1 Black Scholes Model

The seminal research paper published in 1973 by economists Fischer Black and Myron Scholes marked a pivotal moment in the field of finance, offering the first widely utilised model for estimating the theoretical value of options. However, the model's evolution gains an additional thread with the profound contributions of Robert C. Merton. Merton was the first to publish a paper expanding the mathematical understanding of the options pricing model and coined the term "Black–Scholes options pricing model”.

By considering crucial variables such as interest rates and anticipated market volatility, this innovative framework provided a valuable tool for guiding options trading. It empowered investors to effectively mitigate risks and manage exposure to market fluctuations. The global recognition of this significant work culminated in the Nobel Memorial Prize in Economic Sciences being awarded to Scholes and Merton in 1997. It also facilitated the creation of novel financial instruments, reshaping the financial landscape and enabling more adept risk management within society. The equation sparked a surge in the trading of options and conferred mathematical credibility to the operations of the Chicago Board Options Exchange and similar markets across the globe.

The equation for Black Scholes option pricing for a European call option is shown below.

$$C = SN(d_1) - Ke^{-rt} N(d_2)$$

Where

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)t}{\sigma\sqrt{t}}$$

and

$$d_2 = d_1 - \sigma\sqrt{t}$$

And where:

1. C = Call option price
2. K = Strike price
3. S = Spot price (Current stock price)
4. t = Time to maturity
5. r = Risk-free interest rate
6. N = Normal cumulative distribution function
7. $\sigma$ = Volatility

The foundation of the Black-Scholes model rests upon a series of key assumptions, encompassing:

1. Absence of Dividends: No dividends are disbursed during the option's duration.
2. Random Walk of Stock Prices: Stock prices exhibit a stochastic behaviour akin to Brownian motion.
3. Transaction Cost Exclusion: No transaction costs are tied to the acquisition of the option.
4. Constant Risk-Free Rate and Volatility: The risk-free interest rate and the volatility of the underlying asset are unvarying and predetermined.
5. Normal Distribution of Returns: The returns generated by the underlying asset follow a normal distribution.
6. European Option Nature: The option adheres to the characteristics of a European option, conferring the right to exercise solely upon expiration.

While the original Black-Scholes model omits consideration of dividends paid during the option's duration, adaptations have been devised to address dividend effects. These adaptations involve calculating the underlying stock's ex-dividend date value. Additionally, market makers, who frequently sell options, adjust the model to encompass the impact of options that can be exercised before expiration.

Advantages of the Black-Scholes Model:

- Provides a structured approach to estimate options' fair value.
- Supports risk management and hedging strategies.
- Guides well-informed decision-making in options trading.

Drawbacks of the Black-Scholes Model:

- Relies on assumptions that may not hold true in real-world scenarios.
- Might not fully capture market dynamics and non-random behaviour.
- A simplified model that may overlook certain complexities.
- Best suited for European-style options and might be less precise for other types [4].

## 3.2 Neural Networks

A neural network represents an artificial intelligence technique that imparts computers with the ability to process information, drawing inspiration from the functioning of the human brain. This method, a facet of machine learning termed "deep learning," employs interconnected nodes or neurons arranged in tiers, mirroring the intricate arrangement found in the human brain. This configuration fosters an adaptable system, enabling computers to glean insights from errors and progressively enhance their performance. Consequently, artificial neural networks strive to tackle intricate challenges, such as condensing extensive documents or discerning facial features, with heightened precision. [3]

In essence, a neural network serves as an innovative computational model, aligning its operations with the intricate neural connections observed in biological systems. Through iterative processes and exposure to vast datasets, the network refines its comprehension and predictive capabilities, paralleling the cognitive learning process in humans. This transformative approach has revolutionized fields ranging from natural language processing to image recognition, expanding the horizons of what computers can accomplish. By emulating the neural underpinnings of human intelligence, neural networks represent a remarkable fusion of cognitive insight and computational prowess, steering artificial intelligence towards more sophisticated and refined realms of application.

In this dissertation, I will test the capabilities of neural networks to assimilate an understanding of the Black-Scholes model and compute options prices. My focus will revolve around three neural network algorithms—Gated Recurring Unit (GRU), Long Short-Term Memory (LSTM), and Multi-Layer Perceptron (MLP). These algorithms will be employed to train the model, with each explained in subsequent sections.

### 3.2.1 MLP (Multilayer Perceptron)

A Multi-Layer Perceptron (MLP) is a feedforward artificial neural network architecture consisting of multiple interconnected layers of nodes or neurons. It is a supervised learning algorithm that comprises input and output layers, along with multiple hidden layers containing numerous stacked neurons. Unlike the Perceptron, where neurons require an activation function that enforces a threshold, such as ReLU or sigmoid, neurons within a Multilayer Perceptron have the flexibility to employ a diverse range of activation functions without restrictions [6].

The learning process of a Multilayer Perceptron (MLP) primarily hinges on the backpropagation algorithm. During training, this method calculates the difference between the predicted and actual outputs, also known as the error. It then backpropagates through the network, layer by layer, to adjust the weights and biases of neurons. This iterative refinement minimizes the error by updating parameters through gradient descent, enhancing the model's predictive accuracy. Backpropagation leverages the chain rule from calculus to compute gradients efficiently, allowing MLPs to fine-tune their internal representations and adapt to complex patterns within data, making them a robust choice for various tasks in machine learning. [7]

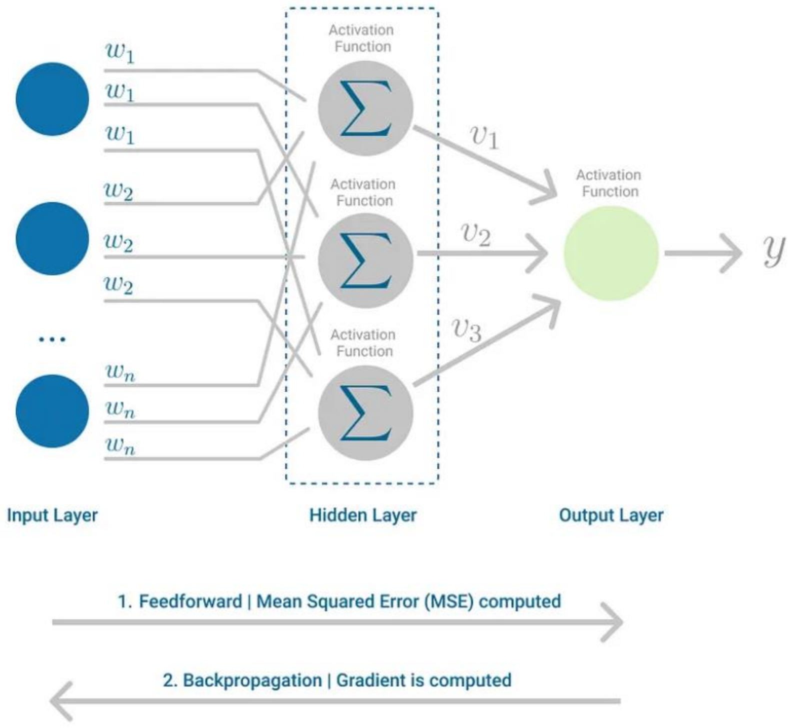


Fig 2: Multilayer Perceptron architecure[6)].

MLPs come in two distinct forms: classification and regression. In the context of this dissertation, I will be employing MLPs for regression purposes. This choice is particularly suited for financial data analysis, such as stocks, which constitutes a significant aspect of the present study.

## 3.2.2 LSTM (Long Short-Term Memory Networks)

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to tackle the vanishing gradient problem and capture long-range dependencies in sequential data [9]. Developed by Sepp Hochreiter and Jürgen Schmidhuber in 1997, LSTMs have become a foundational building block in various fields, such as natural language processing, speech recognition, and time series analysis. It aims to provide a short-term memory for RNN that can last thousands of timesteps, thus "long short-term memory" [10].

Unlike traditional RNNs, LSTMs introduce a memory cell and three gating mechanisms – the input gate, forget gate, and output gate – which control the flow of information within the network. These gates regulate the information flow by selectively allowing the cell state to be updated, forgotten, and used to produce the output. The memory cell's ability to retain information over extended sequences and the gating mechanisms' capacity to manage information flow grant LSTMs their capacity to model long-term dependencies.

The LSTM model has a special memory cell called the "cell state" that helps it remember things for a long time. The cell state is like a conveyor belt that carries information along without changing it. The gates are the ones that decide what information goes in or out of the cell state. They do this by multiplying the information by a number between 0 and 1 and using a sigmoid neural network layer to choose that number. The gates are very important for controlling the information flow inside the LSTM model [12].

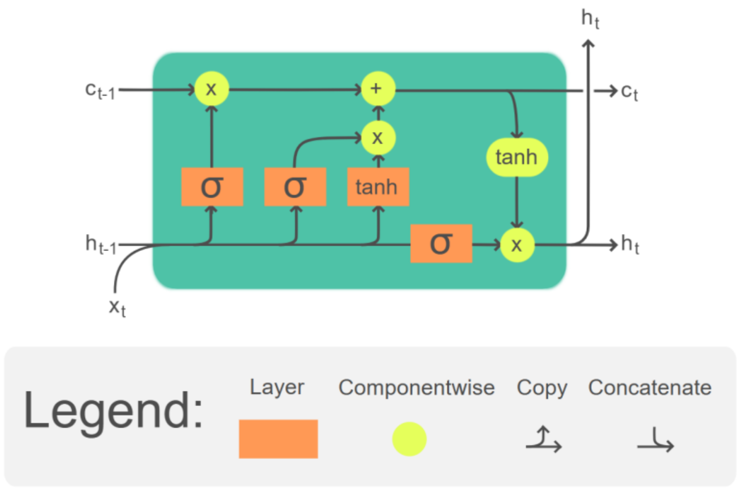


Fig 3: LSTM Model [8]

To explain how an LSTM cell works in a simpler way:

1. **Input Gate**: The input gate looks at the current input and the previous hidden state and decides what new information should be added to the cell state. It gives a score between 0 and 1 for each piece of information. A higher score means more important information.
2. **Forget Gate**: The forget gate looks at the current input and the previous hidden state and decides what old information should be removed from the cell state. It gives a score between 0 and 1 for each piece of information. A lower score means less important information.
3. **Update**: The update step uses the scores from the input gate and the forget gate to change the cell state. The forget gate scores are used to erase some information from the cell state, and the input gate scores are used to add some information to the cell state.
4. **Output Gate**: The output gate looks at the current input, the previous hidden state, and the updated cell state and decides what the next hidden state should be. It gives a score between 0 and 1 for each piece of information. The next hidden state is a combination of these scores and the updated cell state.

The equations for the forward pass of an LSTM cell with a forget gate are [9][13]:

$$f_g = \sigma_g(W_f x_t + U_f h_{t-1} + b_f$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$h_t = o_t \odot \sigma_h(c_t)$$

In the Long Short-Term Memory (LSTM) model, the initial values for the cell state and hidden state are set to zero. These states are updated as the model processes input sequences over time steps indexed by "t." The input vector at each time step is denoted as $x_t$, and there are various activation vectors for gates like the forget gate ($f_t$), input/update gate ($i_t$), and output gate ($o_t$), all of which have values between 0 and 1. The hidden state vector ($h_t$) represents the output of the LSTM unit, and the cell input activation vector ($\tilde{c}_t$) influences the cell state vector ($c_t$). The LSTM model involves weight matrices (W, U) and a bias vector (b) that are learned during training. The activation functions used include the sigmoid function ($\sigma_g$) and the hyperbolic tangent function ($\sigma_c, \sigma_h$), which help regulate the flow of information through the LSTM unit [8].

The study conducted by Liu, Y. and Zhang, X [14] suggests using an LSTM model with realized skewness to price options. The model is tested on ETF50 options (China, 2020-2021) and does better than other models (BS, SVM, RF, RNN) with lower pricing error by 4.50-79.24%. LSTM with realized skewness works best, especially for call options. LSTM's improvement is clear in both realized skewness and realized skewness plus kurtosis dimensions, showing its strong ability to extract information.

### 3.2.3 GRU (Gated Recurrent Unit)

Cho et al [19] proposed the Gated Recurrent Unit (GRU) in 2014 as an effective solution to the vanishing gradient problem that affects standard RNNs. GRU is a simplified variant of LSTM, which achieves comparable performance with a reduced number of parameters. Both GRU and LSTM are designed similarly, using gating mechanisms to regulate the information flow within the network. However, GRU has only two gates: the reset gate and the update gate. The reset gate determines how much of the previous memory state to retain, while the update gate decides how much of the current input to incorporate, thus enhancing the network's ability to process sequential data.

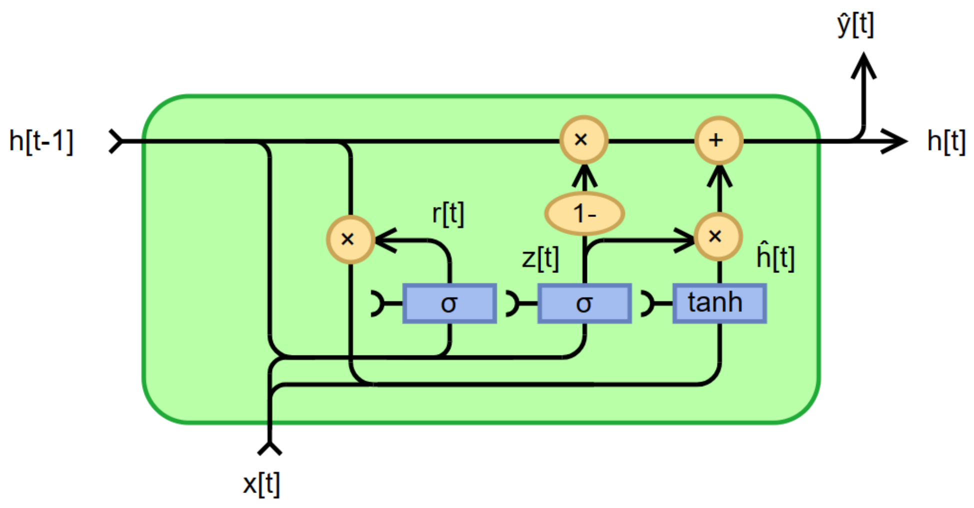


Fig 4: Gated Recurrent Unit, fully gated version [15]

GRU is different from LSTM in that it has only three gates and no separate cell state [17]. The information that is stored in the cell state of an LSTM unit is merged with the hidden state of the GRU unit. This combined information is then passed to the next GRU unit. The three gates of a GRU have the following functions [18]:

1. **Update gate ($z_t$):** It decides how much of the previous knowledge to keep for the future. It is similar to the output gate of an LSTM unit.
2. **Reset gate ($r_t$):** It decides how much of the previous knowledge to erase. It is similar to the combination of the input gate and the forget gate of an LSTM unit.
3. **Current memory gate (($\hat{h}_t$):** It is part of the reset gate, just like the input modulation gate is part of the input gate in an LSTM unit. It adds some non-linearity to the input and makes it zero-mean. Another reason to include it in the reset gate is to reduce the influence of the previous information on the current information that is being passed to the future.

The equations for a fully gated recurrent unit are [15]:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$\hat{h}_t = \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

The symbol $\odot$ denoted the Hadamard product, which is a basic way of multiplying two matrices element by element. The variables involved are $x_t$, which is the input vector; $h_t$, which is the output vector; $\hat{h}_t$, which is the candidate activation vector; $z_t$, which is the update gate vector; and $r_t$, which is the reset gate vector. The model also has matrices W and U, and vector b. There are two important activation functions: σ\sigma, which is usually the logistic function, and ϕ, which is usually the hyperbolic tangent function. σ is often used for gating mechanisms, while ϕ is used for candidate activation calculations.

## 3.3 Activation function

An Activation Function decides whether a neuron should be activated or not [28]. This means that it will decide whether the neuron's input to the network is important or not based on its weighted inputs in the process of prediction using simpler mathematical operations. Activation functions play a fundamental role in shaping the behaviour and capabilities of artificial neural networks, acting as crucial elements that introduce nonlinearity into the network's computations. The primary purpose of an activation function is to introduce a level of complexity that enables neural networks to approximate a wide range of functions, including both linear and nonlinear ones. Without activation functions, neural networks would simply be a linear combination of their input data, incapable of capturing intricate patterns and nuances present in real-world data. The introduction of nonlinearity enables these networks to represent highly complex relationships, making them capable of tackling tasks that require recognizing patterns with varying degrees of intricacy.

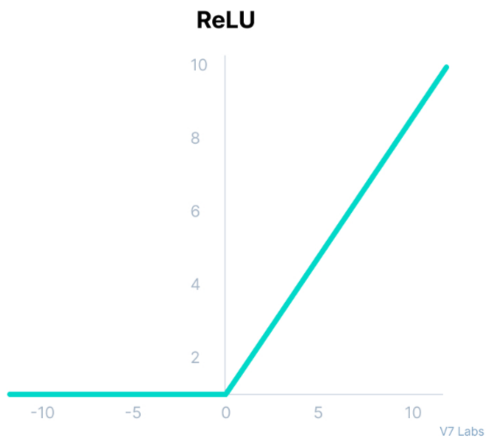


Fig 5: ReLu Activation function

$$f(x) = \max(0, x)$$

In this dissertation the ReLu (Rectified Linear Unit.) function is utilized. It is said to be the most used activation function in the world right now. It’s range lies from 0 to infinity and both the function and it’s derivative are monotonic [29]. The ReLU activation function has several advantages over other activation functions such as sigmoid and tanh. One of the main benefits is its computational efficiency, as only a subset of neurons is activated at any given time. Additionally, the linear, non-saturating property of ReLU helps to speed up the convergence of gradient descent towards the global minimum of the loss function [28].

## 3.5 Adam Optimizer

The Adam optimization technique is a variant of stochastic gradient descent that leverages adaptive estimation of both first-order and second-order moments,[30]. According to the research by Kingma[31], this method boasts computational efficiency, demands minimal memory resources, remains robust when gradients are rescaled along the diagonal, and proves particularly effective for problems characterized by extensive data and parameters.

What sets Adam apart is its combination of the strengths of two other optimization approaches, AdaGrad and RMSProp, to facilitate efficient and adaptive updates throughout the neural network training process[31]. Furthermore, Adam offers numerous advantages over alternative optimization algorithms [32]:

1. Adaptive Learning Rates: Adam possesses the capacity to dynamically adjust its learning rates based on the magnitude of gradients. This adaptability renders it well-suited for a diverse array of problems and architectural configurations.
2. Robustness with Large and Complex Datasets: It excels at handling substantial and intricate datasets, without succumbing to overfitting or becoming trapped in local minima. This makes it an excellent choice for tasks demanding the processing of extensive and intricate data.
3. Efficient Memory Utilization: Adam's low memory requirements make it a practical choice for training deep learning models, particularly when computational resources are limited.

# Chapter 4

# Experimental Setup

In this section, I will discuss the steps and methods I used to ensure the success of my experiments. I will explain my proposed approach, including the process of collecting and preprocessing data. Additionally, I will describe how I trained my dataset on various neural network models and analysed their performance. Finally, I will present the results of my analysis and explain how I used the best-performing neural network to accurately predict real-world option prices.

## 4.1 Experimental Environment

The experimentation was conducted within the Jupyter Notebook environment, employing Python 3.8.5. This programming language is renowned for its user-friendliness, particularly when coupled with a variety of integrated libraries. Various Python libraries such as Matplotlib, Pandas, NumPy, Sklearn, BeautifulSoup, and TensorFlow were harnessed for analysis within this study. The hardware setup for the project involved an Acer laptop operating on Windows 11, driven by an Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz Central Processing Unit, accompanied by 8 GB of RAM.

## 4.2 Proposed Approach

The proposed approach for this project involves experimenting with various algorithms to train neural networks on the Black-Scholes formula, with the goal of using these networks to predict call options for real-world data. To accomplish this, data must first be simulated to mimic the actual stock market. This is achieved using geometric Brownian motion, a mathematical model commonly used in finance to represent the random movements of asset prices over time.

Once the simulated data has been generated, it is used to train the neural networks on the Black-Scholes formula. This involves calculating corresponding strike prices, volatility, and risk-free interest rates, and using these values to generate the corresponding Black-Scholes values. Four different models were used in this study: three Keras models (MLP, LSTM, and GRU) and one Scikit-learn model (MLP). These models were trained on the simulated data and their performance was evaluated.

The best model is then further fine-tuned and used to predict real-world values. To obtain real-world data, web scraping techniques were employed using the BeautifulSoup and yfinance libraries in Python. US S&P500 options data was scraped and used to predict corresponding call option values. The focus was on in-the-money data points with volume, as these are considered to be the most relevant for predicting option prices. This approach should allow us to determine whether a neural network trained on the Black-Scholes model is capable of accurately predicting real-world option prices.

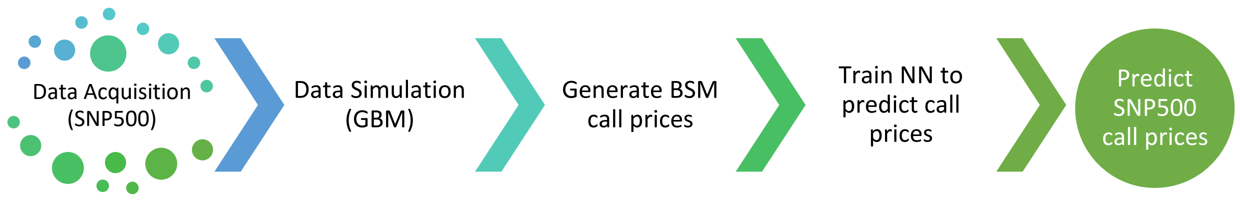

Data Acquisition (SNP500)
Data Simulation (GBM)
Generate BSM call prices
Train NN to predict call prices
Predict SNP500 call prices

Fig 6: Flow chart of the approach

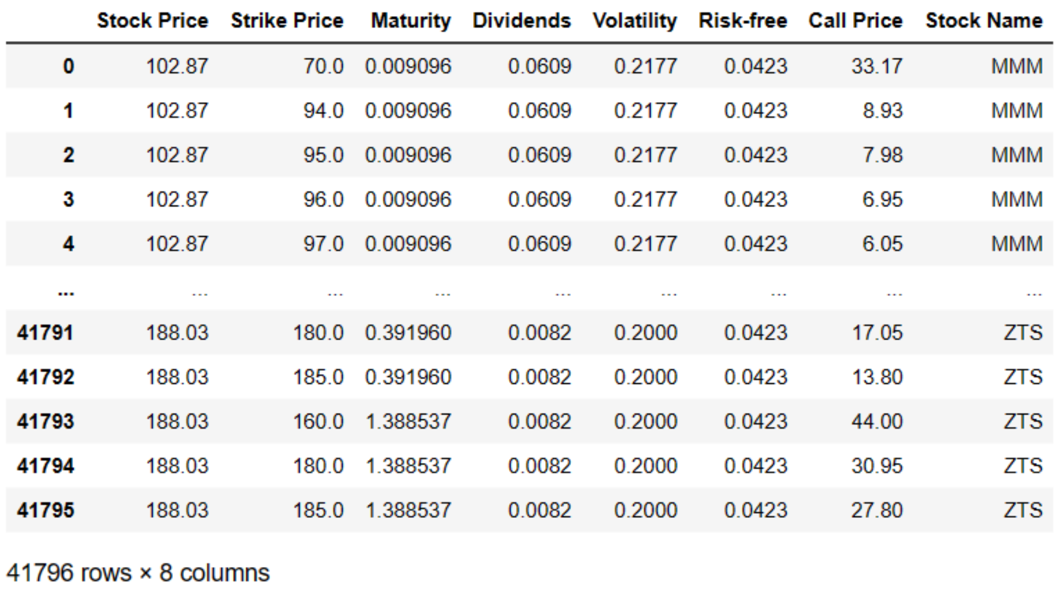
## 4.3 Data Acquisition

The Python code I've created for web scraping is a sophisticated approach to collecting and analysing financial data, specifically centred around options trading activities for companies listed within the S&P 500 index. I began by importing a set of foundational libraries such as BeautifulSoup, datetime, requests and pandas that lay the groundwork for web scraping, data manipulation, and comprehensive analysis. My code is well-structured, comprising a series of functions, each with a distinct purpose that contributes to the overarching data acquisition and processing.

To start with, I created the `*getTickers*` function. This function draws on reliable data sources like Wikipedia to assemble the ticker symbols of S&P 500 companies. This helps in extracting all the S&P500 ticker symbols which will be used further during data extraction from yfinance. Subsequent functions zoom in on the data of options trading, focusing on aspects such as implied volatility and statistics relevant to trading options. The `*voltest*` and `*getStockVol*` functions are used to find the elusive 30-day implied volatility for individual stocks, leveraging the valuable insights offered by the alphaquery.com website. The `*getStockData*` function interfaces with Yahoo Finance to extract an array of stock details, ranging from the stock price to dividend yield and implied volatility.

In addition, my code integrates the `*yfinance*` library, an amazing function in the quest for options-related data. The twin functions, `*get_options_data*` and `*get_options_unix_dates*`, work in tandem to return the expiration dates for options. They are later translated into a more manageable format using Unix timestamps. The `*pick_call_price*` function navigates the complexities of call option pricing, also balancing the intricate interplay between bid and ask prices. This approach not only streamlines data processing but also encapsulates the subtle nuances of the financial domain. Expanding upon this foundation, the `*get_strike_and_call_values'* function returns the in-the-money call option strike prices and their corresponding values.

The backbone of the entire scrapping algorithm is handled with the `*Scrapping*` function. It traverses each iteration through the S&P 500 companies and brings together stock data, maturity calculations, option chain processing, and the creation of a structured DataFrame with curated data. Maturity period is calculated by expiration date - start date. Risk free rate at the time of scrapping the data was *4.23%* which is the 10-year US treasury rate and was taken from ycharts.com.



| | Stock Price | Strike Price | Maturity | Dividends | Volatility | Risk-free | Call Price | Stock Name |
|---|---|---|---|---|---|---|---|---|
| 0 | 102.87 | 70.0 | 0.009096 | 0.0609 | 0.2177 | 0.0423 | 33.17 | MMM |
| 1 | 102.87 | 94.0 | 0.009096 | 0.0609 | 0.2177 | 0.0423 | 8.93 | MMM |
| 2 | 102.87 | 95.0 | 0.009096 | 0.0609 | 0.2177 | 0.0423 | 7.98 | MMM |
| 3 | 102.87 | 96.0 | 0.009096 | 0.0609 | 0.2177 | 0.0423 | 6.95 | MMM |
| 4 | 102.87 | 97.0 | 0.009096 | 0.0609 | 0.2177 | 0.0423 | 6.05 | MMM |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 41791 | 188.03 | 180.0 | 0.391960 | 0.0082 | 0.2000 | 0.0423 | 17.05 | ZTS |
| 41792 | 188.03 | 185.0 | 0.391960 | 0.0082 | 0.2000 | 0.0423 | 13.80 | ZTS |
| 41793 | 188.03 | 160.0 | 1.388537 | 0.0082 | 0.2000 | 0.0423 | 44.00 | ZTS |
| 41794 | 188.03 | 180.0 | 1.388537 | 0.0082 | 0.2000 | 0.0423 | 30.95 | ZTS |
| 41795 | 188.03 | 185.0 | 1.388537 | 0.0082 | 0.2000 | 0.0423 | 27.80 | ZTS |

41796 rows × 8 columns

Fig 7: Web Scrapped data of SNP500

A brief explanation of each term:

- **Stock**: Refers to the ownership units in a company.
- **Strike Price**: Also known as the exercise price, it's the pre-set price at which the owner of an option can buys the underlying stock.
- **Maturity**: Also called the expiration date, it's the date when an option contract expires. After this date, the option is no longer valid, and its value becomes zero.
- **Risk-Free Rate**: The theoretical interest rate at which an investor can invest funds without any risk.
- **Volatility**: Measures the degree of variation of a stock's price over time. High volatility indicates significant price fluctuations, while low volatility indicates more stable prices.

The web-scraped data has a total of *41796* rows and is later imported into the main program, where it is further utilized to evaluate the performance of neural networks in predicting call option prices using the Black-Scholes model. This evaluation is carried out by applying a neural network model trained on simulated data and the Black-Scholes formula to determine if real-world call option prices can be accurately predicted or not.

## 4.4 Data Simulation

In order to obtain more data, I have simulated data that mimics the path of the stock market using geometric Brownian motion. With the help of the stock values generated from the GBM function, other parameters such as the stock price (S), strike price (K), time (t), risk-free rate (R), and a measure of volatility (σ) are generated. These data points were simulated using specific configurations, resulting in a total of 1 million data points being generated.

Here are the details of each variable:

Stock Price, S ~ GBM(mu = 0.01)
Strike Price, K ~ Unif(0.4,1) * S
Time, t ~ Unif (0, 1)
Sigma, $\sigma$ ~ Unif (0.1, 0.8)
Risk-free rate, r ~ Unif (0.01,0.05)

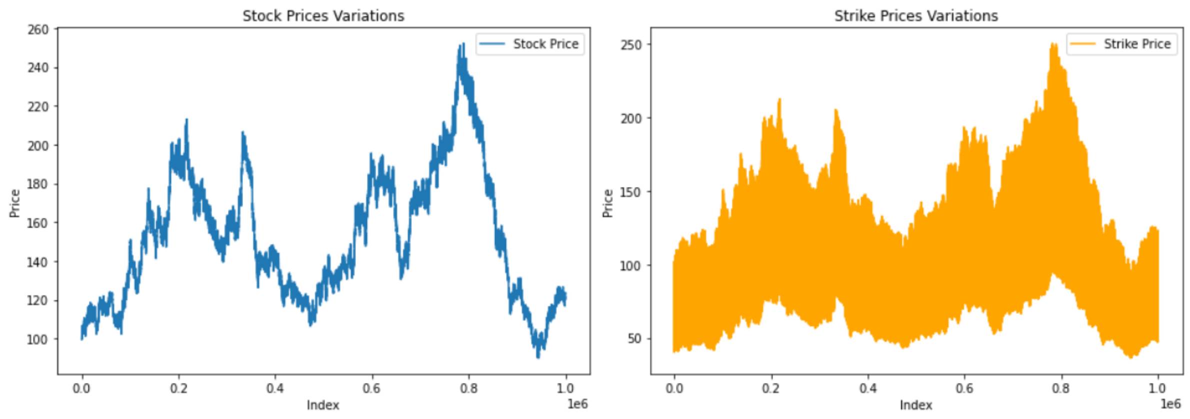
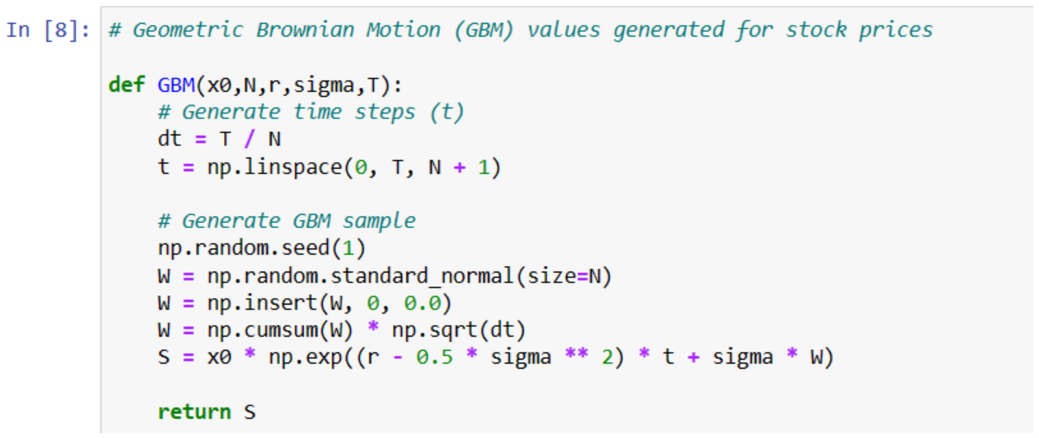


Fig 8: Chart of Stock and Strike prices

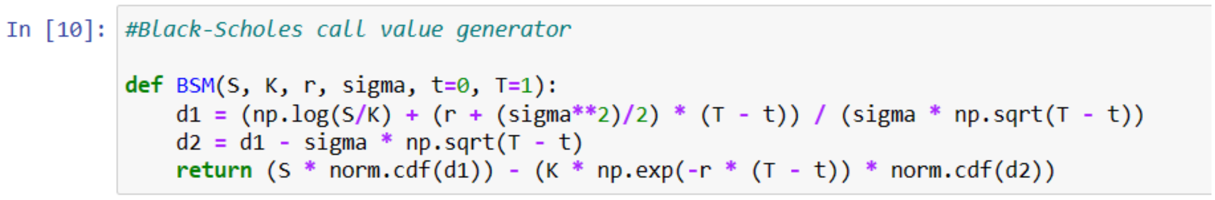
Stock prices were simulated using geometric Brownian motion, also known as the Wiener Process, with a starting price of 100, a drift rate of 0.01, and a volatility of 0.8. Strike prices were generated uniformly from 40 to 100% of the stock price to stay in-the-money. The risk-free rates had a uniform distribution from 0.01 to 0.05, and volatility had a uniform distribution from 0.1 to 0.8. Time-to-maturity was also uniformly distributed from 0 to 1, representing dates expiring the same day to one year from the start date.



```
In [8]: # Geometric Brownian Motion (GBM) values generated for stock prices

        def GBM(x0,N,r,sigma,T):
            # Generate time steps (t)
            dt = T / N
            t = np.linspace(0, T, N + 1)

            # Generate GBM sample
            np.random.seed(1)
            W = np.random.standard_normal(size=N)
            W = np.insert(W, 0, 0.0)
            W = np.cumsum(W) * np.sqrt(dt)
            S = x0 * np.exp((r - 0.5 * sigma ** 2) * t + sigma * W)

            return S
```

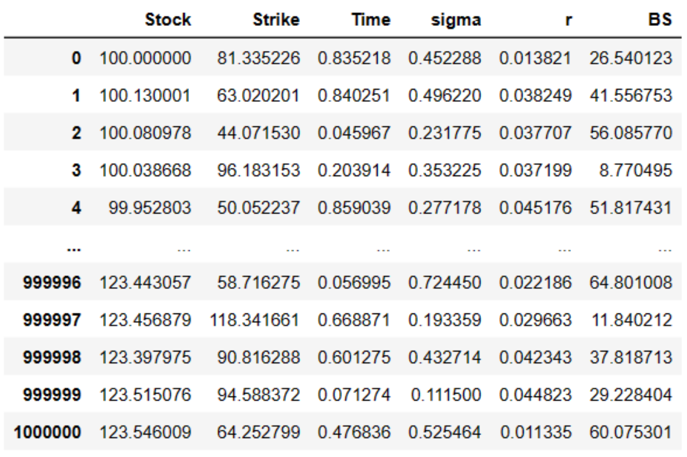
Fig 9: Python implementation of Geometric Brownian Motion

With the help of the Black-Scholes function that was created, data points for Black-Scholes call prices were generated. All the parameters, such as stock price, strike price, maturity, risk-free rate, and volatility, were taken as input for the Black-Scholes function and the corresponding call price was returned. Along with the Black-Scholes call price, the rest of the parameters were then stored in a DataFrame using the pandas library.



```
In [10]: #Black-Scholes call value generator

         def BSM(S, K, r, sigma, t=0, T=1):
             d1 = (np.log(S/K) + (r + (sigma**2)/2) * (T - t)) / (sigma * np.sqrt(T - t))
             d2 = d1 - sigma * np.sqrt(T - t)
             return (S * norm.cdf(d1)) - (K * np.exp(-r * (T - t)) * norm.cdf(d2))
```

Fig 10: Python implementation of Black-Scholes formula



| | Stock | Strike | Time | sigma | r | BS |
|---|---|---|---|---|---|---|
| **0** | 100.000000 | 81.335226 | 0.835218 | 0.452288 | 0.013821 | 26.540123 |
| **1** | 100.130001 | 63.020201 | 0.840251 | 0.496220 | 0.038249 | 41.556753 |
| **2** | 100.080978 | 44.071530 | 0.045967 | 0.231775 | 0.037707 | 56.085770 |
| **3** | 100.038668 | 96.183153 | 0.203914 | 0.353225 | 0.037199 | 8.770495 |
| **4** | 99.952803 | 50.052237 | 0.859039 | 0.277178 | 0.045176 | 51.817431 |
| **...** | ... | ... | ... | ... | ... | ... |
| **999996** | 123.443057 | 58.716275 | 0.056995 | 0.724450 | 0.022186 | 64.801008 |
| **999997** | 123.456879 | 118.341661 | 0.668871 | 0.193359 | 0.029663 | 11.840212 |
| **999998** | 123.397975 | 90.816288 | 0.601275 | 0.432714 | 0.042343 | 37.818713 |
| **999999** | 123.515076 | 94.588372 | 0.071274 | 0.111500 | 0.044823 | 29.228404 |
| **1000000** | 123.546009 | 64.252799 | 0.476836 | 0.525464 | 0.011335 | 60.075301 |

Fig 11: Simulated data

## 4.5 Training the Neural Networks

Using the dataset previously discussed simulated data, I leverage the Python Keras framework to develop and train three distinct neural network architectures: Multi-Layer Perceptron (MLP), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU). Each architecture is designed with an input layer, followed by two hidden layers, and concludes with an output layer. The input layer accommodates five nodes to align with the relevant input parameters, while the hidden layers consist of 20 units each, enabling feature extraction. The output layer encompasses a single node, dedicated to predicting call prices.

To facilitate effective learning, the models are compiled using the Adam optimizer and the loss function is defined as mean squared error. Training is executed on the training data over a span of 20 epochs, with a batch size of 64. To prevent overfitting and promote generalization, an early stopping mechanism is integrated. This mechanism closely monitors the validation loss, halting training if no improvements are observed over a predefined patience period. In case of early stopping, the model's best weights are restored, ensuring optimal performance.

While training the neural networks, we can observe the impact on the loss as the network progresses through the training procedure. During the training of neural networks, we can analyse the changes in loss as the network advances through its training iterations.

The table below shows the performance of each of the three keras neural networks during training over 20 epochs. The numbers in the table represent the values of the validation error. This provides insight into how well each neural network was able to learn and improve its predictions during the training process.

| Epoch | MLP | LSTM | GRU |
|---|---|---|---|
| 1 | 0.9770 | 2.5739 | 0.9128 |
| 2 | 0.5494 | 1.9993 | 0.3305 |
| 3 | 0.4184 | 0.8563 | 0.1952 |
| 4 | 0.3888 | 0.6882 | 0.1301 |
| 5 | 0.3647 | 0.6821 | 0.0790 |
| 6 | 0.3712 | 0.7433 | 0.0515 |
| 7 | 0.3578 | 0.6168 | 0.0439 |
| 8 | 0.3568 | 0.2912 | 0.0440 |
| 9 | 0.3501 | 0.2475 | 0.0526 |
| 10 | 0.3261 | 0.546 | 0.0284 |
| 11 | 0.3583 | 0.1418 | 0.0868 |
| 12 | 0.3052 | 0.1282 | 0.0331 |
| 13 | 0.2910 | 0.0941 | 0.0260 |
| 14 | 0.2906 | 0.1061 | 0.0334 |
| 15 | 0.2703 | 0.043 | 0.0222 |
| 16 | 0.2798 | 0.0881 | 0.0147 |
| 17 | 0.2757 | 0.0524 | 0.0649 |
| 18 | 0.2556 | 0.0243 | 0.0225 |
| 19 | 0.2492 | 0.26 | 0.0877 |
| 20 | 0.2434 | 0.0314 | 0.0146 |

Table 1: Validation error of the implemented keras models

A sci-kit learn model of MLP (MLPRegressor) was applied to the simulated data to compare its performance with the Keras model. The results were very similar, and both models were able to learn the Black-Scholes formula very well from the simulated dataset. The parameters used for the sci-kit learn model were similar to those used for the Keras models, with a batch size of 64 and 20 epochs. This comparison allowed us to determine which model performed better and make an informed decision when choosing a keras or sklearn model for further analysis.

Below are the results of the four models mentioned above. I have compared the Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared value of each model to evaluate their performance. These metrics provide valuable insights into how well each model was able to learn from the data and make accurate predictions for the simulated data. By analysing these results, we can determine which model performed the best and make an informed decision when choosing a model for further analysis.

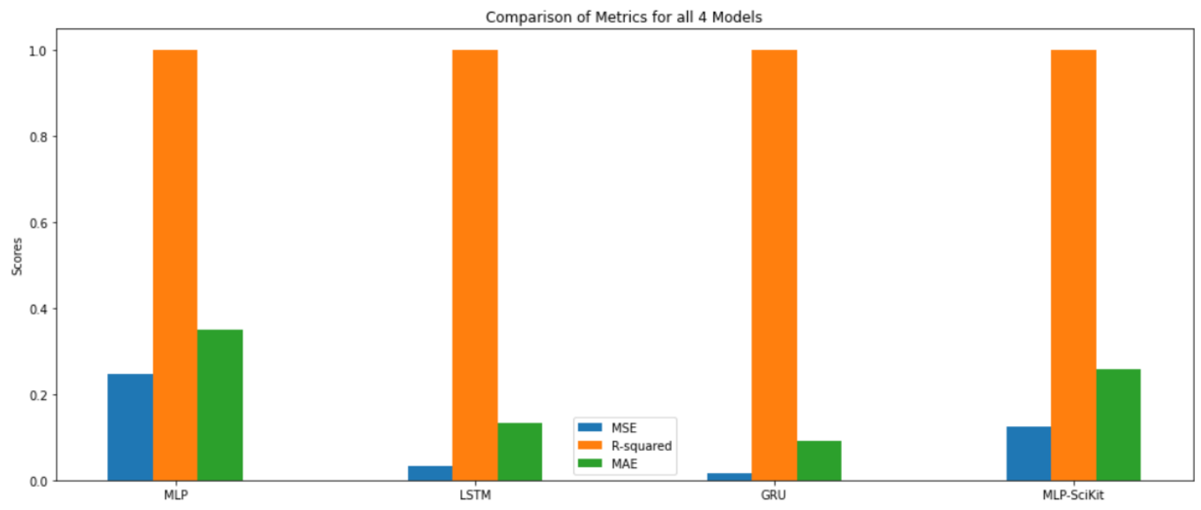
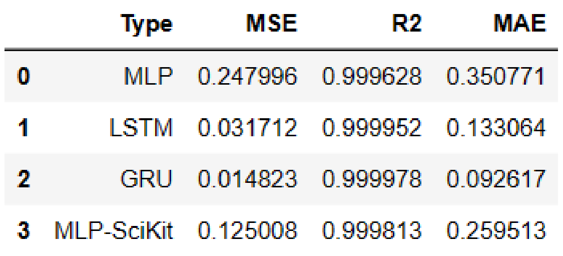


Fig 12: Bar Chart of metrics comparison



| | Type | MSE | R2 | MAE |
|---|---|---|---|---|
| **0** | MLP | 0.247996 | 0.999628 | 0.350771 |
| **1** | LSTM | 0.031712 | 0.999952 | 0.133064 |
| **2** | GRU | 0.014823 | 0.999978 | 0.092617 |
| **3** | MLP-SciKit | 0.125008 | 0.999813 | 0.259513 |

Table 2: Model Performance Metrics Comparison

## 4.6 Predict S&P500 Call prices

Now that we have trained all the models on simulated data, it is clear that the GRU model has performed exceptionally well compared to the rest of the models. As a result, it will be used to predict the real-world values of S&P 500 call option prices. The data was already loaded, so I performed the scaling for the feature data and went on to predict the call prices of S&P stock call option prices using the model that was trained using simulated data. Below, you can see the results of this prediction.

| MSE | R-Squared | MAE |
|---|---|---|
| 14398.9115 | 0.06599 | 58.8709 |

Table 3: Simulated data GRU model performance to actual S&P500 call prices

From the performance metrics above, it can be clearly concluded that GRU model trained on synthetic data fails to accurately predict real-world call prices. To visualize the errors and pricing errors, the figures below will be helpful. The figures below will provide valuable insights into the performance of the model and can help us understand its limitations when it comes to predicting real-world values.

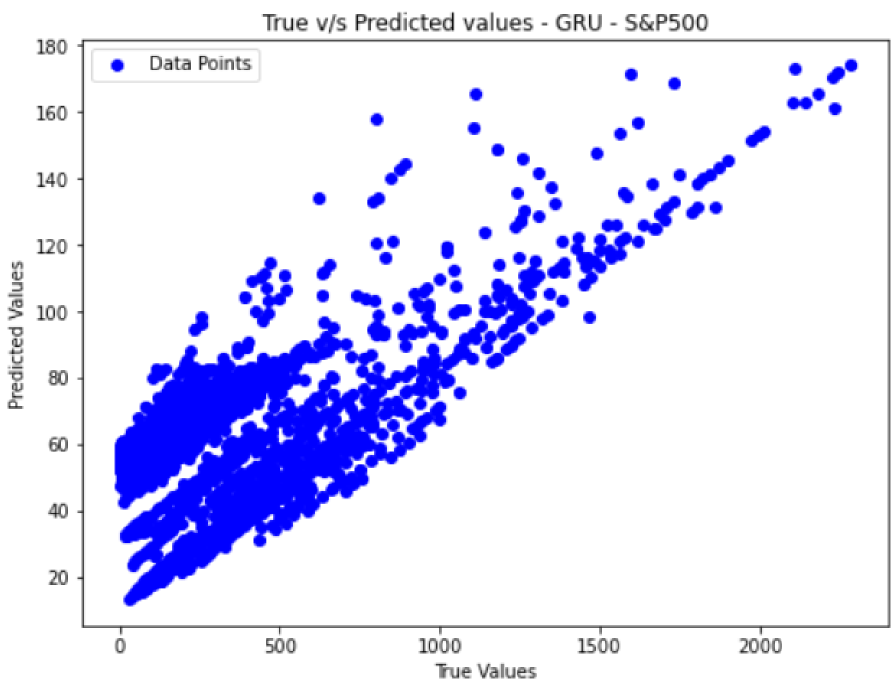


Fig 13: Scatter plot of predicted prices(simulated data) v/s true call prices

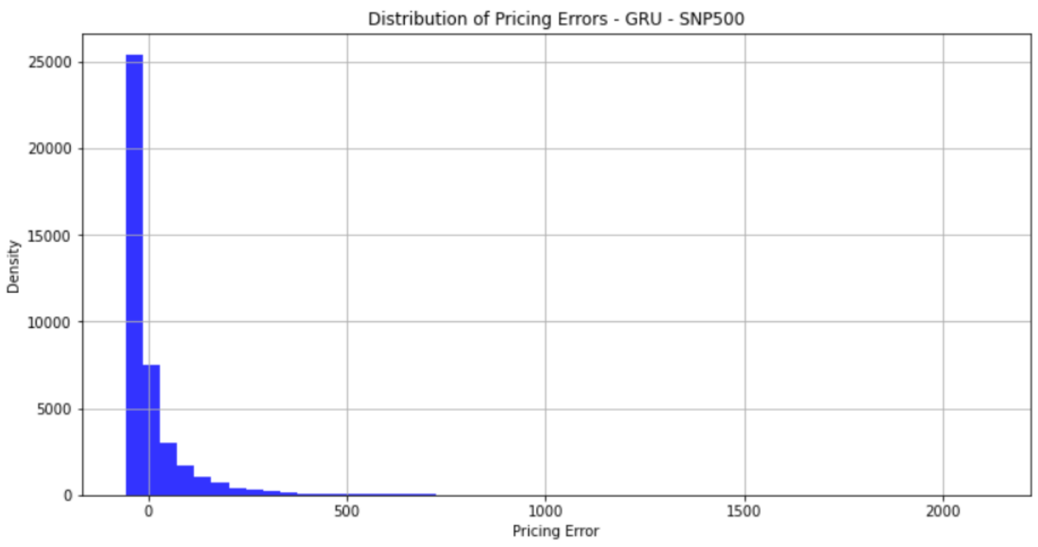


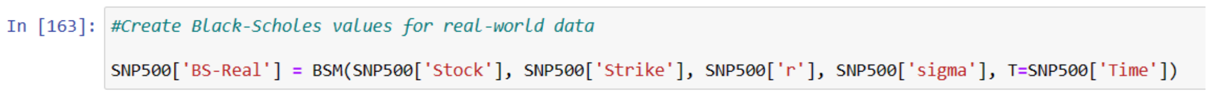
Fig 14: Pricing error distribution for GRU model(Simulated Data trained)

From the scatter plot and pricing error histogram above, we can see that the model did actually predict some of the values nearly well, as evidenced by the pricing errors with values very close to 0. However, there seem to be huge outlier values and the distribution shows high skewness due to which it resulted in very poor performance metric values. Although this is not a reason to select this model, as real-world values do not ideally resonate exactly to a simulated dataset. This is why it makes sense that the model trained on simulated data did not perform well on real-world values. Since this test failed, I will now build a neural network model using real-world data points and generate Black-Scholes values, then compare them with the real call prices to see if that works.

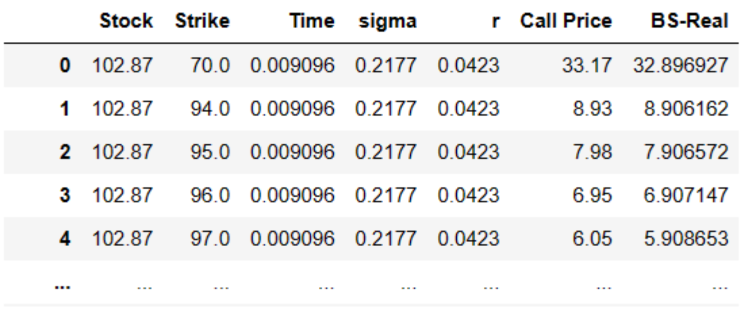
## 4.7 Train Neural Network on S&P500 Dataset

So far, our observations have focused on the behavior of the model when using synthetic data. While the use of synthetic data has greatly contributed to our understanding of the Black-Scholes model, its performance falls short when it comes to predicting call prices for real-world options. As a result, I am now exploring whether models trained on actual data can provide more accurate pricing of options in the market.

It's important to note that a common strategy in options trading involves evaluating whether an option is undervalued or fairly valued in relation to both the market price and the price calculated by the Black-Scholes model. With this in mind, if our model overestimates the price of an option, it could potentially indicate that the option is undervalued.



```
In [163]: #Create Black-Scholes values for real-world data

          SNP500['BS-Real'] = BSM(SNP500['Stock'], SNP500['Strike'], SNP500['r'], SNP500['sigma'], T=SNP500['Time'])
```



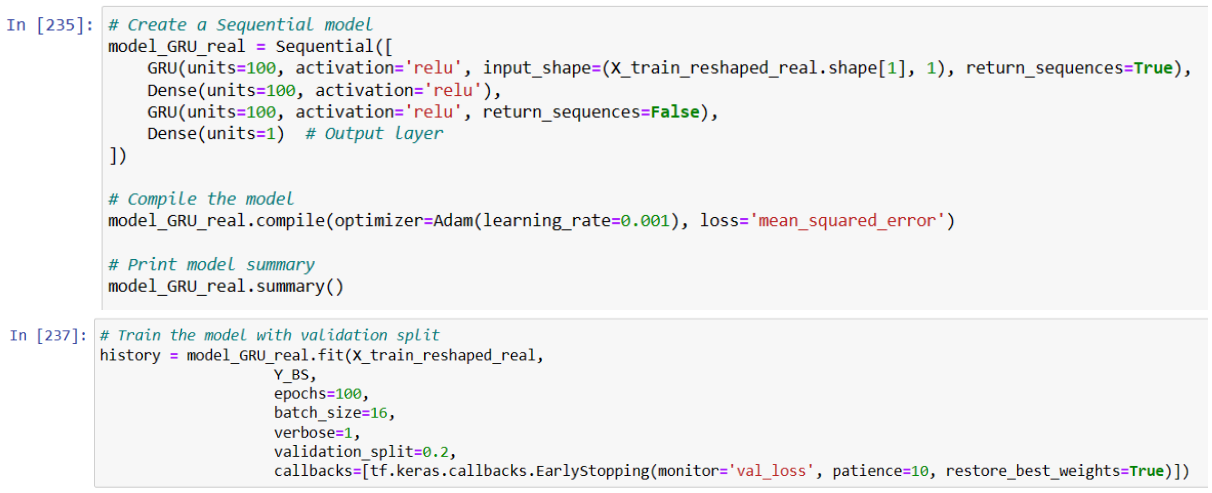
| | Stock | Strike | Time | sigma | r | Call Price | BS-Real |
|---|---|---|---|---|---|---|---|
| **0** | 102.87 | 70.0 | 0.009096 | 0.2177 | 0.0423 | 33.17 | 32.896927 |
| **1** | 102.87 | 94.0 | 0.009096 | 0.2177 | 0.0423 | 8.93 | 8.906162 |
| **2** | 102.87 | 95.0 | 0.009096 | 0.2177 | 0.0423 | 7.98 | 7.906572 |
| **3** | 102.87 | 96.0 | 0.009096 | 0.2177 | 0.0423 | 6.95 | 6.907147 |
| **4** | 102.87 | 97.0 | 0.009096 | 0.2177 | 0.0423 | 6.05 | 5.908653 |
| **...** | ... | ... | ... | ... | ... | ... | ... |

Fig: BS-Real values generated for the S&P500 dataset

Since the best performing model, we had was the Gated Recurrent Unit (GRU), I plan to train the same model using the GRU model from Keras. To start with, I generated the Black-Scholes values for the real-world options chain of S&P500. The model needs to learn to predict the Black-Scholes value, which will then be compared with the real-world call values in order to determine if the call prices can be predicted using the Black-Scholes formula. Since this dataset is much smaller than the simulated data, I have made changes to the parameters to increase its efficiency. Multiple values for parameters were tested and the best one was chosen on the basis of computational efficiency and accuracy.

The model architecture consists of two GRU layers, each with 100 units, utilizing the '*relu*' activation function to introduce non-linearity. These GRU layers process the input data, which is in the shape of time-series sequences, to capture temporal patterns in the option prices. In between the GRU layers, there's a Dense layer with 100 units and '*relu*' activation, enhancing the model's ability to learn complex relationships in the data. The final Dense layer with 1 unit serves as the output layer, producing the predicted option prices. The model is compiled with the Adam optimizer, which adjusts the learning rate automatically during training. The '*mean_squared_error*' loss function is chosen, as it measures the difference between the predicted and actual option prices, guiding the model towards.

Training is carried out on the training data for a total of 50 epochs, with a batch size of 16, as this is a smaller dataset. To prevent overfitting and promote generalization, an early stopping mechanism is implemented. This mechanism monitors the validation loss closely and stops training if no improvements are observed over a patience period of 10.



```
In [235]: # Create a Sequential model
          model_GRU_real = Sequential([
              GRU(units=100, activation='relu', input_shape=(X_train_reshaped_real.shape[1], 1), return_sequences=True),
              Dense(units=100, activation='relu'),
              GRU(units=100, activation='relu', return_sequences=False),
              Dense(units=1)  # Output layer
          ])

          # Compile the model
          model_GRU_real.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

          # Print model summary
          model_GRU_real.summary()

In [237]: # Train the model with validation split
          history = model_GRU_real.fit(X_train_reshaped_real,
                                       Y_BS,
                                       epochs=100,
                                       batch_size=16,
                                       verbose=1,
                                       validation_split=0.2,
                                       callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)])
```

Fig 15: Python implementation of GRU model for S&P500 dataset

The training ran until epoch 31, as the validation error did not increase for 10 epochs. The chart below shows the validation error over time.

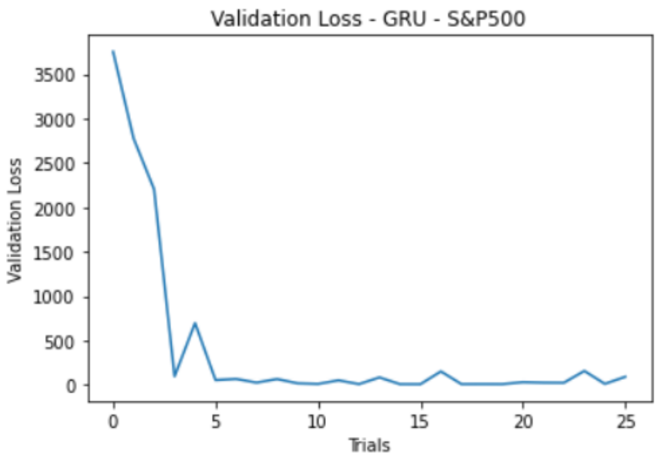


Fig 16: Validation loss chart for GRU model trained on S&P500

The GRU model trained on real-world data performed significantly better than the model trained on simulated data. The results of the performance of the GRU model for predicting the corresponding Black-Scholes values and call prices are shown below. We can see that both have a significantly high R-squared value, indicating a strong relationship between the predictor variables and the outcome variable. This suggests that the GRU model trained on real-world data is able to accurately predict option prices using the Black-Scholes formula.

| | **BS – Values** | **Real Call Prices** |
|---|---|---|
| **MSE** | 22.157 | 1415.5281 |
| **R-Squared** | 0.9987 | 0.9233 |
| **MAE** | 1.6001 | 7.5010 |

Table 4: Performance metrics of GRU model trained on S&P500 dataset

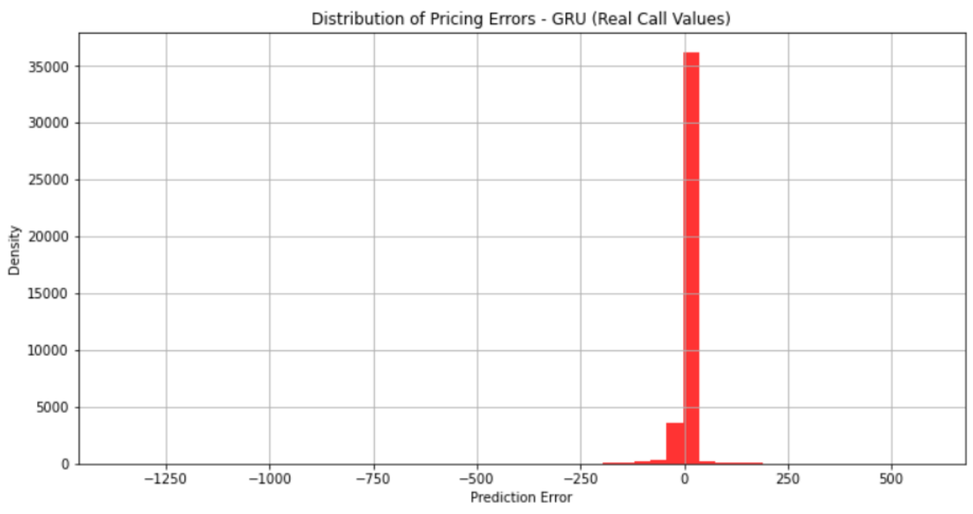


Fig 17: Pricing error distribution for GRU model (S&P500 Dataset trained)

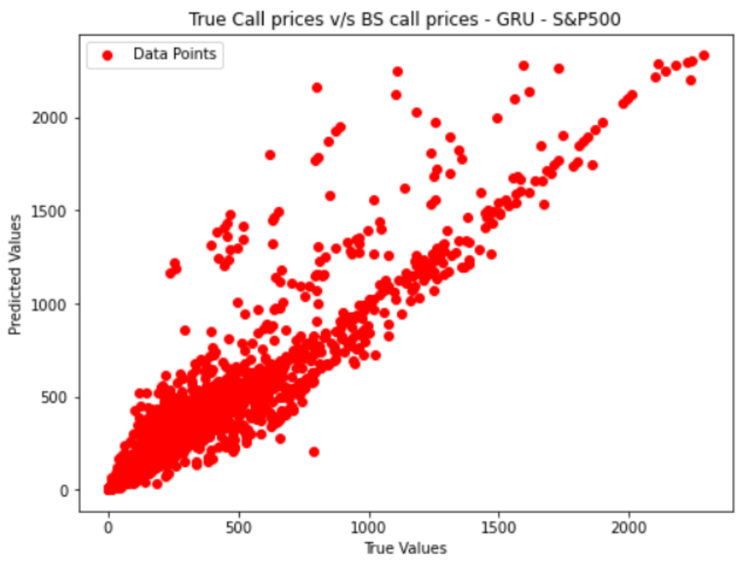
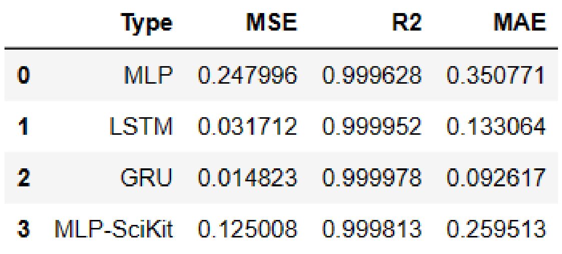


Fig 18: Scatter plot of predicted BS prices(S&P500 data) v/s true call prices

From the first figure of the pricing error distribution, we can confirm that the GRU model was successful in predicting most of the call prices with some error. This is significant, with a good R-squared value of 0.923. This indicates that around 92.3% of the variability in the dependent variable can be explained by the independent variables in the model. This is generally considered a high R-squared value and suggests that the model is performing well in explaining the variation in the data. The scatter plot shows that the majority of the values are being predicted well, although there are a few major deviations, which makes sense for a high MSE value. It's also important to note that the values were predicted from a dataset that was in-the-sample. Hence, before concluding, another test needs to be conducted on a dataset that is new to the model to check if it performs well.

# Chapter 5
# Results

In this section, we present the results of training various neural network architectures to learn the Black-Scholes Pricing formula and predict call prices for the S&P500 options market. The simulated data performed very poorly on learning Black-Scholes formula for a real-world market. Although it was worth noting that the model trained on the real-world values performed very well in comparison to the simulated data.



| | Type | MSE | R2 | MAE |
|---|---|---|---|---|
| **0** | MLP | 0.247996 | 0.999628 | 0.350771 |
| **1** | LSTM | 0.031712 | 0.999952 | 0.133064 |
| **2** | GRU | 0.014823 | 0.999978 | 0.092617 |
| **3** | MLP-SciKit | 0.125008 | 0.999813 | 0.259513 |

Table 5: Model Comparison on simulated data

Since the best performing model among the 3 was the GRU model, it was then fine-tuned and made to learn the Black-Scholes formula using the S&P500 dataset.

Here's a summary of the GRU architecture and training process:

1. **Model architecture:**
    a. Two GRU layers, each with 100 units.
    b. *'relu'* activation function used in the GRU layers to introduce non-linearity.
    c. Input data is in the shape of time-series sequences, allowing the model to capture temporal patterns in option prices.
    d. A Dense layer with 100 units and *'relu'* activation is placed between the GRU layers to enhance the model's ability to learn complex relationships in the data.
    e. The final output layer is a Dense layer with 1 unit, responsible for producing predicted option prices.
2. **Compilation**:
    a. The model is compiled using the Adam optimizer, which automatically adjusts the learning rate during training to optimize convergence.
    b. The chosen loss function is *'mean_squared_error*,' measuring the difference between predicted and actual option prices, guiding the model toward minimizing this error.
3. **Training**:
    a. Training is conducted on the training data for a total of 50 epochs.
    b. A batch size of 16 is used, suitable for a smaller dataset.

c. An early stopping mechanism is implemented to prevent overfitting and promote generalization.
d. This mechanism closely monitors the validation loss and stops training if no improvements are observed over a patience period of 10 epochs.

The S&P 500 dataset used to train the GRU model was scraped on August 28th. Since it was a small dataset, the entire dataset was used to train the GRU model, and the performance metrics indicate the performance of the in-sample dataset. However, to better understand whether the GRU model is good at accurately predicting call prices on an unseen dataset, I used a dataset scraped on July 5th, 2023. This dataset is unseen by the GRU model, so it provides a more accurate representation of the model's performance on new data.

The performance of the GRU model on unseen data can be seen below.

| **MSE** | **R-Squared** | **MAE** |
|---|---|---|
| 4350.9907 | 0.8273 | 17.0146 |

Table 6: Performance of GRU on unseen S&P500 data

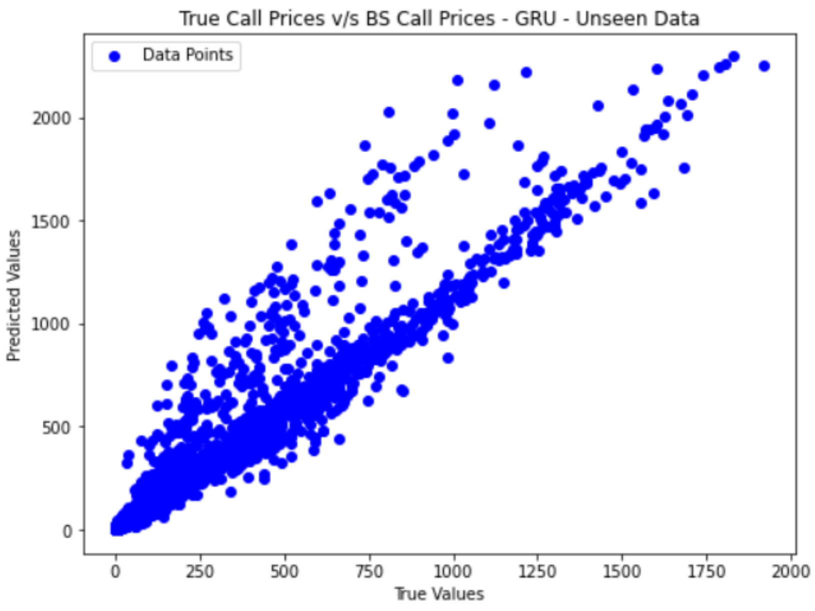


Fig 19: Scatter plot of True Call Price v/s predicted Black Scholes call prices (Unseen data)

We can see that the GRU model performed similarly on unseen data as it did on the trained dataset. From the scatter plot, it is evident that the GRU model predicts the call prices very well for most data points. However, just as with the previous model performance, there are some data points that show high deviation, resulting in a poor MSE value. Nevertheless, an R-squared value of 0.8273 is still good and indicates that approximately 82.73% of the variability in the dependent variable can be explained by the independent variables in the GRU model.

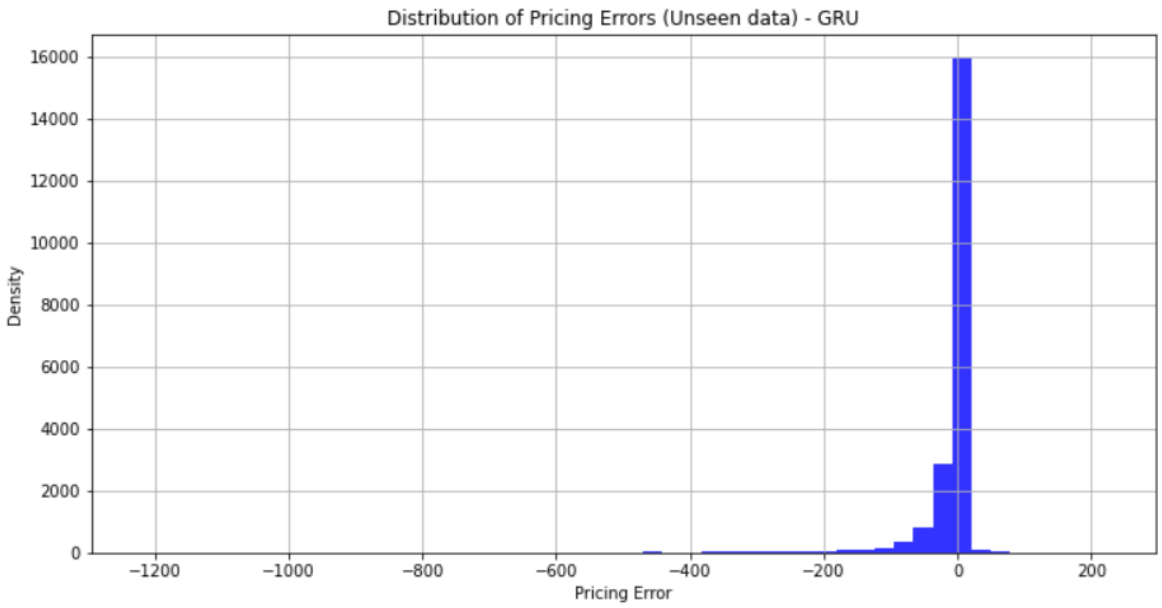


Fig 20: Pricing error distribution for GRU model for S&P500 Call predictions (Unseen data)

From the pricing error histogram, we can clearly see that most of the pricing errors are near 0, indicating that the Black-Scholes values predicted by the model and the actual call prices are very close. This suggests that the model is performing well in predicting call prices and provides valuable insights into the behaviour of option prices in the market. Also the very high negative skewness is the reason behind bad MSE and MAE values.

# Chapter 6
# Conclusion

In this dissertation, we delved into the application of neural networks, specifically the Multi-Layer Perceptron (MLP), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU) architectures, in predicting asset prices with a focus on S&P 500 call option prices. Our research encompassed both synthetic and real-world datasets.

When dealing with simulated datasets, all three architectures proved resilient to overfitting and local minima, demonstrating their suitability for resource-constrained environments due to their minimal memory requirements. A comparison between Keras-based models and scikit-learn's MLPRegressor on synthetic data highlighted the practicality of Keras models. This affirmed their potential for further analysis, considering computational efficiency and user-friendliness.

Coming to real-world data, the GRU architecture was picked, especially in comparison to other models. Its ability to predict Black-Scholes values for S&P 500 call options on simulated data proved and ideal choice for its application to actual market data. However, the attempt to predict real-world S&P 500 call option prices using the GRU model faced challenges. While it accurately predicted some values, it exhibited significant outliers and highly skewed pricing errors, underscoring the complexities of translating knowledge from simulated to real-world financial markets.

Hence, I went ahead into training the GRU model on actual market data. The resulting model exhibited much better performance, with good R-squared values indicating a robust relationship between predictor variables and call option prices. Pricing error distribution and scatter plots further validated its effectiveness in capturing real-world pricing dynamics.

To assess the model's adaptability to unseen data, we evaluated its performance on an independent dataset. Encouragingly, the GRU model demonstrated consistent predictive power, reinforcing its potential as a valuable tool for options pricing.

It is important to note that in financial markets, very high accuracy is crucial. It is difficult to determine if the Black-Scholes model can be directly implemented since even small pricing differences could result in significant losses. However, it is worth noting that the model can help buyers determine if an option is fairly valued or not, even if it does not provide an exact call price.

Predicting financial markets is not an easy task, as no one can accurately predict market movements. Keeping the efficient market hypothesis in mind, we can only perform checks to help buyers understand if an option is fairly valued or not.

# Reference

1. MacKenzie, D. (2023). Black-Scholes at 50: how a pricing model for options changed finance. *Financial Times*. [online] 26 Apr. Available at: https://www.ft.com/content/2ba58381-0de6-4625-bb23-2153c8705f4b [Accessed 17 Aug. 2023].

2. Wikipedia Contributors (2019). *Black–Scholes model*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Black%E2%80%93Scholes_model.

3. AWS (n.d.). *What is a Neural Network? AI and ML Guide - AWS*. [online] Amazon Web Services, Inc. Available at: https://aws.amazon.com/what-is/neural-network/#:~:text=A%20neural%20network%20is%20a.

4. Adam Hayes, (2023). *Black-Scholes Model: What It Is, How It Works, Options Formula*. [online] Available at: https://www.investopedia.com/terms/b/blackscholes.asp#:~:text=The%20Black%2DScholes%20model%2C%20also.

5. scikit-learn.org. (n.d.). *1.17. Neural network models (supervised) — scikit-learn 0.23.1 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

6. Bento, C. (2021). *Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis*. [online] Medium. Available at: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141.

7. Popescu, Marius-Constantin & Balas, Valentina & Perescu-Popescu, Liliana & Mastorakis, Nikos. (2009). Multilayer perceptron and neural networks. WSEAS Transactions on Circuits and Systems.
8. Wikipedia Contributors (2018). *Long short-term memory*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Long_short-term_memory.

9. Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.
10. Olah, C. (2015). *Understanding LSTM Networks*. [online] Github.io. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

11. saxena, S. (2021). *LSTM | Introduction to LSTM | Long Short Term Memor*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/.

12. Intellipaat Blog. (2020). *What is LSTM - Introduction to Long Short Term Memory*. [online] Available at: https://intellipaat.com/blog/what-is-lstm/.

13. Gers, F.A. (1999). Learning to forget: continual prediction with LSTM. *9th International Conference on Artificial Neural Networks: ICANN '99*. doi:https://doi.org/10.1049/cp:19991218.

14. Liu, Y. and Zhang, X. (2023). Option Pricing Using LSTM: A Perspective of Realized Skewness. *Mathematics*, 11(2), p.314. doi:https://doi.org/10.3390/math11020314.

15. Wikipedia Contributors (2019). *Gated recurrent unit*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Gated_recurrent_unit.

16. Kostadinov, S. (2019). *Understanding GRU Networks*. [online] Medium. Available at: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be.

17. d2l.ai. (n.d.). *9.1. Gated Recurrent Units (GRU) — Dive into Deep Learning 0.16.6 documentation*. [online] Available at: https://d2l.ai/chapter_recurrent-modern/gru.html.

18. Aling Gupta, GeeksforGeeks. (2019). *Gated Recurrent Unit Networks*. [online] Available at: https://www.geeksforgeeks.org/gated-recurrent-unit-networks/.

19. Cho, K., Van Merriënboer, B., Gulcehre, C., Bougares, F., Schwenk, H. and Bengio, Y. (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. [online] Available at: https://arxiv.org/pdf/1406.1078.pdf.

20. Wikipedia. (2023). *Efficient-market hypothesis*. [online] Available at: https://en.wikipedia.org/wiki/Efficient-market_hypothesis#Weak [Accessed 23 Aug. 2023].

21. Downey, L. (2023). *Efficient Market Hypothesis (EMH): Definition and Critique*. [online] Investopedia. Available at: https://www.investopedia.com/terms/e/efficientmarkethypothesis.asp.
22. Wikipedia. (2020). *Geometric Brownian motion*. [online] Available at: https://en.wikipedia.org/wiki/Geometric_Brownian_motion.

23. Chello, A. (2020). *A Gentle Introduction to Geometric Brownian Motion in Finance*. [online] The Quant Journey. Available at: https://medium.com/the-quant-journey/a-gentle-introduction-to-geometric-brownian-motion-in-finance-68c37ba6f828.

24. . HUTCHINSON, J.M., LO, A.W. and POGGIO, T. (1994). A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks. The Journal of Finance, 49(3), pp.851–889. doi:https://doi.org/10.1111/j.1540- 6261.1994.tb00081.x.

25. İltüzer, Z. (2021). Option pricing with neural networks vs. Black-Scholes under different volatility forecasting approaches for BIST 30 index options. Borsa Istanbul Review. doi:https://doi.org/10.1016/j.bir.2021.12.001

26. Malliaris, M. and Salchenberger, L. (1993). A neural network model for estimating option prices. Applied Intelligence, 3(3), pp.193–206. doi:https://doi.org/10.1007/bf00871937
27. Culkin, R. and Das, S. (2017). *Machine Learning in Finance: The Case of Deep Learning for Option Pricing*. [online] Available at: https://srdas.github.io/Papers/BlackScholesNN.pdf.

28. Baheti, P. (2022). *12 Types of Neural Networks Activation Functions: How to Choose?* [online] www.v7labs.com. Available at: https://www.v7labs.com/blog/neural-networks-activation-functions.

29. SHARMA, S. (2017). *Activation Functions in Neural Networks*. [online] Medium. Available at: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

30. Team, K. (n.d.). *Keras documentation: Adam*. [online] keras.io. Available at: https://keras.io/api/optimizers/adam/.

31. Kingma, D.P. and Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. [online] arXiv.org. Available at: https://arxiv.org/abs/1412.6980.

32. Mahendra, S. (2023). *What is the Adam Optimizer and How is It Used in Machine Learning*. [online] Artificial Intelligence +. Available at: https://www.aiplusinfo.com/blog/what-is-the-adam-optimizer-and-how-is-it-used-in-machine-learning/#:~:text=problem%20being%20solved.-.

# Appendix

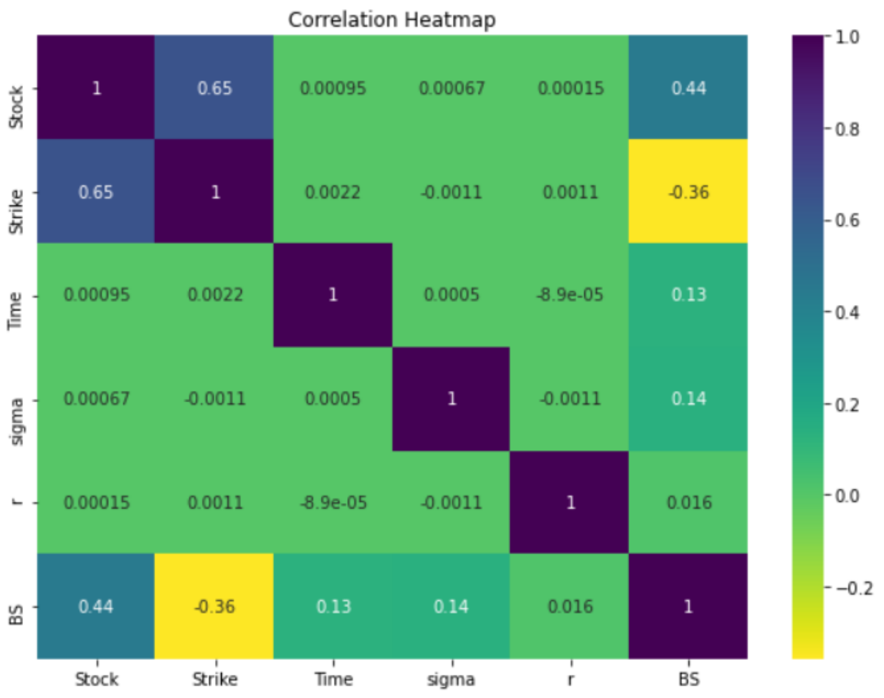


Fig: Correlation heatmap of simulated dataset

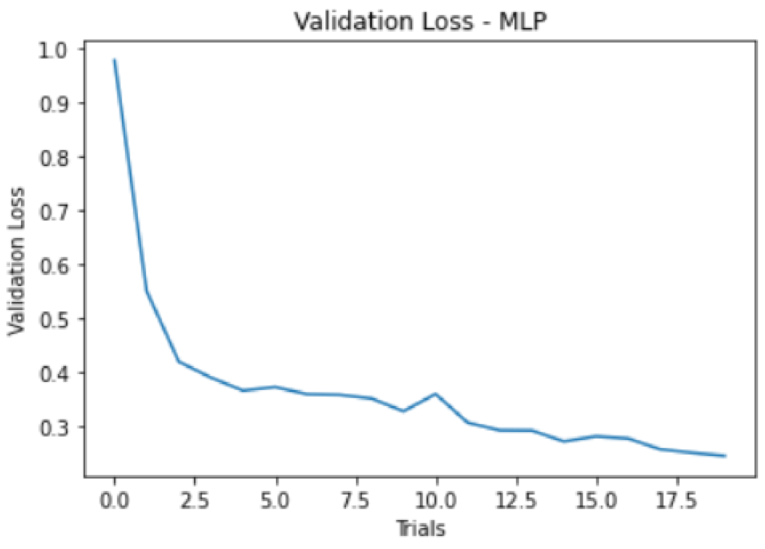


Fig: Simulated Data MLP-Keras Validation loss

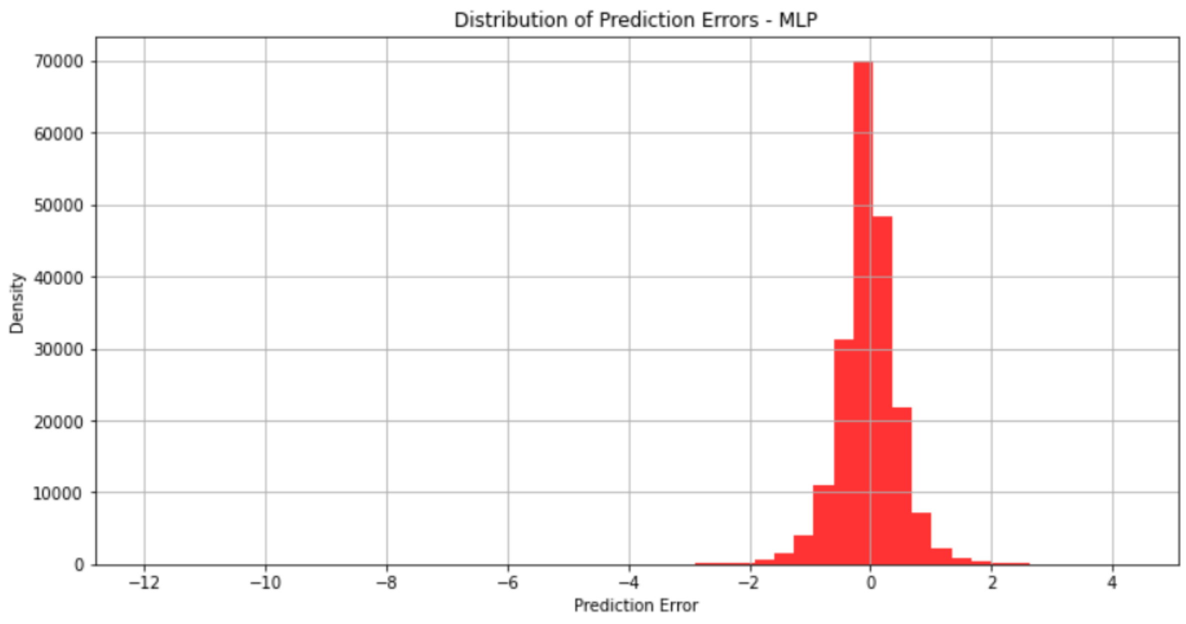


Fig : Pricing Error – MLP – Keras (Simulated Data)

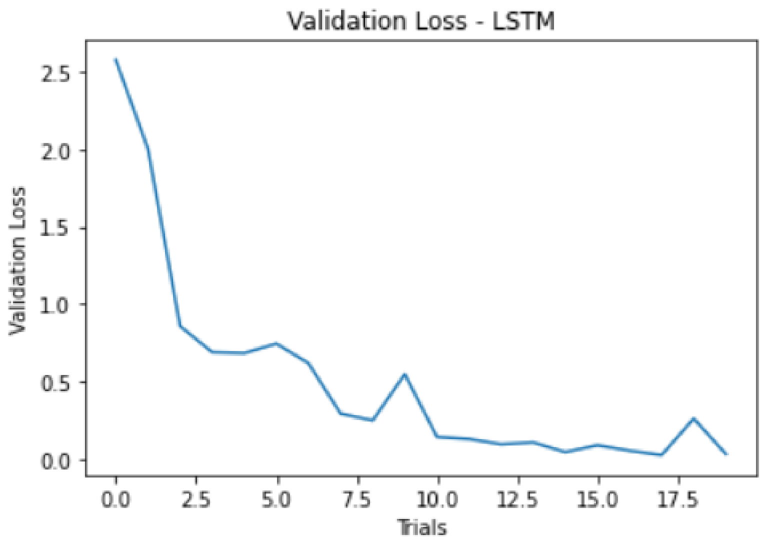


Fig: Validation loss chart LSTM – Keras (Simulated Data)

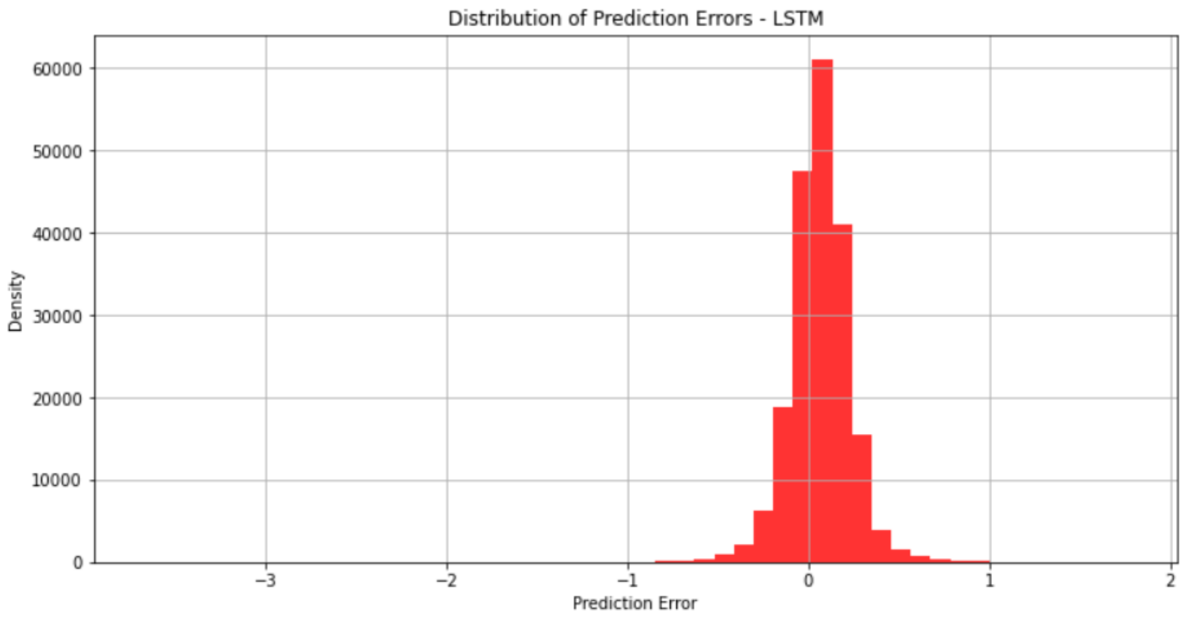


Fig: Pricing Error – LSTM – Keras (Simulated Data)

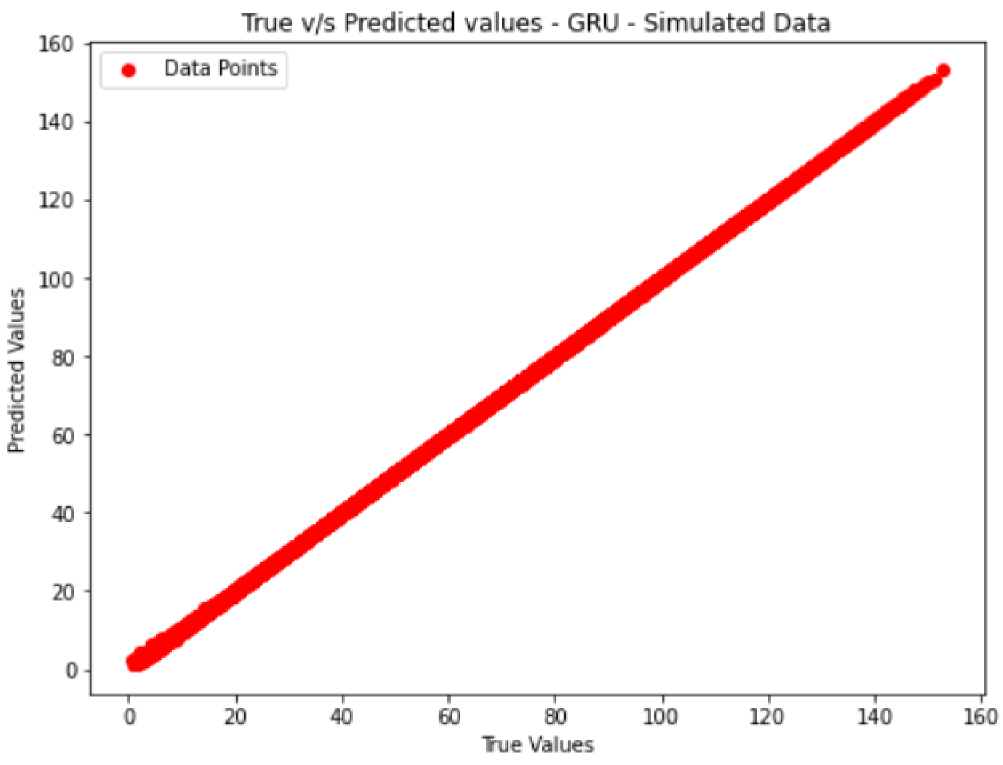


Fig: GRU True v/s Predicted values (Simulated Data)

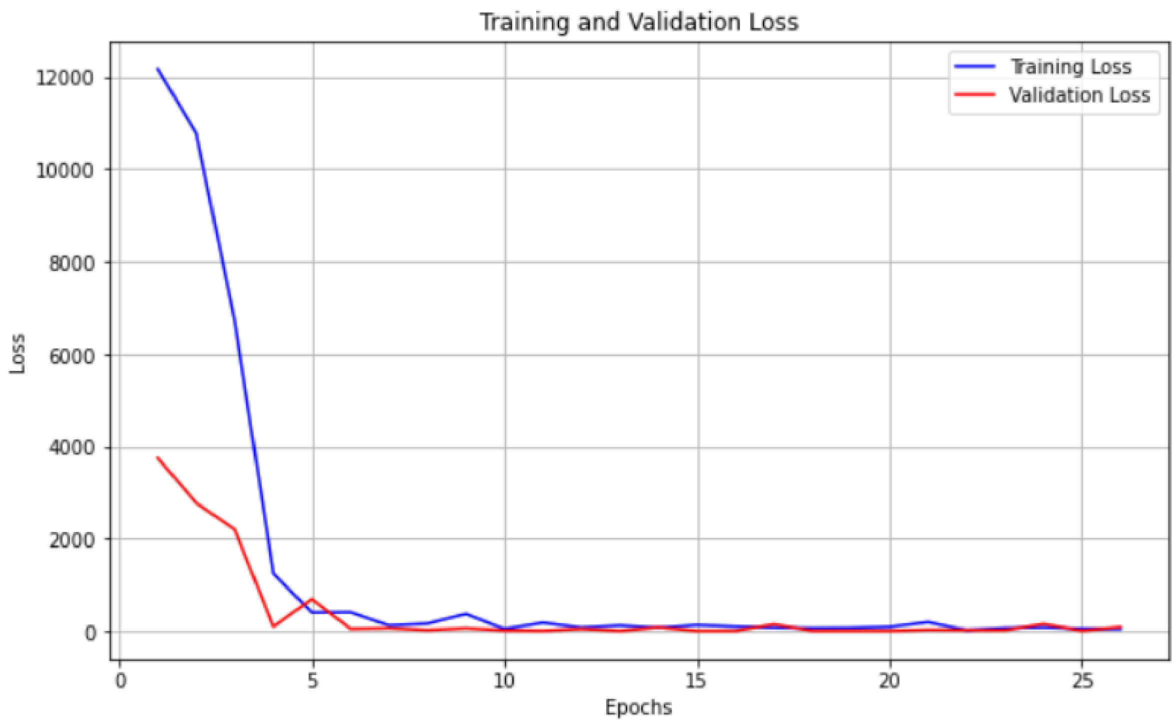


Fig: Training and Validation loss chart for GRU(S&P500 dataset) model