UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Investigating the use of machine learning to value contingent claims

By

Sriya Beharie

14049890

Supervisor: Eben Maré

Submitted in fulfilment of the requirements for

the degree in the Faculty of Natural and Agricultural Sciences

University of Pretoria

December 2024

Anti-Plagiarism Declaration

I, Sriya Beharie, declare that this dissertation, which I hereby submit for the degree of Master of Science in Financial Engineering at the University of Pretoria, is my own work and has not previously been submitted by me for a degree at this or any other tertiary institution.

This dissertation is wholeheartedly dedicated to my parents.
Your unconditional love, your sacrifices and encouragement have shaped the person I am today.
I owe you everything.

I would like to thank my fiancé for being my anchor through every challenge. Your support and belief in me have been my source of motivation.

I would also like to thank my supervisor, whose mentorship and guidance have been invaluable throughout my dissertation.

Contents

Abstract

A relevant area of finance that has gained traction in recent years is the use of machine learning methods with traditional approaches for pricing European and American options. This dissertation investigates the Cox-Ross-Rubinstein binomial model, the Black-Scholes model, and advanced neural network structures, including artificial neural networks and deep neural networks. By using the Black-Scholes model as a benchmark for European options, and the Cox-Ross-Rubinstein and Least-Squares Monte Carlo model for American options, our research aims to evaluate the accuracy and the efficiency of artificial and deep neural networks in option pricing, in constant and volatile market environments. We show that neural networks perform comparably to traditional models, offering an alternative for financial applications. Model limitations and other areas of improvement are also considered.

Keywords: American-style option, European-style option, Machine Learning, Artificial Neural Networks, Deep Neural Networks

Abbreviations

AI	**Artificial Intelligence**
ANN	**Artificial Neural Networks**
ATM	**At-the-money**
BDPP	**Backward Dynamic Programming Principle**
BM	**Brownian Motion**
BS	**Black Scholes**
CRR	**Cox-Ross-Rubinstein**
DL	**Deep Learning**
DNN	**Deep Neural Networks**
FNN	**Feed-Forward Neural Networks**
GBM	**Geometric Brownian Motion**
GD	**Gradient Descent**
HFT	**High Frequency Trading**
ITM	**In-the-money**
LS	**Least Squares**
LSMC	**Least Squares Monte Carlo**
LSM	**Least Squares Method**
LSTM	**Long Short Term Memory**
MBGD	**Multi-Batch Gradient Descent**
MC	**Monte Carlo**
ML	**Machine Learning**
MLP	**Multi-Layer Perceptron**
MSE	**Mean Squared Error**
NN	**Neural Network**
NLP	**Natural Language Processing**
OTM	**Out-of-the-money**
ReLU	**Rectified Linear Unit**
SDE	**Stochastic Differential Equation**
SGD	**Stochastic Gradient Descent**
SVM	**Support Vector Machine**
VaR	**Value at Risk**

List of Notation

- S_t: Asset price at time t
- μ: Drift coefficient indicating the expected rate of return
- σ: Volatility of the asset
- W_t: Wiener process (Standard Brownian Motion)
- r: Risk-free interest rate
- T: Maturity of the option
- K: Strike price of the option
- $V_{\text{EU call}}$: Discounted payoff of a European call option
- $V_{\text{EU put}}$: Discounted payoff of a European put option
- $V_{\text{AM call}}$: Discounted payoff of an American call option
- $V_{\text{AM put}}$: Discounted payoff of an American put option
- $P(S_t)$: Intrinsic payoff function of an option at time t
- $V(S_t)$: Value process of an option
- $g(\cdot)$: Discounted payoff function
- $\mathbb{E}[\cdot]$: Expectation operator
- ρ: Correlation between Wiener processes in the Heston model
- $h(l)$: Hidden units in layer l of a neural network
- $W(l)$: Weight matrix for layer l in a neural network
- $b(l)$: Bias vector for layer l in a neural network
- z: Output of the neural network
- $f(x;\theta)$: Nonlinear regression model

Other Mathematical Notation

- sup -The supremum of a set is the least upper bound of that set. It is the smallest value that is greater than or equal to every element in the set (Hunter, 2024).
- inf -The infimum of a set is the greatest lower bound of that set. It is the largest value that is less than or equal to every element in the set (Hunter, 2024).
- arg - The argument refers to the input values of a function that maximize or minimize the function's value (Schmidt, 2016).

Glossary of Frequently Used Terms

Artificial Intelligence (AI): Makes reference to the simulation of intelligence by machines designed to perform tasks such as decision-making, visual perception, natural language understanding, and problem-solving. AI systems aim to replicate cognitive functions such as learning and reasoning. A subset of AI, known as Machine Learning (ML), involves building algorithms that can learn and adapt from data without explicit programming, forming the backbone of modern advancements in AI.

Machine Learning (ML): A subfield of AI that focusses on creating systems capable of identifying patterns in data and making predictions or decisions. ML encompasses several techniques, including supervised, unsupervised, and reinforcement learning. These methods enable automation in tasks such as image recognition, speech processing, and predictive modeling. ML algorithms require data preprocessing (such as data transformation and data cleaning), feature selection, and iterative training to optimize model performance.

Neural Networks (NN): Drawing inspiration from the biological brain, neural networks consist of interconnected layers of nodes (neurons) that process data through weighted connections and activation functions. These models are categorized by depth: shallow networks, known as artificial neural networks (ANNs), have fewer layers and handle simpler problems, while deep neural networks (DNNs) utilize multiple layers to capture complex, hierarchical patterns in data. They are foundational to deep learning and widely applied across industries.

Deep Learning (DL): A subset of ML utilizing DNNs to learn intricate and abstract data patterns. By leveraging multiple hidden layers, deep learning models excel at processing high-dimensional data and addressing complex tasks like image recognition, natural language processing, and autonomous driving. The "depth" of these networks enables them to discover hierarchical representations of data, providing unparalleled accuracy for tasks involving large-scale datasets.

Least Squares Monte Carlo (LSMC): A hybrid method combining Monte Carlo simulations with regression analysis to value financial derivatives, particularly American options. LSMC uses simulated paths of the underlying asset to estimate continuation values, which guide the decision of whether to exercise the option early. This method balances computational efficiency with accuracy and serves as a benchmark for evaluating advanced pricing models like neural networks.

Black-Scholes (BS) Model: A cornerstone in financial modeling, the Black-Scholes model provides a closed-form solution for pricing European options. It assumes constant volatility and interest rates, log-normal asset price distributions, and frictionless markets. Despite its limitations, the BS model remains a critical benchmark for understanding derivative pricing and serves as a foundation for more advanced stochastic models like Heston.

Monte Carlo (MC) Simulation: A computational technique that depends on the repeated random sampling to approximate numerical solutions. In finance, MC simulation is used to model stochastic processes, evaluate complex derivatives, and assess risk. Its flexibility makes it suitable for pricing options under various conditions, including those with path dependency or stochastic volatility.

Supervised Learning: A machine learning technique where models are trained on labeled datasets, enabling them to map inputs to desired outputs. Common applications include

fraud detection in finance, disease diagnosis in healthcare, and customer segmentation in marketing. The success of supervised learning hinges on the quality and diversity of the labeled data, ensuring generalization to new, unseen inputs.

Unsupervised Learning: A machine learning technique that operates on unlabeled data, primarily for clustering, anomaly detection, and dimensionality reduction. Unlike supervised learning, it identifies hidden structures or patterns within data. This approach is not suited for tasks requiring labeled predictions, such as option pricing, which relies on explicit mappings between inputs and outputs.

Reinforcement Learning: A paradigm in machine learning where an agent develops optimal decision-making skills through interactions with an environment. By receiving rewards or penalties for its actions, the agent improves its strategy over time. Reinforcement learning is particularly useful in sequential decision-making tasks, such as trading strategies, but is not directly applicable to the static pricing models explored in this dissertation.

Heston Model: A stochastic volatility model that improves upon the Black-Scholes framework by incorporating a mean-reverting stochastic volatility process. It captures empirical phenomena like the implied volatility smile and skew, making it a robust tool for pricing derivatives in markets characterized by volatility dynamics.

Gradient Descent (GD): A foundational optimization algorithm for minimizing loss functions in machine learning. GD adjusts model parameters iteratively in the direction of the negative gradient, ensuring convergence to a local or global minimum. It forms the backbone of neural network training.

Stochastic Gradient Descent (SGD): A variantion of gradient descent that adjusts weights using randomly selected subsets (mini-batches) of data. This reduces computational overhead, enabling efficient training for large-scale datasets while maintaining convergence properties.

Dropout: A regularization method for neural networks whereby a subset of neurons "drops" are randomly dropped during training. By preventing co-adaptation among neurons, dropout enhances generalization and reduces the risk of overfitting, making it crucial for training deep networks.

Activation Function: A mathematical transformation that is applied to a neuron's output to introduce non-linearity, enabling complex modelling patterns in neural networks. Common activation functions include ReLU, Tanh and Sigmoid, with each being suited to specific tasks and architectures.

Chapter 1

Motivation

"AI is the new electricity"

-Andrew Ng, founder of the Google Brain

The valuation of derivatives, in particular options, is a cornerstone of quantitative finance. Traditional methods such as the Black-Scholes model, the Cox-Ross-Rubinstein binomial tree, and the Longstaff-Schwartz Least Squared Monte Carlo method are well-renowned for their simplicity and elegance, contributing to their dominance in financial markets. However, these methods often struggle with capturing the complexities of real-world market behaviour. This is attributable to the underlying assumptions upon which the traditional theories are built, namely that volatility is constant, markets are frictionless, and the lognormal price dynamics of assets. While methods such as the Least-Squares Monte Carlo method offer more flexibility in the handling of path-dependent options, it essentially relies heavily on pre-specified basis functions which, as noted by Wei & Zhu (2022), limits it's ability to capture non-linear dependencies. In additional to the assumption constraints of traditional methods, the Cox-Ross-Rubinstein model becomes computationally inefficient as the number of time steps increase as noted by Johnson (2020). In additional to this, Contreras, et al.(2015) provided research on the Black-Scholes model's struggle with extending to the multi-asset case with increased dimensionsionality. An added disadvantage of using the Least-Squares Monte Carlo method is the computational cost inefficiency of using repeated simulations and regressions as highlighted by Tian & Benkrid (2012). In a fast-paced and competitive market environment, the value of a minute saved is immeasurable. It translates directly into higher profitability and minimized risk exposure, providing a substantial competitive edge. It raises the critical question, by what means can one achieve this additional minute?

"The playing field is poised to become a lot more competitive, and businesses that don't deploy AI and data to help them innovate in everything they do will be at a disadvantage"

-Paul Daugherty, chief technology and innovation officer, Accenture

Clara Shih (2023), CEO of Salesforce AI, has asserted that we are currently experiencing an "AI and data revolution." Embracing AI, especially in quantitative finance, has proven quite beneficial. Within an intensely competitive market-making environment, the ability to rapidly quote option prices that align with constantly shifting market conditions is critical. It is noted by Anderson & Ulrych (2023) that even the slightest improvement or acceleration over traditional pricing methods can be vital in the prevention of arbitrage opportunities. With decades of research to support the value of incorporating artificial intelligence to finance, its implementation

has demonstrated improved model efficiency and facilitated the capturing of data-complexities that traditional methods often overlook. In particular, the application of neural networks to finance and specifically option pricing has proven to be a promising field with substantial research prompting its advancements.

The earliest discoveries surrounding neural networks date as far back as 1943, with McCulloch and Pitts introducing a mathematical model of how neurons behave in the brain. This model, however limited, provided the theoretical foundation for the computational abilities of neural networks. Significant contributions by Hebb (1949) and Rosenblatt (1985) propelled the development of the neural network architecture, laying the groundwork for modern advancements in the field of arificial intelligence and deep learning. The notable research by Culkin and Das (2017) explores the application of deep learning to option pricing, by training a fully connected, feed-forward neural network to replicate the option pricing formula for the Black-Scholes model with high accuracy. Similarly, Anderson and Ulrych (2023) explored the utilization of deep learning to efficiently price American-style options. Focus was placed on maintaining regulatory compliance by leveraging explainable artificial intelligence, and the neural network was trained using the PDE-based Heston stochastic volatility model. The neural network performance was measured against traditional methods such as PDE's, binomial trees, and the Least-Squares Monte Carlo algorithm, and outperformed these methods by achieving pricing near instantaneously. Whilst traditional methods are flawed, they still largely remain the basis for which accuracy is benchmarked.

This dissertation will adopt a similar approach to those outlined above, with focused attention given to the parameter choice and architecture of neural networks applied to option pricing. The aim of this disseration is to comprehensively investigate the intricate dynamics between neural networks and option pricing, striving to further advance our understanding of artificial intelligence and bring us closer to resolving the challenge of securing the critical additional minute.

1.2 Aims and Objectives

This dissertation aims to demonstrate the potential of neural networks as a viable alternative to traditional methods for option pricing.

The disseration also aims to evaluate whether the choice of input parameters significantly influences neural network performance in option pricing. This is achieved by:

- **Scaler Variation:** Comparing sensitivity graphs generated using different scalers, such as MinMax and StandardScaler.
- **Testing Dropout Layers:** Examining the impact of varying dropout layers through sensitivity analyses.
- **Ideal Parameters Scenario:** Presenting an "ideal parameters" case, combining insights from the above analyses and prior research to identify optimal parameter settings for comparing traditional Cox-Ross-Rubinstein (CRR) models with artificial and deep neural network (ANN/DNN) architectures.

The dissertation also aims to explore the performance of neural networks in specific areas, focusing on:

- **Unstable Market Environments:** Using the Heston Stochastic Volatility model, the dissertation demonstrates the robustness of neural networks in handling volatile market conditions.
- **Computational Efficiency:** Training and execution time metrics are analyzed to highlight the superior time efficiency of neural networks compared to traditional benchmark methods.

This dissertation will meet these objectives, specifically highlighting differences between the performance of the artificial 'shallow' neural network structure, with the deep neural network structure. Through these analyses, the dissertation seeks to emphasize the potential of neural networks to not only match but, in some cases exceed the capabilities of traditional option pricing approaches in efficiency.The outlined objectives are achieved through the following structure:

We begin with a comprehensive introduction of European and American option pricing, accompanied by a brief discussion of alternative traditional and established option pricing methods. This section will lay the groundwork for vanilla option pricing by use of the Black-Scholes model for European options and the Cox-Ross-Rubinstein (CRR) binomial model for American options. Additionally, a framework and algorithm for the Heston stochastic volatility model will be presented. Following this, we will introduce the concept of machine learning and its historical application in finance, particulary in the domain of Neural Networks. This integration aims to merge the fields of option pricing and machine learning, enabling an in-depth analysis of its effectiveness.

Subsequently, we construct a neural network model for option pricing. This involves simulating the relevant option pricing parameters, training the model, and rigorously testing its performance against the results of traditional 'benchmark' models. Additionally, we explore the Least Squares Monte Carlo (LSMC) approach proposed by Longstaff and Schwartz, examining its application to pricing American options. This is done with the intention of providing a further benchmark to assess the efficiency of neural networks. To enhance the understanding of the neural network model's structure, we provide a detailed exploration of its architecture, complemented by

graphical representations illustrating each stage of its design. Furthermore, the stages of neural network construction are depicted mathematically, ensuring a comprehensive understanding of its operational framework.

Finally, we conduct a thorough analysis of the model's results, emphasizing the potential and effectiveness of employing neural networks in option pricing. This includes an evaluation of the model's accuracy across different market conditions and the presentation of practical strategies to enhance its performance and its robustness.

Chapter 2

Dissertation Structure

2.1 Framework Overview

This dissertation is divided into two primary sections, designed to ensure a flow between the theoretical foundations and the practical implementation:

- **Theoretical Background Section:** This section lays the groundwork by presenting the essential concepts and mathematical frameworks that underpin this dissertation. It includes a detailed literature exploration of traditional option pricing models, neural network architectures, and the integration of machine learning into financial applications. By addressing these theoretical foundations, the section ensures that the reader is equipped with the necessary understanding to extend to the practical implementations.
- **Practical Application Section:** Building on the theoretical insights, this section focuses on the implementation and the evaluation of neural network models (artificial and deep) in the context of option pricing. It details the methodologies adopted, the training and testing processes, sensitivity analyses, and the comparison of results with traditional pricing methods. This section bridges the theoretical underpinnings from the theoretical section with real-world applications, thus demonstrating the potential of machine learning, in particular neural networks, in addressing financial challenges.

By structuring the dissertation into these two distinct sections, we were able to ensure a logical progression from foundational concepts to advanced applications. This approach allows for a comprehensive analysis that not only depicts the relevance of neural networks in financial modeling but also highlights their practical advantages and limitations. Furthermore, this division provides clarity to the reader, offering a clear distinction between the conceptual frameworks and the practical implementations, thereby enhancing the overall cohesion of the dissertation. Chapters 3 through 8 form the theoretical background component, while Chapters 9 through 13 comprise the practical application component.

Chapter 3: Introduction

This chapter provides an overview of foundational principles and context for the dissertation. It explores advancements in option pricing, beginning with traditional methods like the Cox-Ross-Rubinstein (CRR) binomial model and the Black-Scholes (BS) formula. It sets the stage for the research by connecting these traditional benchmark approaches to the implementation of machine learning techniques in financial engineering, with a focus on neural network application.

Chapter 4: Literature Review

The literature review builds on the introduction, combining prior work on option pricing methods and the incorporation of machine learning. It evaluates the evolution of neural network architectures, such as shallow ANNs and deep DNNs, in comparison to benchmark models like CRR and LSMC.

Chapter 5: Important Concepts

This chapter examines essential mathematical and financial concepts. It begins with an analysis of European and American options, followed by discussions on stochastic processes like Standard Brownian Motion and the Heston model. These concepts are foundational for understanding the implementation of machine learning in option pricing.

Chapter 6: Neural Networks

This chapter examines neural network architectures and their relevance to option pricing. It contrasts shallow ANN models with deeper DNNs, focusing on their capacity to handle complex, non-linear relationships inherent in financial markets. It also introduces training algorithms, activation functions, and hyperparameter tuning to optimize performance.

Chapter 7: Monte Carlo Techniques

Monte Carlo simulations, including the Least-Squares Monte Carlo (LSMC) method, are presented as benchmarks for neural network models. The chapter highlights the flexibility of these techniques for American option pricing and their integration with the Heston model to simulate stochastic volatility.

Chapter 8: Methodology

The methodology section provides a detailed workflow for implementing neural networks in option pricing. It outlines data preprocessing, model architecture, and training protocols. Additionally, it explains the evaluation metrics used to compare neural network performance against traditional models. We include a comparison of the ANN and DNN characteristics as per our results section, as well as the relative error metrics for the "Ideal Parameters Scenario" section.

Chapter 9: Results

The results section is divided into three key areas:

- **Constant Volatility Results**: Performance of ANN/DNN models against Black-Scholes, Cox-Ross-Rubinstein, and Least-Squares Monte Carlo models.
- **Stochastic Volatility Results**: Insights into ANN/DNN robustness under Heston stochastic volatility dynamics.
- **Time Performance**: Computational efficiency metrics for different models.

Chapter 10: Constant Volatility

This chapter centers on assessing the performance of ANN and DNN models in constant volatility scenarios, with a specific focus on sensitivity analysis and the influence of dropout layers on model predictions. It addresses two pivotal objectives of the dissertation: showcasing the potential of neural networks as a credible alternative to traditional benchmark option pricing methods, and analyzing the impact of input parameter selection on the performance of neural networks in option pricing.

Chapter 11: Stochastic Volatility

This chapter analyzes the performance of ANN and DNN models within stochastic volatility environments, focusing on their robustness and accuracy in comparison to LSMC. The primary objective of this section is to evaluate the resilience of neural networks in navigating volatile market conditions effectively.

Chapter 12: Time Performance

This section explores the time efficiency of neural networks, highlighting their advantages in computational speed and scalability in comparison to traditional methods. It addresses a key objective of the dissertation: determining whether neural network implementations offer superior computational efficiency over conventional benchmark approaches.

Chapter 13: Conclusion

The concluding chapter consolidates the findings, validating the potential of neural networks in option pricing. It discusses how the objectives of this dissertation were met through theoretical and practical implementation. We also discuss practical implications, challenges, and avenues for future research, including advanced architectures like LSTM-GRU.

Our dissertation framework is carefully structured, aligning each chapter with a specific focus to ensure a coherent and logical progression throughout. The framework is visually summarized in the Figure 2.1 below, which provides a clear representation of the dissertation's flow and major components.

The theoretical background section, encompassing Chapters 3 to 8, is divided into three key components: the Introduction, the Central Argument, and the Methodology. Each of these sections plays a critical role in establishing the foundation for this dissertation. The Introduction provides the necessary background, a literature review, and a discussion of critical concepts, including European and American options, Heston stochastic volatility, and machine learning techniques. The Central Argument delves into the neural network architecture and the role of Monte Carlo techniques, such as Least-Squares Monte Carlo (LSMC) and its application to the Heston stochastic volatility model, forming the theoretical core of the dissertation. The Methodology is presented as the bridge to the practical application, and it encompasses diagrammatic representations, Python implementation, scalers, dropout configurations, and comparisons between artificial neural networks (ANN) and deep neural networks (DNN).

The practical application section builds directly on the theoretical groundwork. It begins with the Results, which are subdivided into three primary analyses: stochastic volatility, constant

volatility, and time efficiency. These analyses leverage the methodology to demonstrate the performance of neural networks under varying conditions. The findings are then synthesized in the Conclusion, which ensures that the dissertation objectives are fulfilled and provides a summary of the research outcomes.

Finally, the framework addresses the Shortcomings and Other Areas of Research, drawing insights not only from the practical application but also from the central argument. This approach ensures a comprehensive and forward-thinking discussion, emphasizing potential enhancements in neural network architecture and identifying promising directions for future research. The diagrammatic representation effectively captures this interconnected narrative, emphasizing how our dissertation's components are integrated to provide an extensive exploration of the research topic. Figure 2.1 below illustrates the diagrammatic representation of the structure.

2.2 Diagrammatic Framework

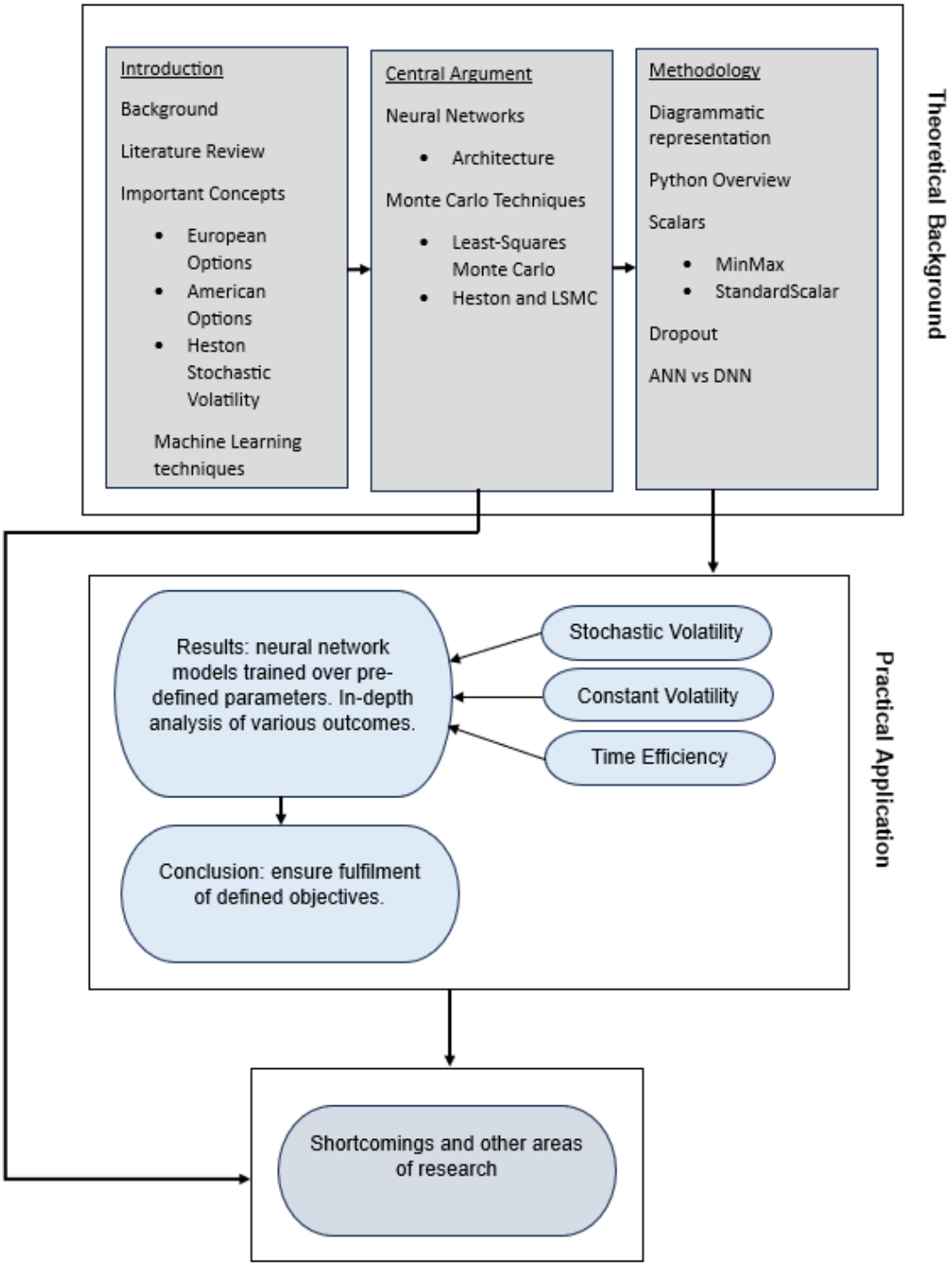


Figure 2.1: Diagrammatic Representation

Theoretical Component

Chapter 3

Introduction

3.1 Background

Over time, significant advancements have been made in improving both the efficiency and accuracy of option pricing, particularly for American-style options in the field of financial engineering. Progress has been evident in approximating American option prices, beginning with the introduction of the binomial method (Cox et al., 1979), followed by the finite difference method (Barone-Adesi and Whaley, 1987), and later the implementation of Monte Carlo simulation techniques (Boyle et al., 1996). Of particular note is the Least-Squares Monte Carlo (LSMC) simulation proposed by Longstaff and Schwartz (2001), which is analyzed in depth in this dissertation. Additionally, the dissertation will explore the pricing of European options using established techniques, such as the Black-Scholes pricing algorithm introduced by Black & Scholes (1973).

Parallel to this, there has been remarkable progress in the application of machine learning techniques to the pricing of options. From the earliest development of the perceptron model by McCulloch & Pitts (1943) and Rosenblatt (1958) to financial application seen today. This includes modern derivative pricing approaches such as those by Culkin and Das (2017) which have demonstrated the increasing relevance of neural networks in addressing the complexities of American option pricing. Building on these advancements, this dissertation considers the implementation of artificial neural networks (ANNs) and deep neural networks (DNNs) to price both American and European options effectively.

While pricing European options is relatively straightforward due to their lack of early exercise features, American-style derivatives present additional challenges. These complexities arise because the holder of an American option may exercise the option at any time, necessitating a balance between immediate payoff and the expected discounted payoff from continuation. As highlighted by Longstaff and Schwartz (2001), this requires the determination of an optimal exercise strategy, making advancements in American option pricing instrumental in financial markets. Enhancing the accuracy and efficiency of pricing methods for these options is a task of substantial practical significance. For European option pricing, this dissertation focuses on the Black-Scholes closed-form method. European options, which can only be exercised at maturity, provide a simpler framework compared to American options. However, these methods are essential benchmarks for understanding more complex pricing scenarios.

The Cox-Ross-Rubinstein (CRR) method, introduced by Cox, Ross, and Rubinstein (1979), rep-

resents a pivotal advancement in the field of option pricing. The CRR model utilizes a binomial tree framework to discretize the price movements of the underlying asset over time, allowing for the evaluation of both European and American options. Its appeal lies in its flexibility and simplicity, thus making it a cornerstone method in the evolution of option pricing methodologies. Over the years, the CRR model has been widely adopted as a benchmark due to its ability to approximate continuous models, such as the Black-Scholes framework, with increasing accuracy as the number of time steps increases. Thorough research has been conducted on this by Leisen (1996), Broadie and Detemple (1996), and Jiang and Dai (2004). This dissertation utilizes the CRR method as a benchmark for comparison with neural network models, establishing its validity through convergence properties and its capacity to handle early exercise features in American options. The CRR method's role as a benchmark is particularly crucial for analyzing the robustness and computational efficiency of alternative approaches, such as ANN and DNN architectures, and in this instance, with regard to option pricing tasks.

Longstaff and Schwartz (2001) introduced an innovative and effective approach for pricing American options using the LSMC method. This approach was devised in response to inefficiencies in prior methods, such as computational inefficiency, excessive time consumption and high costs, as noted by Barone-Adesi and Whaley (1987). In contrast to traditional simulation techniques, the LSMC method employs regression techniques to approximate continuation values, effectively reducing computational complexity. Additionally, the LSMC method allows for comparisons with traditional methods once convergence is depicted and discussed, making it a robust tool for analyzing American option pricing.

The Heston stochastic volatility model, introduced by Heston (1993), provided a significant advancement in the modeling of financial markets. Unlike simpler models such as Black-Scholes, where constant volatility is assumed, the Heston model incorporates a stochastic process for volatility thus capturing the dynamic nature of market conditions. This makes the Heston model particularly effective for pricing options in markets where volatility exhibits mean-reverting behavior and is influenced by random shocks. The model's ability to account for the empirical features of financial markets makes it a robust choice for sophisticated pricing tasks. This can be seen more recently in Anderson & Ulrych (2023), where they demonstrated the effectiveness of leveraging advanced stochastic models, such as the Heston model, in conjunction with deep learning to analyze the option pricing accuracy and computational efficiency. In this dissertation, the Heston model serves as both a theoretical and computational foundation for generating synthetic training data for neural networks. The simulated paths from the Heston model provide realistic market scenarios, which enables the evaluation of neural network performance in capturing complex stochastic behaviour. In this dissertation, the Heston stochastic volatility model is implemented alongside the Least-Squares Monte Carlo (LSMC) method to simulate option price paths, providing a robust framework for the analysis of the performance of artificial and deep neural networks in pricing options under volatile market conditions. By using the Heston model, we aim to assess the potential of artificial and deep neural networks in replicating and surpassing traditional pricing methods under stochastic volatility conditions. Additionally, the Heston model is used as a comparative benchmark to highlight the effectiveness of neural networks in pricing options with complex underlying dynamics, particularly in volatile market environments.

It is important to note, deep learning black-box models in financial applications often lack interpretability, making it challenging to justify the reasoning behind predictions, as noted by Rudin (2019). This lack of transparency can hinder the adoption of such models in domains like option pricing, where transparency and explainability are crucial. Anderson and Ulrych (2023) highlight the growing concern around the use of black-box methods, emphasizing the importance

of developing interpretable frameworks in financial modeling. Similar to the approach taken by Anderson and Ulrych, this dissertation's application of neural networks is structured to provide insights into the complete pricing process by incorporating domain-specific parameters. This includes those derived from the Heston stochastic volatility model, and the parameters used for the constant volatility models. By integrating known financial principles the neural networks employed in this dissertation move beyond a black-box model, offering a more transparent and interpretable tool for option pricing. Our approach ensures that neural networks not only deliver accuracy but also align with established theoretical benchmark frameworks, thus closing the gap between computational efficiency and practical usability.

This dissertation leverages the aforementioned advancements in both traditional option pricing models and machine learning methods, specifically with respect to deep learning, to assess their relative performances and explore the potential of using neural networks as viable alternatives.

Chapter 4

Literature Review

4.1 American and European Options

Determining the value of an option has been a core focus in financial mathematics for decades. Black & Scholes (1973) introduced a closed-form framework for pricing European options based on an underlying asset that follows a geometric brownian motion with constant drift and volatility. Under the risk-neutral framework, the asset price follows a log-normal distribution. This method provides a significantly more straightforward process for European options compared to American options, which involve additional complexities due to early exercise possibilities.

For American options, pricing relies on identifying an optimal stopping boundary, as the holder can exercise the option prior to maturity. The concept of optimal stopping is succinctly defined by Pu (2021) as "the problem of choosing a time to adopt an action based on sequentially observed random variables so as to maximize an expected payoff or minimize an expected loss". The finite difference method introduced by Brennan & Schwartz (1977) played a pivotal role in pricing American put options. This approach utilizes a mesh of grid points and solves the partial differential equations governing option prices. Variants of this method include implicit, explicit, or Crank-Nicholson schemes, which provide flexibility in handling numerical approximations, as noted by Moreno & Navas (2003).

In 1998, Zvan, Forsyth, and Vetzal proposed a finite element method (FEM) for solving the Black-Scholes partial differential equation for American options. This method introduces a nonlinear penalty term to transform the PDE and improve convergence efficiency. More recently, Ballestra (2018) explored numerical schemes using Richardson extrapolation to enhance accuracy while minimizing computational costs. According to Pu (2021) these methods are typically constrained to lower-dimensional problems, such as those up to four dimensions[1].

The binomial model introduced by Cox, Ross, and Rubinstein (1979) represents another cornerstone in option pricing methodologies. This lattice-based approach discretizes the risk-neutral framework, employs backward induction, and calculates the optimal strategy at each node. Cox et al. (1979) emphasized its computational efficiency and simplicity, making it an effective benchmark for validating more advanced numerical and analytical methods. Moreno and Navas (2003) note that this method provides a foundation for comparing modern innovations in pricing techniques.

[1] Multi-asset options add more dimensions, which increases complexity

By use of the binomial tree pricing method or finite-difference methods we can reach the correct solutions for pricing American options. However, Barone-Adesi & Whaley (1987) notes that applying these methods are often cost ineffective and time consuming. As a result, research was conducted on alternative methods for the pricing of American derivative and in 1986, Boyle proposed was an extension of the binomial tree to the trinomial tree method.

Even with the more attractive approximation methods that have relatively inexpensive computations and are as a result more efficient than using numerical methods, there are still high costs involved when estimating higher order multivariate density functions. This is seen with the heuristic method proposed by Geske and Johnson (1985).

Using the consistent assumptions as stated by Black & Scholes (1973) and Merton (1973), Giovanni Barone-Adesi and Robert Whaley (1987) were able to find a "quadratic" approximation which is suitable in the valuation of American-style commodity options and commodity futures options. The alternative method they proposed was considered to be a more "accurate" and "inexpensive" way of pricing American call and put options written on underlying commodities.

Barone-Adesi and Whaley (1987) compared the finite-difference method, the binomial tree pricing method, the Johnson (heuristic) method and the quadratic approximation method and were able to use their findings to conclude that the quadratic approximation method was an ideal choice with respect to accuracy and cost-effectiveness in pricing options for a specific time to maturity of less than a year. The quadratic approximation method, although not as fast as the heuristic method proposed by Geske and Johnson, is significantly more efficient and more accurate in comparison. For time to maturity extending over a year, the finite-difference technique or binomial tree pricing technique are more accurate.

In 1993, James A. Tilley introduced an algorithm to price American options using a path simulation model. The precision of the solutions was demonstrated through the use of a put option on a non-dividend-paying stock, thereby determining the appropriate premium. The research examined the estimator mean, standard deviation, and biases associated with the approach. Simulated paths of the underlying stock or asset were drawn from an arbitrage-free distribution, and the prices of put and call options were subsequently estimated.

The methodology proposed by Tilley (1993) involved defining the options at each path through an indicator variable, where a value of 1 signified the exercise of the option and 0 indicated the option being held. A sequence of binary decisions (0's and 1's) was applied to evaluate whether to exercise or hold the option at any given time t. The cash flows from the stock were then discounted along these paths, and an average was computed across all paths. Notably, biases arising from the finite sample of paths became apparent. However, as the sample size of paths approached infinity, the biases diminished, thereby converging to the exact option value.

Tilley's research provides a method to approximate the exact value of an option by estimating the exercise-hold decision boundaries at each time step. This process maximizes the value obtained through the premium estimator equation. With an infinite sample space comprising asset price paths, the algorithm produces the exact price. Conversely, with finite asset path samples, the solution approximates the true value.

A key objective of Tilley's research was to challenge the prevailing belief that American options cannot be accurately and efficiently valued using simulation-based models. Tilley's findings indicate that such methods can indeed be effective, even for derivatives in general. While the research acknowledges certain limitations, such as complications in estimating the exercise-hold decision

boundaries in multidimensional models, Tilley asserts that these challenges can be addressed by refining the algorithm.

The historical utilization of Monte Carlo simulation techniques is summarized in the following table:

Reference	**Methodology/Technique**	**Advantages/Drawbacks**
Boyle (1977)	Introduced Monte Carlo simulation for option pricing, generating returns on the underlying asset under the assumption of a risk-neutral process.	**Advantages:** Versatile for complex payoff structures. **Drawbacks:** Standard error inversely proportional to the square root of the number of simulations, i.e., $\mathcal{O}(1/\sqrt{n})$.
Boyle, Broadie, and Glasserman (1996)	Improved efficiency of Monte Carlo simulations by introducing Quasi-Monte Carlo methods using deterministic low-discrepancy sequences.	**Advantages:** Improved efficiency and precision compared to standard Monte Carlo techniques.
Hammersley and Handscomb (1964)	Proposed variance reduction techniques such as antithetic variates and control variates to improve the precision of Monte Carlo simulations.	**Advantages:** Significant reduction in variance and improvement in simulation efficiency.
Duffie (1996)	Modeled options and underlying state variables as a continuous-time stochastic process under the no-arbitrage principle. Suggested taking the expected discounted payoff as the price.	**Advantages:** Provides a theoretical framework for generic derivative securities.
Boyle, Broadie, and Glasserman (1996)	Discussed advanced variance reduction techniques for standard Monte Carlo simulations.	**Advantages:** Improved computational efficiency.
Cheyette (1992)	Proposed Quasi-Monte Carlo simulation by using deterministic low-discrepancy sequences to increase efficiency in Monte Carlo methods.	**Advantages:** Enhanced computational efficiency and accuracy.

Table 4.1: Literature Review of Monte Carlo Simulation Techniques in Option Pricing

In 2001, Longstaff and Schwartz introduced a Least-Squares Monte Carlo (LSM) algorithm to value American-style options, a significant advancement in the field. This innovative approach utilizes least-squares regression to approximate the conditional expected payoff for option holders, thereby facilitating decisions on whether to exercise or continue holding the option. Notably, the LSM algorithm addresses limitations of traditional finite difference and binomial methods, which were often computationally inefficient in multifactor scenarios. Furthermore, the method's compatibility with parallel computing enhances its computational speed and accuracy, especially as it focuses solely on in-the-money paths. The approach by Longstaff and Schwartz (2001) operates by comparing the immediate payoff from exercising the option with the expected discounted payoff from holding it. This comparison is made at every exercise timepoint, allowing

the optimal decision to be based on the highest payoffs observed. The conditional expectation of these payoffs is determined through cross-sectional information derived from the least-squares regression. By fitting values across all paths and evaluating the outcomes, the algorithm effectively identifies the optimal exercise strategy for each path.

To demonstrate the practicality of their methodology, Longstaff and Schwartz (2001) provided several examples. They first valued an American put option within a single-factor framework, then extended the analysis to a multifactor context, including path-independent and path-dependent scenarios. These examples included complex instruments such as an American-Bermuda-Asian option, a cancelable index amortizing swap, and others. Their work also drew from existing literature, including research by Ibanez and Zapatero (1998). However, the unique contribution of Longstaff and Schwartz lies in their novel approach to conditional expectation using regression, which has since become a benchmark for pricing American-style options.

Additionally, Longstaff and Schwartz highlighted the role of basis functions in ensuring the convergence of their method. They emphasized the importance of selecting appropriate basis functions for accurate results and noted that this choice offers scope for further research to refine the algorithm's effectiveness.

In 2002, Clément, Lamberton, and Protter demonstrated that the estimated conditional expectation progressively converges to the true conditional expectation as the number of basis functions approaches infinity. They also established that the error is asymptotically normalized through the application of the Central Limit Theorem, linking it to the convergence rate of Monte Carlo simulation.

Building on this, Moreno and Navas (2003) evaluated the robustness of the Least-Squares Monte Carlo (LSM) method for pricing American-style options. They analyzed the influence of both the choice of basis function and the number of basis functions. Their research explored orthonormal basis functions, including Chebyshev polynomials (first kind and second kind), Hermite polynomials, Laguerre polynomials, Legendre polynomials, and Power polynomials.

4.2 Neural Networks

Artificial neural networks are computational models inspired by the human brain, consisting of interconnected nodes (neurons) organized into layers. These networks are capable of learning complex patterns from data through training and can be applied across various fields, including finance, to enhance efficiency and accuracy.

The concept of neural networks originated with the introduction of the perceptron, regarded as the earliest neural network model. This innovation was made possible through the pioneering work of McCulloch & Pitts(1943), Rosenblatt (1958) and Hebb (1949), who integrated ideas of brain interaction and machine learning. The perceptron model was designed with input nodes connected to an output node, illustrating how neural networks could process information and learn from data. This foundational model is explored in greater depth later in this dissertation.

Building on the perceptron, the concept of multiple-layered perceptrons emerged, forming the basis for what is now referred to as deep neural networks (DNNs). As noted by Wrigglesworth (2021) these advanced networks extend the capabilities of the perceptron by incorporating addi-

tional layers enabling the processing of more complex data structures and relationships.

In 1994, Hutchinson et al use the concept of a learning network to price futures options under the Black-Scholes pricing framework. This piqued the interest of many others to use artificial neural networks for option pricing as well as hedging. When the models are trained upfront (feeding the network with data and adjusting parameters to minimize prediction error), it offers an advantage in computational time as opposed to when the approximators are used (statistical techniques are used to approximate functions or predict outcomes). It is noted by Horvath, et al. (2019) that it also provide the benefit of being able to approximate very complex functions, as is stated in the universal approximation theorem.

Extensive research has been conducted on pricing European options using artificial neural networks (ANNs), such as the work by Liu et al. (2019), which combines traditional methods like the Black-Scholes model and the Heston model with the added capabilities of neural networks. Similarly, research on pricing the more complex American options has employed machine learning techniques to enhance performance and calibration. For instance, Jang and Lee (2019) utilized generative Bayesian neural network models to predict American option prices. Pu (2021) notes that their approach was distinct because it treated the network weights as distributions instead of fixed values, enabling robust modeling of uncertainties and improving the network's ability to generalize from training data.

In contrast to Jang and Lee's (2019) focus on the S&P 100 index, Gaspar et al. (2019) investigated pricing American put options using Bloomberg data. Their findings suggested that models incorporating additional input variables, such as dividend yield and interest rate, outperformed simpler neural network models. These simpler models used input variables like stock price, strike price, implied volatility, and maturity, combined with the Least-Squares Monte Carlo method. Unlike Gaspar et al. (2019), who relied on an artificial neural network (ANN) without observed market option prices, Karatas et al. (2019) used a model-based approach. Their method generated artificial sample paths for the price processes of American options using the Ju-Zhong approximation under the geometric Brownian motion (GBM) framework. This approach employed finite difference methods and iterative procedures to numerically solve the partial differential equations governing option prices, which Pu (2021) notes was effective in striking a balance between computational efficiency and accuracy to produce stable and convergent results. Karatas, et al. (2019) further demonstrated that recurrent neural networks (RNNs) offered more promising training times compared to feedforward neural networks (FNNs).

In the context of supervised learning, Pu (2021) notes that numerous researchers have focused on solving partial differential equations (PDEs) using ANNs, instead of relying on traditional numerical techniques, which are often challenged by dimensionality issues. For example, Han et al. (2018) transformed the PDEs into backward stochastic differential equations, calculating the gradient of the solution through deep neural networks (DNNs). Chen and Wan (2021) extended these methods by applying neural networks to American option pricing, utilizing the least squares residual of the PDE as the training loss function. Salvador et al. (2020) employed ANNs to approximate solutions to the reformulated Black-Scholes framework for American options. Their approach was reformulated into a variational inequality, addressing early exercise features for American options. This reformulated version relied on incorporating the free boundary of the American option into the training model, making it ideal for application to neural networks. The solution was further minimized to improve its convergence and accuracy.

Culkin & Das (2017) notes that the use of Artificial Intelligence (AI) in finance was initially

not explored extensively due to the lack of computing power. However, use of AI in finance is becoming more prevalent as seen in a range of areas, such as derivative pricing, credit default prediction as researched by Wrigglesworth (2021), and with Option pricing using supervised vs unsupervised learning researched by Pu (2021) and using the (Heston) stochastic volatility model as researched by Anderson & Ulrych (2023).

We will go on to explore the use of artificial neural networks (ANNs) and deep neural networks with respect to option pricing in the following dissertation.

Chapter 5

Introduction to Important Concepts

To demonstrate the potential of neural networks as a credible alternative to traditional option pricing methods, we will comprehensively define and discuss the foundational concepts pertaining to vanilla options. This will include an exploration of their pricing mechanics, theoretical underpinnings and including a brief review of their practical applications. This will serve as a basis for contextualizing the incorporation of machine learning techniques.

5.1 Option Types

According to John C. Hull (2015) in his widely acclaimed textbook *Options, Futures, and Other Derivatives*, a derivative is defined as:

> *" A derivative can be defined as a financial instrument whose value depends on (or derives from) the values of other, more basic, underlying variables. "*

Derivatives play a crucial role in finance, serving as essential tools for hedging, speculation, and arbitrage. They enable investors, financial institutions, and corporations to effectively manage risk exposure and implement elegant trading strategies tailored to specific market conditions.

Hull identifies four primary types of derivatives:

1. Futures
2. Forwards
3. Swaps
4. Options

Among these, options are particularly versatile, granting the holder the right—but not the obligation—to buy (call option) or sell (put option) an underlying asset at a predetermined strike price on or before a specified expiration date.

This dissertation focuses on the pricing and analysis of European and American options, both of which are widely utilized in financial markets, and the incorporation of machine learning techniques. For the purposes of this dissertation, we operate under the assumption of a constant, risk-free interest rate unless explicitly stated otherwise.

Definition 1: European Option - (Shreve, 2004)
A European option grants the holder the right, but not the obligation, to either purchase (call option) or sell (put option) the underlying asset at a specified strike price K, at a maturity T. Incorporating the above formula with the interest rate r, the discounted payoff for a European option is expressed as:

$$V_0^{EUcall} = e^{-rT}(S_T - K, 0)^+ \tag{5.1}$$

$$V_0^{EUput} = e^{-rT}(K - S_T, 0)^+ \tag{5.2}$$

Definition 2: American Option - (Shreve, 2004)
An American option grants the holder the right, but not the obligation, to purchase (call option) or sell (put option) the underlying asset at any time up to or on the maturity date, at a specified strike price K.
Incorporating the above formula with the interest rate r, the discounted payoff of an American option is expressed as:

$$V_0^{AMcall} = e^{-rt^*}(S_{t^*} - K, 0)^+ \tag{5.3}$$

$$V_0^{AMput} = e^{-rt^*}(K - S_{t^*}, 0)^+ \tag{5.4}$$

where $t^* \in [0, T]$ represents the optimal exercise time

5.2 Standard Brownian Motion

The incorporation of Brownian motion and standard Brownian motion is essential as they form the mathematical foundation for modeling stochastic processes, which form the basis of modern option pricing methodologies. These concepts underpin the Black-Scholes framework, the Cox-Ross-Rubinstein model, and Monte Carlo simulations, all of which assume the random evolution of asset prices and are covered extensively in this dissertation. By introduction of these processes, we aim to establish a comprehensive theoretical basis that provides the necessary context for applying machine learning techniques.

The process W_t is a Wiener process depicting standard Brownian motion. Drawing upon the work by Ross (1995) and Øksendal (2003), the following properties are established:

1. **Initial Value:** $W_0 = 0$.
2. **Independent Increments:** The change in W_t over non-overlapping intervals is independent.
3. **Stationary Increments:** The distribution of $W_t - W_s$ depends only on $t - s$, not on s or t individually.

4. **Normal Distribution of Increments:** For $0 \leq s < t$,

$$W_t - W_s \sim \mathcal{N}(0, t-s),$$

meaning the increments are normally distributed with mean 0 and variance equal to the time difference $t - s$.

5. **Continuous Paths:** W_t exhibits paths that are almost surely continuous, yet almost surely lack differentiability at any point.

6. **Markov Property:** the 'memoryless nature' implies that the future state of a process depends only on its current state, not on the sequence of events that preceded it.

7. **Martingale Property:** W_t is a martingale, such that

$$\mathbb{E}[W_t \mid \mathcal{F}_s] = W_s, \quad \text{for all } 0 \leq s \leq t. \tag{5.5}$$

Where:

- $\mathcal{F}_s$ is the **natural filtration**, representing all the information available up to time s.
- $\mathbb{E}[\cdot]$ denotes the **expected value**.

The graph below illustrates a one-dimensional Brownian motion, modeled between $T = 0$ and $T = 1$ and $N = 1000$ time steps, presented solely for visualization purposes.

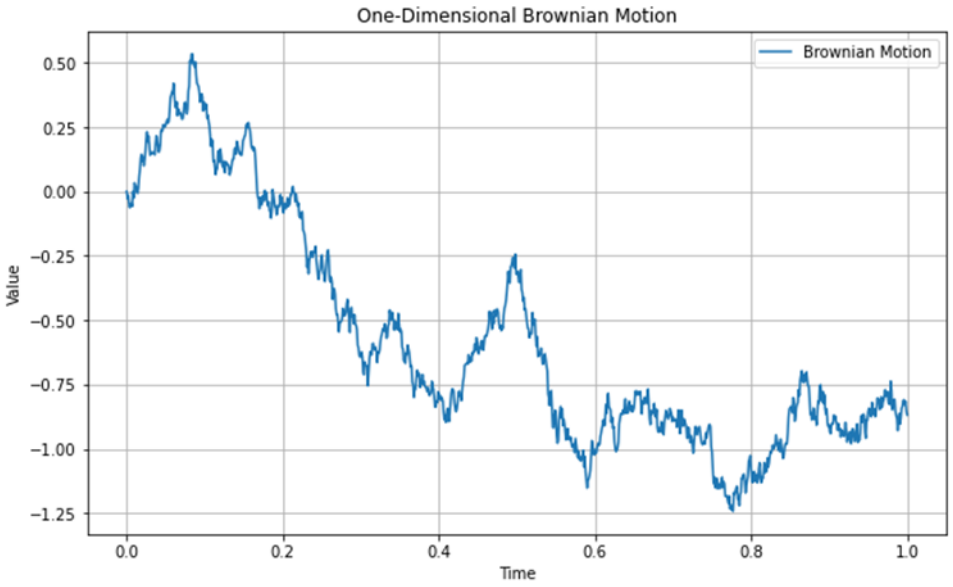


Figure 5.1: One-dimensional Brownian motion

5.3 Geometric Brownian Motion

Drawing from the work by Ross (1995), Øksendal (2003) and Baxter & Rennie (1996), the Geometric Brownian motion and its characteristics are defined next. Geometric Brownian Motion

(GBM) is a stochastic process used to model the dynamics of financial assets, such as stock prices. It satisfies the following Stochastic Differential Equation (SDE):

$$dS_t = \mu S_t\, dt + \sigma S_t\, dW_t, \tag{5.6}$$

where:

- $\mu \in \mathbb{R}$: Drift coefficient (expected return),
- $\sigma > 0$: Volatility,
- W_t: Standard Brownian motion.
- S_t is the asset price at time t

Note the following characteristics of Geometric Brownian motion (GBM):

1. **Lognormal Distribution**: The solution to the SDE implies that S_t follows a lognormal distribution:
$$S_t = S_0 e^{\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t}, \tag{5.7}$$
where $S_0 > 0$ is the initial value. As a result, $\ln(S_t)$ is normally distributed.
2. **Risk-Neutral Pricing**: In finance, GBM is adapted for no-arbitrage pricing by setting $\mu = r$, with r the risk-free rate.

5.4 The Heston Model

We define the Heston model as it provides a suitable framework for capturing the stochastic nature of asset price volatility. This is a crucial objective in this dissertation as it provides a realistic approach to the modelling of complex financial markets. Its incorporation into option pricing using neural networks allows us to test the robustness of machine learning techniques in its approach to volatile market conditions.

The Heston model is a stochastic volatility model introduced by Steven Heston (1993). It addresses the limitations of the Black-Scholes model by incorporating stochastic volatility, enabling it to capture market phenomena such as the implied volatility smile and skew. We will outline the dynamics of the underlying asset price (S_t) and its variance (v_t) drawing from the research by Heston (1993), Korn et al (2010), Anderson & Ulrych (2023).

Stochastic Differential Equations (SDEs)

1. Asset Price Dynamics:

$$dS_t = \mu S_t\, dt + \sqrt{v_t} S_t\, dW_t^1, \tag{5.8}$$

where:

- S_t: Asset price at time t,
- $\mu \in \mathbb{R}$: Drift rate (expected return),
- v_t: Stochastic variance,
- W_t^1: Standard Brownian motion determining the asset price.

2. Variance Dynamics:

$$dv_t = \kappa(\theta - v_t)\,dt + \sigma\sqrt{v_t}\,dW_t^2, \tag{5.9}$$

where:

- $\kappa > 0$: Speed of mean reversion (v_t reversion rate to mean),
- $\theta > 0$: Long-term mean variance,
- $\sigma > 0$: Volatility of volatility (randomness in variance),
- v_t: Non-negative variance at time t,
- W_t^2: Standard Brownian motion driving the variance process.

The volatility follows an Ornstein-Uhlenbeck[1] process and we use Ito's lemma[2] to get to the square-root process form, defined by Cox, Ingersoll & Ross (1985), above.

3. Correlation Between Brownian Motions:

$$dW_t^1 \cdot dW_t^2 = \rho\,dt, \tag{5.10}$$

where ρ is the correlation coefficient. Negative ρ implies that an increase in the asset price are often accompanied by a corresponding decreases in the volatility.

Euler Discretization of the Heston Model

Euler discretization[3], as discussed by Korn et al (2010), approximates the continuous SDEs with discrete time steps Δt. For the Heston model:

1. Discretized Asset Price:

$$S_{t+\Delta t} = S_t + \mu S_t \Delta t + \sqrt{v_t} S_t \Delta W_t^1, \tag{5.11}$$

where $\Delta W_t^1 \sim \mathcal{N}(0, \Delta t)$ (normal distribution with mean 0 and variance Δt).

2. Discretized Variance:

$$v_{t+\Delta t} = v_t + \kappa(\theta - v_t)\Delta t + \sigma\sqrt{v_t}\Delta W_t^2, \tag{5.12}$$

where $\Delta W_t^2 \sim \mathcal{N}(0, \Delta t)$.

3. Correlation Adjustment:

$$\Delta W_t^2 = \rho \Delta W_t^1 + \sqrt{1-\rho^2} Z, \tag{5.13}$$

where $Z \sim \mathcal{N}(0, \Delta t)$ is independent of ΔW_t^1.

The variance process v_t follows a square-root Ornstein-Uhlenbeck process, ensuring non-negativity and mean reversion to θ. It captures the implied volatility smile and skew observed in options

[1] Ornstein-Uhlenbeck used by Stein and Stein (1991)
[2] Ito's Lemma defined in the Appendix
[3] This scheme may give rise to negative variance. This can be dealt with by the absorption or reflection approach. This can be alleviated by using higher order schemes such as Milstein. For more information on this, see the book: The Volatility surface: A Practitioners guide by Jim Gatheral.

markets[4], which are not explained by constant volatility models like Black-Scholes. This ensures that the Heston framework provide a more accurate representation of financial markets. By simulating price paths with mean-reverting and correlated stochastic volatility, the Heston model provides a realistic dataset for training neural networks. This will ensure the effectiveness of neural network implementation in scenarios where traditional models might struggle.

[4]Discussed by Weron & Wystup (2005)

Algorithm 1: Path Simulation of the Underlying in the Heston model - (Korn, et al., 2010)

1. **Initialize Volatility and Price Process:** Define the model parameters:
   - S_0: Initial asset price,
   - v_0: Initial variance,
   - μ: Drift rate (expected return),
   - κ: Speed of mean reversion,
   - θ: Long-term mean variance,
   - σ: Volatility of volatility,
   - ρ: Correlation coefficient,
   - T: Maturity of option (total simulation time),
   - N: Number of time steps,
   - $\Delta t = T/N$: Time step size.
2. **Discretize the SDEs:** Use the Euler discretization method for both asset price and variance SDEs:
   - Variance dynamics:
   $$v_{t+\Delta t} = v_t + \kappa(\theta - v_t)\Delta t + \sigma\sqrt{\max(v_t, 0)}\Delta W_t^2,$$
   where $\Delta W_t^2 \sim \mathcal{N}(0, \Delta t)$, and $\max(v_t, 0)$ ensures non-negativity[5].
   - Asset price dynamics:
   $$S_{t+\Delta t} = S_t + \mu S_t \Delta t + \sqrt{\max(v_t, 0)} S_t \Delta W_t^1,$$
   where $\Delta W_t^1 \sim \mathcal{N}(0, \Delta t)$.
   - Correlation adjustment:
   $$\Delta W_t^2 = \rho \Delta W_t^1 + \sqrt{1 - \rho^2} Z,$$
   where $Z \sim \mathcal{N}(0, \Delta t)$ is independent of ΔW_t^1.
3. **Simulate the Paths:**
   - Initialize S_0 and v_0 as the starting values for the asset price and variance.
   - For each time step, compute $v_{t+\Delta t}$ and $S_{t+\Delta t}$ iteratively using the discretized equations.
   - Repeat the process for multiple paths (Monte Carlo simulation).
4. **Aggregate Results:**
   - Store the simulated paths for S_t at each time step.

[5]This is the absorption approach, as discussed in the footnote in the previous page

- Compute option prices, risk measures, or volatility metrics from the simulated paths.
- Average the paths at maturity for Monte Carlo-based option pricing.

The graphs below illustrate the simulated variance paths and simulated asset paths using the Heston model. It is presented solely for visualization purposes.

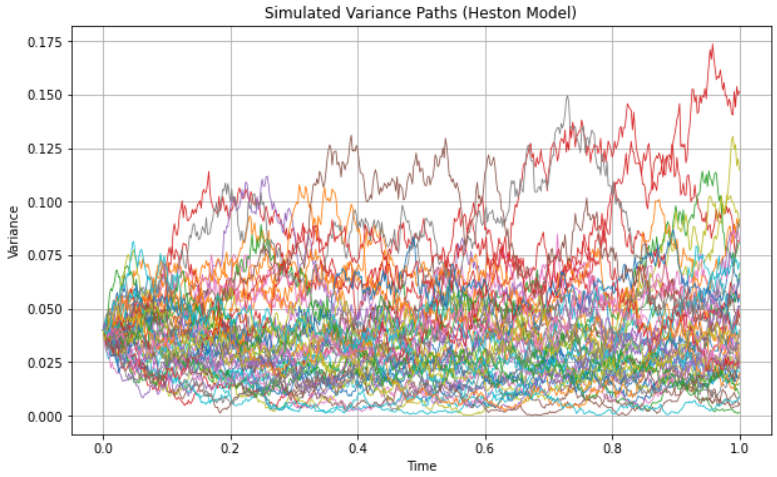


Figure 5.2: Simulated variance paths

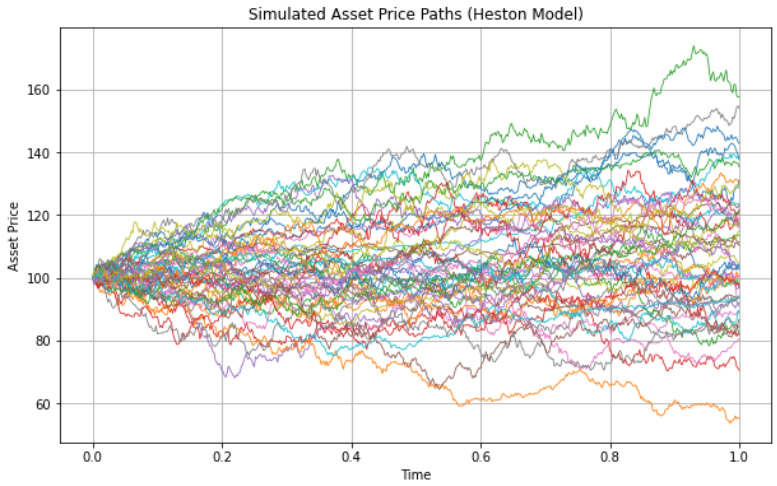


Figure 5.3: Simulated asset price paths

5.5 European Options

Partial differential equations (PDEs) are used to describe the relationships between the partial derivatives of a multivariable function and are commonly used to model natural phenomena and multidimensional dynamical systems. In finance solving PDEs plays a pivotal role in addressing problems such as derivative pricing, optimal execution, and various other applications.

5.5.1 Black Scholes PDE

In this section, we will define the assumptions underlying the Black-Scholes model and present its partial differential equation (PDE). The Black-Scholes model plays a pivotal role as a benchmark of accuracy in this dissertation, and an overview of its foundational principles and the derivation will deepen the reader's understanding and provide context for the machine learning application.

The Black-Scholes model, a Nobel Prize-winning framework developed by Fischer Black and Myron Scholes (1973), revolutionised the domain of quantitative finance by introducting a closed-form solution for pricing options and predicting the behavior of financial instruments over time. Based on the assumption that the price of the underlying asset follows a geometric Brownian motion with constant drift and volatility, the model delivers a mathematically elegant approach. In this dissertation, it plays a significant role as a benchmark for neural network performance due to its simplicity, offering a foundational standard for comparing more complex models. Neural networks can be comprehensively tested against the Black-Scholes model to evaluate their accuracy and robustness, particularly in capturing market-dynamics that deviate from the model's core assumptions, such as stochastic volatility.

We will examine the ideal market conditions underpinning the Black-Scholes model and outline its theoretical framework.

Assumptions of the Black-Scholes Model

1. The option can only be exercised at maturity; hence, the model is applied to European options.
2. No dividends are paid on the underlying asset during the life of the option.
3. Markets are assumed to be efficient, with no arbitrage opportunities available.
4. Transaction costs, such as those incurred during the buying or selling of the stock or option, are neglected.
5. The risk-free interest rate is assumed to be constant.
6. The returns of the underlying asset are assumed to follow a normal distribution, with the implication that the volatility of the market is constant over time.

This foundation provides the basis for analyzing the model's applicability as well as it's limitations in real-world financial markets.

The Feynman-Kac theorem plays a crucial role in linking stochastic processes to partial differential equations (PDEs), forming the basis for the valuation of derivatives like European options. By providing a comprehensive mathematical framework, it enables the transformation of probabilistic asset price models into the deterministic PDE governing option values, as outlined by

Shreve (2004). To derive the Black-Scholes equation for valuing a vanilla European option, we begin by formulating the Feynman-Kac equation, which serves as the foundation for defining the analytical solution and the parameters d_1 and d_2. We draw upon the work by Pu (2021) and Øksendal (2003).

Theorem 1: Feynman-Kac (Pu, 2021)

Suppose $V(x_t, t)$ is the solution to the partial differential equation:

$$\begin{cases} \frac{\partial V}{\partial t} + \mu(x_t, t)\frac{\partial V}{\partial x} + \frac{1}{2}\sigma(x_t, t)^2 \frac{\partial^2 V}{\partial x^2} - r(t, x)V(x_t, t) = 0, & t < T; \\ V(x_t, T) = g(x), & t = T. \end{cases}$$

The Feynman-Kac theorem stipulates that $V(x_t, t)$ has solution:

$$V(t, x_t) = \mathbb{E}\left[e^{-\int_t^T r(X_u, u)\, du} g(X_T) \mid \mathcal{F}_t\right]. \tag{5.14}$$

where $x = (x_s)_{s\in[t,T]}$ is the solution to the stochastic differential equation

$$dx_s = \mu(x_s, s)\, ds + \sigma(x_s, s)\, dB_s, \quad x_t = x \tag{5.15}$$

with B_s as a Brownian motion.

To apply the Feynman-Kac theorem to the Black-Scholes model, we consider the price of an option under the risk-neutral measure $\mathbb{Q}$, where the underlying asset S_t follows the SDE:

$$dS_t = rS_t\, dt + \sigma S_t\, dB_t^{\mathbb{Q}}, \tag{5.16}$$

where:

- r: Constant risk-free interest rate,
- σ: Constant volatility,
- $B_t^{\mathbb{Q}}$: Standard Brownian motion under $\mathbb{Q}$.

The fair value of the option $V(t, S_t)$, with a payoff $g(S_T)$ at maturity T, is given by:

$$V(t, S_t) = \mathbb{E}^{\mathbb{Q}}\left[g(S_T)e^{-r(T-t)} \mid \mathcal{F}_t\right]. \tag{5.17}$$

Substituting into the general Feynman-Kac framework, $V(t, S_t)$ satisfies the Black-Scholes PDE:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV = 0, \quad t < T, \tag{5.18}$$

with boundary condition $V(T, S) = g(S)$.

Boundary Conditions for European Options

Call Option:

- Terminal condition: $V(T, S) = \max(S - K, 0)$.

- When $S \to 0$, $V(t, S) \to 0$.
- When $S \to \infty$, $V(t, S) \to S - Ke^{-r(T-t)}$.

Put Option:

- Terminal condition: $V(T, S) = \max(K - S, 0)$.
- When $S \to 0$, $V(t, S) \to Ke^{-r(T-t)}$.
- When $S \to \infty$, $V(t, S) \to 0$.

Analytical Solution

With the boundary conditions imposed, the analytical solution for the option price is:

$$V_{\text{Call}}(t, S) = S\mathcal{N}(d_1) - Ke^{-r(T-t)}\Phi(d_2), \tag{5.19}$$

$$V_{\text{Put}}(t, S) = Ke^{-r(T-t)}\mathcal{N}(-d_2) - S\Phi N(-d_1), \tag{5.20}$$

where:

$$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}}, \quad d_2 = d_1 - \sigma\sqrt{T-t}. \tag{5.21}$$

Φ represents the cumulative distribution function (CDF) of the standard normal distribution.

The graph below illustrates the European call and put option using the Black-Scholes model. It is presented solely for visualization purposes. We consider the parameters with S and $T - t$ being varied:
$S_0 = 100$, $K = 100$, $r = 0.05$, $\sigma = 0.2$, and $T = 1$.

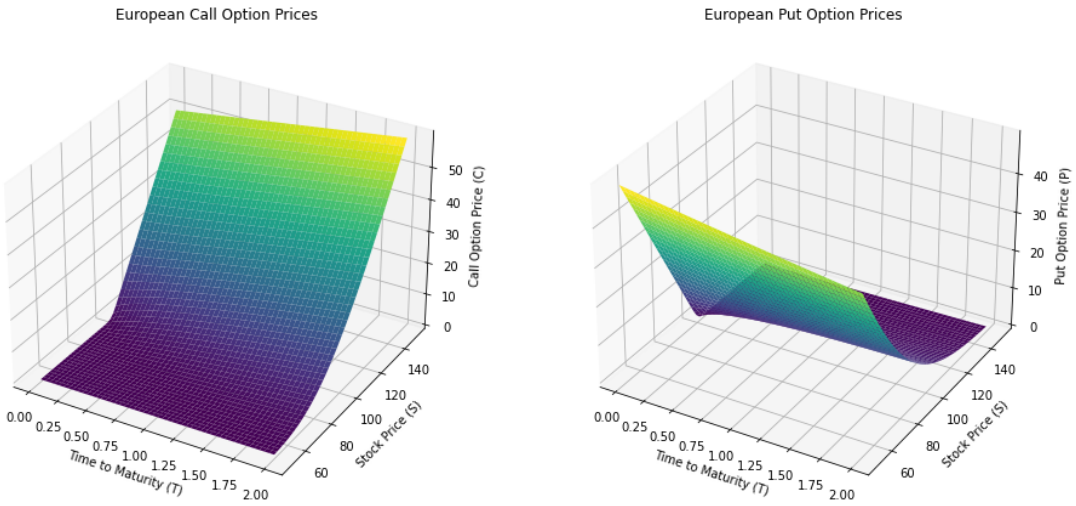


Figure 5.4: Black-Scholes: European call and put option prices

5.6 American Options

5.6.1 The PDE for American Options

Unlike European options, American options allow the holder to exercise the option at any time prior to the maturity date, receiving the payoff immediately based on the current value of the underlying asset. This problem can be described mathematically as an **optimal stopping problem**. The stopping problem and the dynamic programming approach can be expressed drawing upon the research from Pu (2021).

Definition 3: Stopping Time - (Pu,2021)

Let $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\geq 0}, \mathbb{P})$ be a filtered probability space. A stopping time τ is a random variable $\tau : \Omega \to [0, \infty)$ such that for all $t \geq 0$:

$$\{\omega : \tau(\omega) \leq t\} \in \mathcal{F}_t. \tag{5.22}$$

This ensures that the decision to stop is based solely on the information available up to time t.

Remark: For every t, the above definition is equivalent in discrete time to $\{\tau = t\} \in \mathcal{F}_t$

Valuation of an American Option

The value of an American option at time t, given the underlying price $S_t = S$, is given by:

$$V(t, S) = \sup_{\tau \in [t,T]} \mathbb{E}^{\mathbb{Q}} \left[g(S_\tau) e^{-r(\tau - t)} \mid S_t = S \right], \tag{5.23}$$

where:

- $\mathbb{Q}$ is the risk-neutral measure,
- $g(S_\tau)$ is the payoff upon exercise at time τ,
- r is the constant risk-free rate,
- T is the maturity of the option.

The condition $V(t, S) \geq g(S)$ ensures that the option value at any time is at least as large as its immediate exercise payoff.

Dynamic Programming Approach

To analyze the behavior of $V(t, S)$, we consider the decision to hold or exercise the option over

a very small time interval $[t, t+\delta t]$.

$$\begin{aligned} e^{r\delta t}V(t,S) &= e^{r\delta t} \sup_{\tau\in\mathcal{T}[t,T]} \mathbb{E}^{\mathbb{Q}}\left[g(S_\tau)e^{-r(\tau-t)} \mid S_t\right] \\ &= \sup_{\tau\in\mathcal{T}[t,T]} \mathbb{E}^{\mathbb{Q}}\left[e^{r\delta t}g(S_\tau)e^{-r(\tau-t)} \mid S_t\right] \\ &\geq \sup_{\tau\in\mathcal{T}[t+\delta t,T]} \mathbb{E}^{\mathbb{Q}}\left[g(S_\tau)e^{-r(\tau-(t+\delta t))}\cdot e^{-r\delta t} \mid S_{t+\delta t}\right] \\ &= \mathbb{E}^{\mathbb{Q}}\left[\sup_{\tau\in\mathcal{T}[t+\delta t,T]} \mathbb{E}^{\mathbb{Q}}\left[g(S_\tau)e^{-r(\tau-(t+\delta t))} \mid S_{t+\delta t}\right] \mid S_t\right] \\ &= \mathbb{E}^{\mathbb{Q}}\left[V(t+\delta t, S_{t+\delta t}) \mid S_t\right]. \end{aligned} \tag{5.24}$$

Here, $S_{t+\delta t}$ is the underlying asset price at time $t+\delta t$. We have depicted that:

$$e^{r\delta t}V(t,S) \geq \mathbb{E}^{\mathbb{Q}}\left[V(t+\delta t, S_{t+\delta t}) \mid S_t\right]. \tag{5.25}$$

The value of the option at time t can be expressed as:

$$e^{r\delta t}V(t,S) \geq \mathbb{E}^{\mathbb{Q}}\left[V(t+\delta t, S_{t+\delta t}) \mid S_t = S\right], \tag{5.26}$$

where $S_{t+\delta t}$ represents the underlying asset price at $t+\delta t$. This inequality ensures that the option holder maximizes their value by choosing the optimal exercise time.

5.7 Tree Based Methods

5.7.1 Lattice Methods

This section outlines the key equations of the binomial tree framework, emphasizing the logical flow between the steps and their derivations. These equations provide the foundation for the Cox-Ross-Rubinstein (CRR) binomial tree model, a pivotal component of this dissertation. The CRR method serves as a benchmark for evaluating the accuracy of neural networks in option pricing, offering a direct comparison between traditional and machine-learning-based approaches. By leveraging the CRR model as a standard, we can assess the viability of neural networks as an alternative pricing method. Furthermore, this comparison allows us to examine how the choice of input parameters affects neural network performance across varying architectures in order to find the optimal parameters for best performance. These are key objectives of our dissertation.

The binomial tree method is a discrete-time numerical approach use for the valuing of options, particularly American options. Originally developed by Cox, Ross, and Rubinstein in 1979, the binomial tree method models the potential future price paths of the underlying asset over the life of the option through a lattice or "tree" structure. By considering more than one period, the binomial tree is formed where the price of the underlying branches either upward or downward. We will provide a high-level overview of the process, drawing upon the work by Hull (2015), Cox, Ross and Rubinstein (1979), and Wang (2022). The complete derivations with diagrams are provided in the appendix.

The hedge portfolio is constructed by writing a single call and buying Δ units of the underlying

stock. The portfolio value is given by:

$$\Delta S_0 d - c_d = \Delta S_0 u - c_u, \tag{5.27}$$

where:

- $c_u = \max(0, S_0 u - K)$ is the call option value in the upward state,
- $c_d = \max(0, S_0 d - K)$ is the call option value in the downward state.

Solving for Δ, we obtain the hedge ratio:

$$\Delta = \frac{c_u - c_d}{S_0 u - S_0 d}. \tag{5.28}$$

This introduces the construction of a risk-free portfolio to eliminate arbitrage, forming the basis for option pricing in a binomial tree model. The price of the option at $t = 0$ is derived by discounting the expected payoff under the risk-neutral measure as depicted by Wang (2022) is:

$$c = e^{-rT} \left(p c_u + (1 - p) c_d \right), \tag{5.29}$$

where p is the risk-neutral probability Hull (2015):

$$p = \frac{e^{rT} - d}{u - d}, \quad 1 - p = \frac{u - e^{rT}}{u - d}. \tag{5.30}$$

The risk-neutral valuation formula ensures that the expected returns of all securities are adjusted to reflect the risk-free rate. Under the risk-neutral measure, the expected value of the underlying asset at time $t + \Delta t$ as noted by Korn and Korn (2001) is:

$$\mathbb{E}_B(S_{t+\Delta t}) = p(uS_t) + (1 - p)(dS_t), \tag{5.31}$$

where p and $1-p$ are the risk-neutral probabilities defined earlier. The variance of the underlying asset at time $t + \Delta t$ is:

$$\mathrm{Var}_B(S_{t+\Delta t}) = \mathbb{E}_B(S_{t+\Delta t}^2) - \left(\mathbb{E}_B(S_{t+\Delta t})\right)^2. \tag{5.32}$$

Using S_t, u, and d, this simplifies to:

$$\mathrm{Var}_B(S_{t+\Delta t}) = S_t^2 \sigma^2 \Delta t, \tag{5.33}$$

where σ is the annualised standard deviation of the underlying asset. Assuming that $\ln S_T \sim \mathcal{N}(\ln S_0 + (r - \frac{\sigma^2}{2})T, \sigma^2 T)$, the mean and variance of S_T are given by:

$$\mathbb{E}(S_T) = S_0 e^{rT}, \quad \mathrm{Var}(S_T) = S_0^2 e^{2rT}(e^{\sigma^2 T} - 1). \tag{5.34}$$

We've now established the relationship between the lognormal distribution of the underlying price and the binomial tree framework. To match the mean and variance of the binomial tree with the continuous-time model, the parameters u, d, and p are chosen as:

$$u = e^{\sigma\sqrt{\Delta t}}, \quad d = e^{-\sigma\sqrt{\Delta t}}, \tag{5.35}$$

$$p = \frac{e^{r\Delta t} - d}{u - d}, \quad 1 - p = \frac{u - e^{r\Delta t}}{u - d}. \tag{5.36}$$

5.7.2 Early Exercise: Binomial

When using the CRR method, we calculate the option price at each time step and determine the stopping by comparing the immediate exercise value with the expected continuation at each tree node. This utilizes a dynamic programming approach that works backwards through a binomial tree. According to the research conducted by Oussama (2018), the CRR's performance decreases significantly with higher dimensions and this makes it more computationally intensive. The process will be outlined below, drawing upon work by Kwok (2008).

If we recall for American options, the holder has the flexibility to exercise the option early, which introduces an additional intrinsic value at each node in the binomial tree. The continuation value, V_{cont}, is computed as:

$$V_{\text{cont}} = \frac{(p)V_u^{\Delta t} + (1-p)V_d^{\Delta t}}{e^{r\Delta t}}, \tag{5.37}$$

where:

- $V_u^{\Delta t}$ and $V_d^{\Delta t}$ are the option values at the upward and downward states, respectively,
- $p = \frac{e^{r\Delta t}-d}{u-d}$ is the risk-neutral probability of an upward movement.

The total option value, V, at any node is given by:

$$V = \max\left(V_{\text{cont}}, u(S)\right), \tag{5.38}$$

where $u(S)$ is the intrinsic value, representing the payoff from early exercise:

$$u(S) = \begin{cases} S - X, & \text{for a call option,} \\ X - S, & \text{for a put option.} \end{cases} \tag{5.39}$$

This decision-making process ensures that the holder exercises the option when the intrinsic value $u(S)$ exceeds the continuation value V_{cont}; otherwise, the option is held.

Dynamic Programming for Binomial Tree

At each node in the binomial tree, the value of an American option is calculated iteratively using backward induction:

$$P_k^n = \max\left(\frac{(p)P_{k+1}^{n+1} + (q)P_k^{n+1}}{e^{r\Delta t}}, X - S_k^n\right), \tag{5.40}$$

where:

- S_j^n: The underlying asset price at the (n,k) node,
- X: The strike price of the option,
- P_k^n: The option price at the (n,k) node,
- $n = 0, 1, \ldots, N-1$: Time steps in the binomial tree,
- $k = 0, 1, \ldots, n$: Index of the state at a given time step.

This equation compares the continuation value with the intrinsic value to incorporate the early exercise feature, ensuring the flexibility inherent in American options.

The graphs below illustrates the American call and put option using the binomial model. It is presented solely for visualization purposes. We consider the parameters:
$S_0 = 100$, $K = 100$, $r = 0.05$, $\sigma = 0.2$, $T = 1$, and $N = 100$.

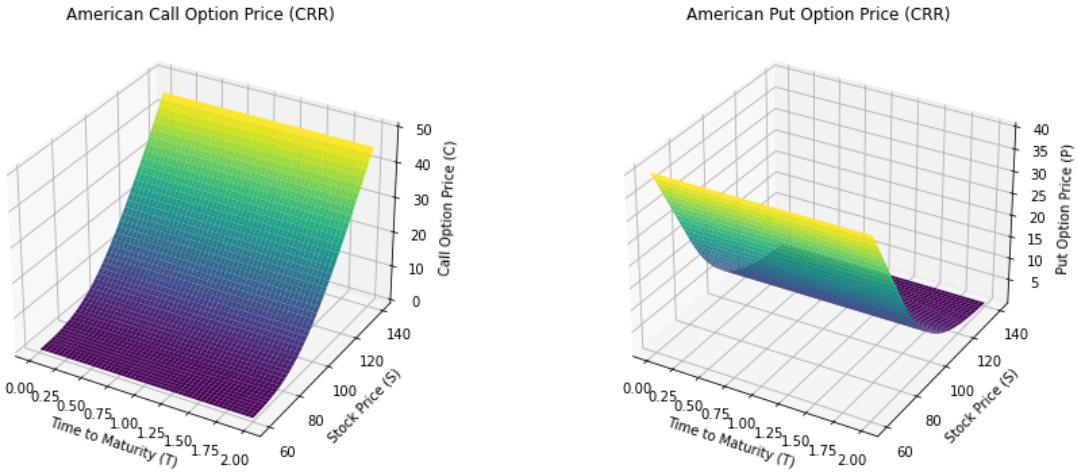


Figure 5.5: Cox-Ross-Rubsinstein model: American call and put option prices

---

Algorithm 2: Cox-Ross-Rubinstein Lattice Method for Valuing American Options - (Tang, 2015)

---

1. For (i, j) such that $i = 0, 1, \ldots, N$ and $j = 0, 1, \ldots, i$, let (i, j) depict the jth node at time i such that the underlying asset value $S_{i,j}$ at node (i, j) is:

$$S_{i,j} = S_0 u^j d^{i-j}.$$

   The option payoff at potential exercise dates represented as $V^*_{i,j}$, is calculated as:

$$V^*_{i,j} = \begin{cases} (S_{i,j} - K)^+ & \text{(Call)} \\ (K - S_{i,j})^+ & \text{(Put)}. \end{cases}$$

   At maturity, the option value $V_{i,j}$ is depicted as $V_{N,j}$.

2. For $i = N - 1, N - 2, \ldots, 0$, the value of the option at node (i, j) can be determined by stepping backward over a single period Δt:

$$V_{i,j} = e^{-(r-\delta)\Delta t} \left(pV_{i+1,j+1} + (q)V_{i+1,j} \right).$$

   To decide on potential exercise, we make a comparison between the intrinsic value of the option with the expected discounted cash flow:

$$V_{i,j} = \max(V^*_{i,j}, V_{i,j}).$$

3. Output $V_{0,0}$.

5.7.3 Convergence

A primary objective of this dissertation is to present the potential of neural networks as a viable alternative to traditional benchmark methods for option pricing. This is achieved by comparing the option values generated using neural networks with those obtained from traditional benchmark methods, namely the Black-Scholes model (for European options), the Cox-Ross-Rubinstein binomial method and the Least-Squares Monte Carlo method (for American options). To ensure the reliability of the Cox-Ross-Rubinstein and the Least-Squares Monte Carlo models as benchmarks, it is essential to establish their convergence properties. In this section, we examine the convergence of the Cox-Ross-Rubinstein model drawing from work by Jiang & Dai (2004) and Leisen (1996). The convergence of the LSMC model will be discussed in a later chaper.

The convergence of the Cox-Ross-Rubinstein (CRR) model, is a critical topic in ensuring the accuracy and reliability of option pricing. In this context, we refer to convergence as the property that as the number of time steps N in the binomial tree increases, the computed option price approaches the theoretical (continuous-time) price, typically derived using a model like Black-Scholes. The convergence proof for American vanilla options using the partial differential equation approach is described by Jiang and Dia (1999), and the proof for American options using the probabilistic approach is described by Amin & Khanna (1994). In their work, Jiang and Dai (2004) provide a unifying framework, using a partial differential equation (PDE) approach, to demonstrate the uniform convergence of binomial tree methods for both European and American path-dependent options. Their methodology builds on the results of Barles & Souganidis (1991), which establishes that any numerical scheme that is stable, monotone, and consistent will converge, provided the limiting equation satisfies a strong comparison principle in the context of viscosity solutions.

Leisen and Reimer (1996) examined the convergence speed by order of convergence for European call options. However, they were unable to extend their analysis to the American put option scenario. Similarly, Lamberton (1995) encountered the same problem for the Cox, Ross, and Rubinstein (CRR) model, and derived bounds for the convergence error. Despite this, Broadie and Detemple (1996) presented simulation results suggesting a convergence order of one. They depicted the oscillatory convergenve of the binomial model graphically.The primary objective of the research by Leisen (1996) is to extend on the results, and demonstrate that the CRR model achieves convergence with an order of one. Additionally, Leisen derives an error representation based on the concept of the order of convergence, enabling a more detailed error analysis.

Given the aim of pricing a contingent claim on a stock S_0, let the continuous-time price be denoted as P_∞. Suppose we employ an approach that generates a sequence of lattices $(\overline{B}_n)_n$, which in turn produces a sequence of discrete prices $(P_n)_n$. Leisen (1996) draws upon the work from Kushner (1977) and Lamberton & Pagès(1990) to demonstrate that discrete American put prices converge to the continuous-time price P_∞.This implies that we define the error for each lattice as:

$$e_n := |P_\infty - P_n|, \tag{5.41}$$

such that if $n \to \infty$, $e_n \to 0$.

To measure the rate of convergence we compare it to sequences such as $(1/n)_n, (1/n^2)_n, \ldots,$

therefore, employing the 'order of convergence' concept.

Definition 4: Order of Convergence - (Leisen, 1996)
Let $(\overline{B}_n)_n$ represent a sequence of lattices. If $(P_n)_n$ is the corresponding sequence of prices derived from these lattices, then the sequence $(P_n)_n$ is said to *converge with order* $\rho > 0$ if $\exists$ a constant $\kappa > 0$ such that:

$$\forall n \in \mathbb{N}: \quad e_n \leq \frac{\kappa}{n^\rho}. \tag{5.42}$$

Also depicted as $e_n = O(1/n^\rho)$.

Therefore, there is an implied convergence of prices, by any order greater than zero.

Drawing upon the work from the above sources, we note that the CRR method for American options converges as the number of time steps $n \to \infty$. By increasing n the CRR model can reliably approximate American option prices. We will graphically depict that the CRR method converges by increasing the number of time steps. The graph below depicts the convergence of the CRR method (American options), using the following example parameters:

- $S = 100$,
- $K = 100$,
- $r = 0.05$,
- $\sigma = 0.2$,
- $T = 2$,
- $N = 50$

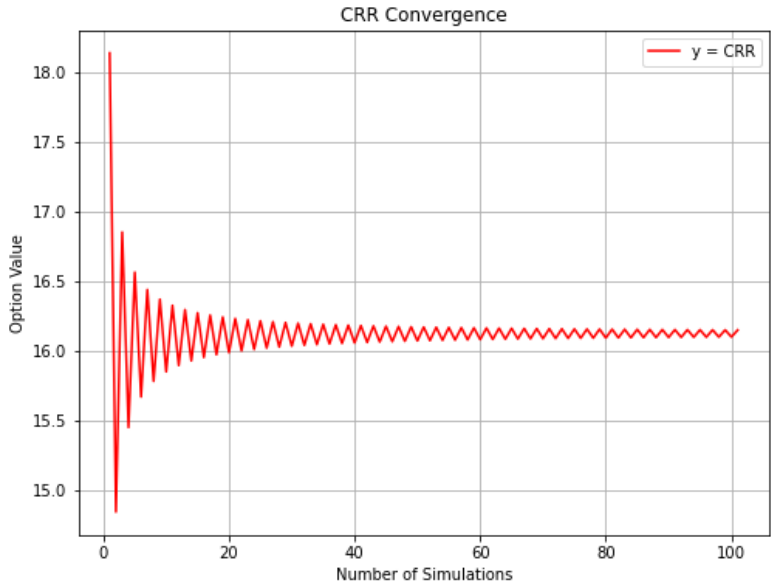


Figure 5.6: CRR convergence rate

The CRR model can now be used as a reliable benchmark for evaluating the performance of neural network models. Building on the detailed exploration of the fundamental concepts underlying traditional methods, we will extend this to encompass the application of machine learning, with a particular focus on neural networks.

5.8 Introduction to Machine Learning

This section provides a comprehensive overview of machine learning (ML) and its associated subfields, offering a foundational understanding essential for the subsequent exploration of deep learning techniques. While the terms "artificial intelligence" (AI) and "machine learning" are often used synonymously, we will distinguish the conceptual distinctions between them. Given that this dissertation is centered on the application of machine learning methodologies for valuing contingent claims, this section is pivotal in laying the groundwork necessary for the detailed implementation of deep learning to follow.

The section begins with an introduction to the principles of machine learning and its diverse applications within the financial field. This is followed by an examination of the various subsets of artificial intelligence, where we provide descriptions and examples of each. The section concludes with a focused analysis of deep learning, particularly artificial and deep neural networks, which form the cornerstone of this dissertation. This foundational discussion will act as a basis for the subsequent chapter, where we delve into the intricate architectures of neural networks in greater detail.

The term "artificial intelligence" (AI) was first coined by John McCarthy in 1956 to describe the concept of "thinking machines." However, progress in the field was significantly hindered by the technological limitations of the time, including insufficent storage capacity and inadequate computational power. These challenges led to a decline in investor interest, resulting in a substantial reduction in financial support and funding for AI research. This period of stagnation, occurring between 1974-1980 and again between 1987-1993, is commonly referred to as the "AI winters," as highlighted by Bahoo et al. (2024). While no unified agreement exists on the definition of AI, a comprehensive definition was proposed by Acemoglu and Restrepo (2020), who claim that artificial intelligence is:

> *"... the study and development of intelligent (machine) agents, which are machines, software or algorithms that act intelligently by recognising and responding to their environment."*

The integration of artificial intelligence is expected to significantly boost global GDP, primarily driven by enhancements in productivity, which are estimated to account for more than half of the overall economic gains. According to Szőke (2017), by 2030, AI is expected to increase global GDP by 14%, translating to an additional $15.7 trillion, highlighting its transformative economic impact.

The earliest applications of artificial intelligence in finance, as noted by Quinn (2023), were relatively basic, relying on deterministic, pre-programmed rules proposed by experts. These systems were applied to areas such as automated trading and risk analysis, offering transparent and predictable results. However, their effectiveness was constrained by their inflexible design, making them unsuitable for dynamic environments characterized by unpredictable conditions. Quinn (2023) further highlights that these limitations prompted the development of more adaptive approaches, leading to the rise of machine learning algorithms. Unlike their rule-based predecessors, ML models possess the capability to analyze data, identify patterns, and make

predictions, thereby introducing a level of flexibility. These advancements have enabled ML to be implemented across a broad spectrum of financial domains, including portfolio management, fraud detection, and algorithmic trading. The rapid evolution of AI technologies has not only transformed this sector but also revealed new avenues for academic research, prompting continuous innovation. Remarkable and sustained advancements in computational methodologies have been observed, as noted by (LeCun et al., 2015). However, the progression of machine learning algorithms has been accompanied by a range of challenges that demand careful consideration.

Rudin (2019) highlights the degree of intricacy and obscurity introduced when using complex machine learning models. This is known as 'black box' problems, where the internal decision-making process is hidden from users. Deep learning models are considered black boxes due to their recursive nature. The trade-off between efficiency and interpretability is an important point of consideration, especially with its application to neural networks.

A notable example of effective use of AI is in the stock market, particularly in two areas: algorithmic trading and stock prediction. The benefits of AI in finance, and specifically of algorithmic trading have been covered extensively by Hendershott et al (2011), Frino et al.(2017), Kelejian & Mukerji, (2016) and Litzenberger et al. (2012), and are discussed below. The benefits of using algorithmic trading outweigh the drawbacks, and include: enhancing market liquidity by narrowing spreads, mitigating adverse selection, and improving trade-related price discovery. Firms benefit from a lowered cost of equity, and as a result of improved adaptability to information there are higher profits due to superior market timing and faster execution. Although High-Frequency Trading (HFT) has occasionally amplified volatility, algorithmic trading has debatably decreased the return volatility variance and enhanced market efficiency.

The paper by (Balan & Pulakkazhy, 2013) explores how data mining techniques can be applied to the banking sector to enhance the decision-making processes. ML techniques such as: classification, clustering, regression and association are used to discover any hidden patterns, trends and relationships in the vast amounts of data from multiple sources. Examples include: classification techniques such as decision trees and neural networks being used to classify transactions as fraudulent or non-fraudulent as noted by Ngai et al. (2011); linear and non-linear regression used for predicting stock prices, credit scoring and loan repayment probabilities as noted by Li & Liao (2011); K-means clustering used to improve on customer segmentation as noted by Madhavan et al. (2012); and association rules, such as the apriori algorithm, to decipher relationships between different financial products or transaction patterns. We also take note of the specific use of neural networks in the finance sector. Examples outlined by Shih (2011) include using feedforward neural networks applied to credit scoring and fraud detection, and self-organizing maps (a type of neural network for clustering), and the value it can add in credit rating analysis and customer segmentation.

While the examples above demonstrate the effective use of machine learning in finance, it's important to carefully weigh both its advantages and limitations. The decision to use machine learning methods over traditional approaches ultimately depends on the discretion of the user.

In highly regulated environments, the use of machine learning is often constrained due to its lack of transparency. Regulatory bodies, such as the Basel Committee, European Central Bank, and Federal Reserve, require banks to explain and justify their decision-making processes. Black box models, including deep neural networks and complex machine learning algorithms, often produce outcomes that are difficult to interpret, making them less suitable for such regulated settings. Often the simpler machine learning models may prove to be more effective, especially

in environments when accuracy is prioritized over time efficiency. A simple example in banking is an AIRB Probability of default model, where using linear regression is still an ideal choice over more advanced machine learning models (Wrigglesworth, 2021).

A machine learning model was developed by Creamer & Freund (2010) that analyses stock price series and identifies the best-performing assets, by recommending either a short position or a long position. Additionally, the model incorporates a risk management layer that prevents transactions when the trading strategy is considered unprofitable. This logic was applied by Creamer (2012) in high-frequency trading futures. The model identifies the most porfitable and least risky futures, providing either a long or a short recommendation. Research has also been carried out in the field of AI applied to portfolio management. For example, Soleymani and Vasighi (2020) use a clustering approach combined with Value at Risk (VaR) analysis to improve asset allocation. The least risky and most profitable stocks were grouped together, and are then allocated to the portfolio.

> *"It is difficult to think of a major industry that AI will not transform. This includes healthcare, education, transportation, retail, communications, and agriculture. There are surprisingly clear paths for AI to make a big difference in all of these industries."*
> *- Andrew Ng, Computer Scientist and founder of the Google Brain*

To gain a comprehensive understanding of the integration of artificial intelligence with deep learning, particularly within the domain of neural networks, it is crucial to grasp the subsets of artificial intelligence and the categories within each of these subsets. To analyze these subsets, we will draw upon work from Wrigglesworth (2021), Pykes (2024), Maini & Sabri(2017) and Pramoditha (2022). The subset diagram is depicted below, and was obtained from Pramoditha (2022)

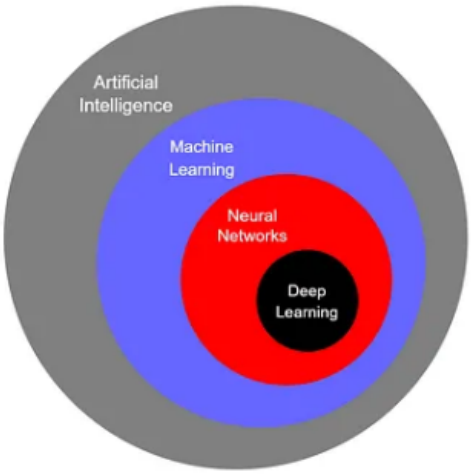


Figure 5.7: Subset Diagram of Artificial Intelligence

Artifical Intelligence

Artificial Intelligence (AI) is described as a field of computer science that focuses on creating systems that can perform tasks which typically require human intelligence. This includes recognizing patterns in data, making decisions, solving problems and it can be used for prediction. AI

encompasses a range of techniques, algorithms, and methodologies which is aimed at mimicking cognitive functions from data it is provided with, or from past experience. Machine Learning and Deep Learning are important subsets of AI. The diagram below illustrates the different branches and techniques of machine learning.

Machine Learning

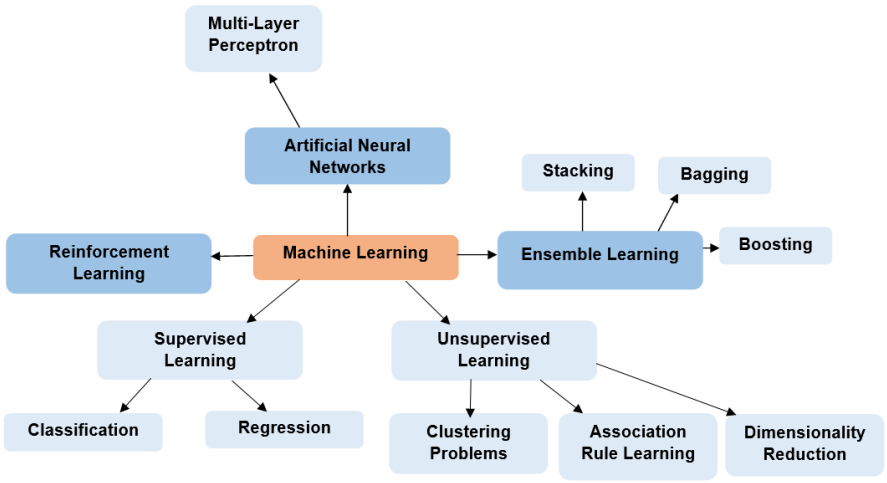


Figure 5.8: Machine Learning Diagram

Machine learning (ML) is a specialized branch of artificial intelligence (AI) which focuses on developing algorithms enabling computers to learn from data and improve performance over time without explicit programming to do so (Russell & Norvig, 2020). This contrasts with traditional programming, where explicit instructions are provided for each scenario. Instead, machine learning systems adapt and refine their outputs as they are exposed to more data. While AI encompasses a wide range of applications such as robotics, rule-based systems, and natural language processing (NLP), machine learning is primarily data-driven, relying on statistical and computational techniques (Mitchell, 1997). ML methods are extensively used in applications such as pattern recognition, classification, and decision-making. Unlike rule-based systems, machine learning incorporates adaptive processes, allowing it to handle dynamic and complex environments effectively. Machine learning methods are typically categorized into three primary types: supervised learning, unsupervised learning, and reinforcement learning. Each method serves a unique purpose based on the type of problem and the structure of the data.

Classical Machine Learning Methods

- Supervised Learning

  Supervised learning involves training a model on labeled data, where each data sample (input) is paired with a corresponding label (output). The objective is to discover a function that maps inputs to outputs accurately, enabling the model to predict outputs for new, unseen data.

Supervised learning is divided into 2 types:

- **Classification**: Drawing upon the work by Hastie et al.(2009), we note that classification involves predicting a discrete class label. For instance, logistic regression, support vector machines (SVMs), and k-nearest neighbors (k-NN) are typically used algorithms for classification problems such as spam detection in emails, where messages are categorized as either "spam" or "not spam".
- **Regression**: Drawing upon the work by James, et al. (2013), we note that in regression tasks, the output variable is continuous. Examples include predicting housing prices or weather conditions based on historical data. Techniques such as linear regression, polynomial regression, and support vector regression are widely examples of regression techniques.

Supervised learning has been pivotal in domains such as healthcare for disease diagnosis, and this has been highlighted in research for clinical epidemiology by Ono & Goto (2022). Other applications include its use in finance for fraud detection, and marketing for customer segmentation. Its effectiveness lies in its ability to generalize from labeled datasets, provided they are diverse and accurately represent the underlying problem space. The following diagram depicts the general dataflow process for supervised learning, obtained from Raschka (2014).

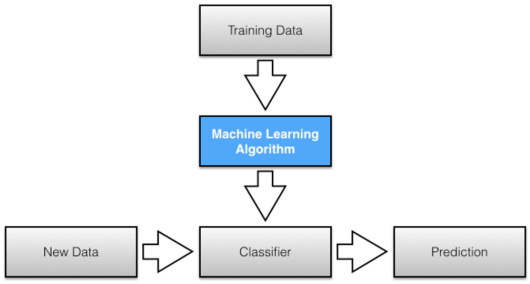


Figure 5.9: Supervised Learning Dataflow

The figure illustrates the dataflow process in supervised learning, which follows these steps:

1. **Training Data:** The process begins with labeled training data which is composed of input-output pairs. These are utilized to train the machine learning model.
2. **Machine Learning Algorithm:** The training data is fed into the machine learning algorithm, which actively identifies any patterns by the minimization of the difference between predicted and actual outputs.
3. **Classifier:** After the training process, the ML algorithm produces a model (classifier) which is capable of generalizing identified patterns to new, unseen data.
4. **New Data:** The trained classifier processes new input data, which lacks labels, for making predictions.
5. **Prediction:** Predictions are then generated by the classifier, and are based on the patterns learned during the training process.

In this dissertation, supervised learning plays a crucial role in the development of predictive models for option pricing. By utilizing labeled datasets comprising of simulated market data, and variables such as stock prices, volatilities, and strike prices, supervised learning algorithms can generalize certain patterns that are essential for option pricing. This aligns with our dissertation's goal of using neural networks, particularly artificial neural networks (ANN) and deep neural networks (DNN), to provide efficient and accurate pricing methods in comparison to traditional benchmark models. We aim to assess the effectiveness of supervised learning and its application to handling the complexities of financial data and improving prediction accuracy, including in volatile market conditions.

- Unsupervised Learning

  Unsupervised learning involves training machines on unlabeled datasets to uncover patterns, relationships, or structures within the data. Unlike supervised learning, there are no explicit labels to guide the algorithm, making this approach well-suited for exploratory data analysis and identifying hidden patterns. Unsupervised learning can be broadly categorized into three types:

  - **Clustering Problems:**
    The objective is to group a set of objects so that objects within the same cluster are more similar to one another than to those in other clusters. Popular clustering models include K-Means Clustering and Hierarchical Clustering. Clustering is frequently used to enhance search engines by grouping similar content or organizing large datasets for easier analysis (Jain et al., 1999). For example, clustering can enhance recommendation systems by grouping users based on their similar behavior or preferences.
  - **Association Rule Learning:**
    This approach seeks to identify specific rules or relationships between attributes in a dataset. A common method used is the Apriori Algorithm, widely used in market basket analysis and discussed by Ain (2023). For example, patterns such as customers who purchase milk and bread are also likely to buy butter can help businesses optimize product placement and inventory strategies (Agrawal et al., 1993). This type of unsupervised learning supports applications that are aimed at increasing sales and improving customer targeting.
  - **Dimensionality Reduction:**
    Dimensionality reduction involves reducing the number of input features in a dataset while preserving the most critical information. Techniques such as Principal Component Analysis (PCA) and t-SNE (t-Distributed Stochastic Neighbor Embedding) are commonly used. These methods simplify complex models, improve computational efficiency, and visualise high-dimensional data. In financial applications, dimensionality reduction can aid in risk assessment and portfolio optimisation by isolating the most influential factors. The application to PCA is discussed extensively by Jolliffe (2002).

  To better visualise the difference between supervised vs unsupervised learning, we consider the illustration by Khatun (2018):

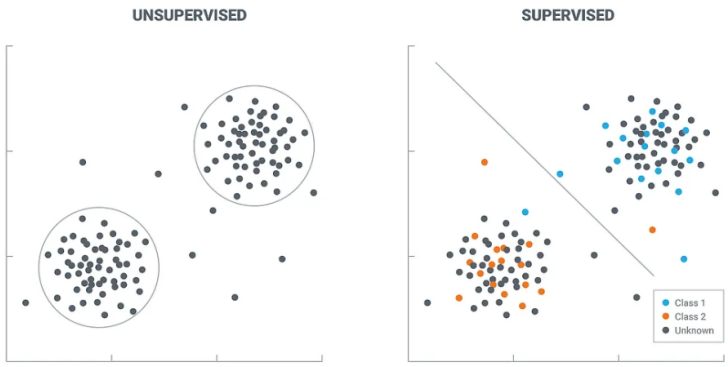


Figure 5.10: Supervised vs unsupervised learning

On the left, the unsupervised learning example demonstrates how data is grouped into clusters based on inherent similarities or patterns, without the use of labeled data. On the right, the supervised learning example depicts labeled data, where each data point belongs to a predefined class (e.g., "Class 1" or "Class 2" in the illustration). A classification boundary is drawn by the model to separate the classes based on the training data.

While unsupervised learning may be effective for discovering hidden patterns and structures in unlabeled data, is not directly applicable to the objectives of this dissertation. The primary focus of this dissertation is on accurately pricing options using neural networks, which requires a supervised learning framework. Since unsupervised learning does not utilize labeled datasets and its application is more geared towards clustering, association rule mining, or dimensionality reduction, it cannot provide the targeted predictive capabilities we required for option pricing models in this particular dissertation focus.

Reinforcement Learning

Reinforcement learning (RL) is essentially a mid-point between supervised and unsupervised learning. It is presented by Sutton & Barto (2018) as a machine learning framework in which an agent acquires decision-making capabilities by engaging with its environment, receiving evaluative feedback in the form of rewards or penalties, and progressively refining its actions to optimize cumulative rewards. Unlike supervised learning, which relies entirely on labeled data, RL involves learning through trial and error by exploring the consequences of actions within a given environment. Semi-supervised learning, on the other hand, operates at the intersection of supervised and unsupervised learning by leveraging both labeled and unlabeled data for training, and is discussed in detail by Maini & Sabri (2017).

A classic example of reinforcement learning is a robot navigating a simulated warehouse environment to optimize item placement on shelves. In this scenario, the robot (or agent) learns to identify items, navigate aisles, and place objects in their correct locations based on state-action interactions. The states represent the robot's understanding of the environment, such as item positions or shelf arrangements, while the actions are the robot's decisions, for example picking up an item or moving to a shelf. Correct placements yield a reward, while incorrect actions result in penalties. Over time, the robot learns the most efficient routes and correct actions to achieve its goals with minimal error. This is visualized in the diagram below, and is depicted by

Wang, et al. (2023). The diagram illustrates the reinforcement learning iterative loop: the agent observes the environment's state, selects an action, receives feedback in the form of rewards or as penalties, and updates its behavior to improve future performance. This interaction continues until the agent achieves best performance for the given task. The reinforcement learning framework is covered extensively by Partridge (2023). The diagram below is obtained from Wang, et al (2023):

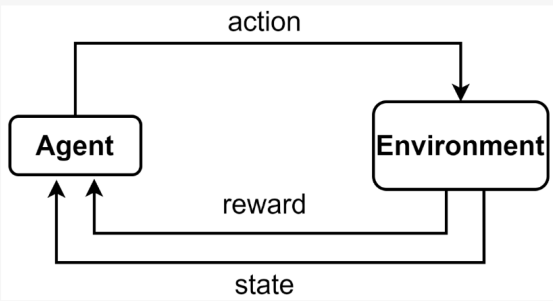


Figure 5.11: Diagram of the reinforcement process

While reinforcement learning is an advanced and promising approach in machine learning, its reliance on the trial-and-error feedback loop and simulated environments makes it unsuitable for the objectives of this dissertation, which require a supervised framework for pricing options as discussed above.

Ensemble Methods

Ensemble methods in machine learning involves the integration of multiple models in order to achieve improved predictive performance in comparison to individual models. This technique leverages the strengths of each model while compensating for their weaknesses, which results in more accurate as well as more reliable predictions. We draw upon the research by Anwar (2021) and from the website (IBM Data[6], 2024). Examples of emsemble methods inclue:

- Boosting: Boosting involves the sequential building of models, with each new model focussing on the correction of errors of its predecessors. By combining these models, boosting seeks to minimize bias and transform weak learners into a strong, composite model (Soni, 2023).
- Bagging: Bagging involves training several models on various subsets of the training data, created through sampling with replacement (bootstrap resampling). The final prediction is then determined by averaging (for regression tasks) or taking the majority vote (for classification tasks) across all models. Bagging is effective in reducing variance and minimizing overfitting.
- Stacking: Stacking involves training of multiple models (base learners) and combines their outputs using a meta-model, which essentially learns the best method to merge the predictions. Stacking enhances predictive accuracy by capturing complex relationships. However, if tuned incorrectly there is the risk of overfitting from predictions (Dutta, 2019).

[6]https://www.ibm.com/topics/ensemble-learning?

Ensemble methods were not considered for this dissertation.

5.9 Neural Networks and Deep Learning

Neural networks are a class of machine learning algorithms and a subset of artificial intelligence (AI), inspired by the structure and functionality of the human brain. These networks consist of layers of interconnected nodes, or *neurons*, which collaboratively process input data, generating meaningful outputs. Each neuron receives inputs, applies specific weights and biases, processes the data using a defined activation function, and transmits the result to the next layer of neurons. During training, the network learns by adjusting these weights and biases to minimize the error between predicted outputs and actual targets. In this section we draw upon the work by Russel & Norvig (2020), Mitchell (1997) and Goodfellow et al. (2016).

5.9.1 The Perceptron: The Simplest Neural Network

The simplest form of a neural network is the perceptron, which was introduced by Rosenblatt (1958) and was conceptually inspired by the nervous system. The perceptron serves as the fundamental building block of more complex neural networks and adheres to the following design structure:

- **Input Nodes**: Each input node is assigned a specific weight and represents a distinct feature of the data.
- **Summation Function**: The perceptron calculates the weighted sum of the inputs.
- **Activation Function**: If the weighted sum surpasses a specified threshold, the perceptron *activates*, outputting one class. Otherwise, it outputs another class.

5.9.2 Neural Network Structure

The Multilayer Perceptron (MLP), building upon the simple perceptron, is a fundamental and versatile architecture in neural networks. Unlike a single-layer perceptron, which is limited to linearly separable problems, MLP introduces multiple layers of neurons, thus allowing the network to learn complex and non-linear patterns in the data (Jaiswal, 2024). The MLP consists of three main types of layers:

- **Input Layer**: Processes the initial data and forwards it to subsequent layers.
- **Hidden Layer(s)**: These layers perform intermediate computations, enabling the network to identify patterns and relationships within the data.
- **Output Layer**: Produces the final prediction or classification based on the processed data.

The diagram below provides a visual representation of a basic neural network, showcasing the interaction between inputs, hidden layers, and outputs. The detailed inner workings of these layers will be discussed and illustrated in the following chapter.

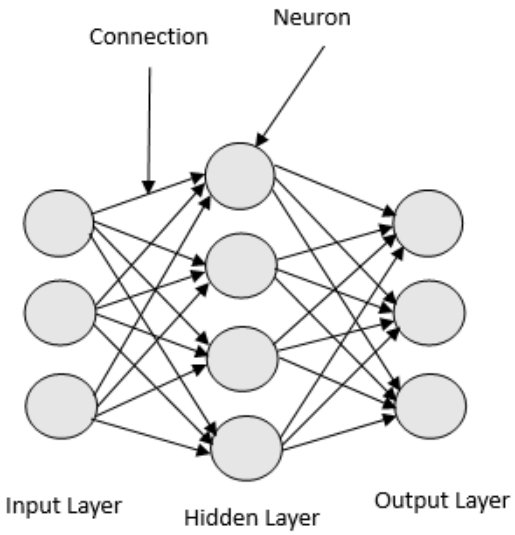


Figure 5.12: Simple Diagram of a Neural Network

Neural Networks can be applied to supervised and unsupervised learning models. Successful application of NNs extend across a broad range of fields with notable examples including:

- Facial recognition
- Medical Imaging
- Language Translation (NLP)
- Fraud detection
- Algorithmic trading
- Climate Modelling

Artificial Neural Networks (ANNs) represent a fundamental type of neural network architecture. Typically consisting of three main layers—an input layer, one or more hidden layers, and an output layer as described above. They are designed to process data by transforming inputs into outputs through weighted connections and activation functions. In this dissertation, we refer to simpler neural networks as "shallow" networks or ANNs.

Deep Learning (DL), on the other hand, is a specialised subset of neural networks characterised by its use of multiple hidden layers, thereby forming deep neural networks (DNNs). While shallow networks are limited to one or two hidden layers, deep learning architectures often include three or more hidden layers. This added depth enables DNNs to capture intricate and non-linear patterns within data, making them particularly well-suited for more complex tasks such as image recognition, natural language processing (NLP), and other high-dimensional applications. The additional layers allow deep learning models to handle more complex datasets and extract more intricate features thus expanding their capabilities beyond traditional ANNs.

Although all deep learning models are neural networks, not all neural networks qualify as deep learning models. The distinction is discussed by Schmidhuber (2015) and lies in the architecture's depth and complexity. By incorporating multiple layers, deep learning provides the computational power needed to address sophisticated problems and adapt to dynamic data environments. We anticipate that the implementation of deep learning improves time performance and accuracy as training data increases, and this has been depicted in areas outside the financial domain. A notable depiction is included below by Wang & Cao (2021) discussing the implementation of machine learning in urban computing:

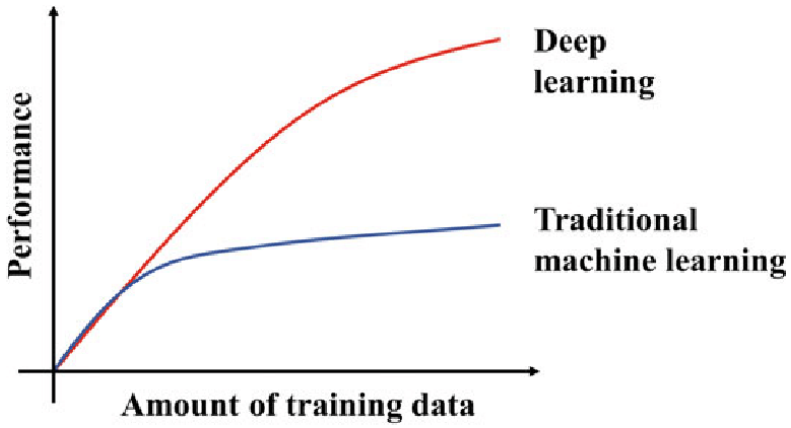


Figure 5.13: Deep Learning vs Traditional Machine Learning

This section lays the groundwork for the next chapter, which delves into both shallow artificial neural networks and deeper learning architectures.

This concludes the introductory section of the dissertation. In the next chapter we will delve into the structure and functionality of neural networks in greater detail. By exploring both shallow artificial neural networks (ANNs) and deep neural networks (DNNs), we will analyze their underlying architectures, key components, and how these models are employed to solve complex problems. This foundational understanding will set the stage for examining their application in option pricing, with a focus on the comparative performance of traditional benchmark models and neural networks. This will lay the foundation for us to reach our dissertation objectives.

Chapter 6

Neural Networks

6.1 Introduction to Neural Networks

The forthcoming section constitutes the crux of our argument, as outlined in the structure of this dissertation. Here, we delve into neural networks with considerable depth and precision. To begin we will present a succinct overview and background of neural networks, building upon the foundation laid in the introduction. Subsequently, a detailed analysis of the architecture of neural networks and their associated graphical and mathematical notation will be provided. Within the scope of this dissertation, we will confine our exploration to their application in supervised learning, dedicating focussed attention to this particular domain.

This section establishes the groundwork necessary for achieving the objectives outlined in our dissertation's results. To summarize, we have laid the conceptual framework for option pricing, positioning neural networks as a compelling alternative to conventional methodologies. In this section, we delve into the architecture of neural networks, subsequently analyzing how the selection of input parameters profoundly impacts their performance in option pricing. Additionally, we will demonstrate the robustness of neural networks in navigating volatile market conditions by comparing their results with the solution provided by the Heston stochastic volatility model, as introduced in the preliminary section.

Haykin (1994) articulated the concept of neural networks non-mathematically, describing them as:

"A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use.
It resembles the brain in two respects:
1. Knowledge is acquired by the network from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge."

Drawing upon reseach conducted by Goodfellow et al. (2016), Neural Networks, as mentioned, derive their name from the behaviour of biological neural networks in the human brain, where neurons communicate through signaling pathways. Structurally, these systems are composed of interconnected nodes, organized into distinct layers. Each node (neuron), receives input data, processes it, and generates an output. The connections between these nodes are characterized

by associated weights, which are iteratively adjusted during the training process to enable the network to learn patterns within the data. When a node's output surpasses a specified threshold linked to its weight, the node becomes activated and will transmit information to subsequent layers of the network. Conversely, if the output fails to meet the threshold, the data does not propagate further.

Using the research by Cybenko (1989) and Hornik (1991) and Hutchinson et al. (1994), it becomes apparent that a multi-layer perceptron (a network consisting of multiple layers of interconnected nodes) can approximate almost any linear or non-linear function with bounded inputs and outputs to arbitrary accuracy. Typically, a neural network with a single hidden layer will be sufficient for this purpose.

However, it is important to note that while a neural network is able to represent almost any function that we are attempting to approximate, there is no guarantee that the function can be learnt. It is noted by Pu (2021) that this may be due to the algorithm not being able to locate the parameters that accurately learn the function, or the wrong functions being learnt instead due to overfitting. An important property to take note of is the 'Universal Approximation Theorem' which can be informally stated as:

Theorem 2: The Universal Approximation Theorem- (Dowling, 2020)

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function on a compact subset of $\mathbb{R}^n$.
For any $\epsilon > 0, \exists$ a feed-forward neural network (NN) with a single hidden layer and a finite number of neurons, using a non-linear activation function, such that the neural network function z approximates f within an error of ϵ over the input space.

6.2 Background

The earliest discussions around the concept of a neural network dates as far back as 1943. McCulloch and Pitts (1943) introduced a mathematical model of how neurons behave in the brain. They explored the concept of neurons as binary units, which either activated or did not based on a certain threshold. They proposed that a neuron fires if the total weighted sum of its inputs surpasses a threshold, which resembles the behaviour of binary activation functions used in the later developments of neural network models. McCulloch and Pitts siggested that the neural networks could be considered universal computation systems, which suggests that they could solve any problem provided there are sufficient neurons and the ideal structure. The comparison to 'Turing machines' assisted in laying the theoretical foundation for the computational abilities of neural networks, as noted by Chakradhar (2024).

Below is the representation of the proposed artifical 'MP' neuron depicted by Chandra (2024). From the diagram, g will take an input (referred to as a 'dendrite' by McCulloch and Pitts), perform an aggregation and based on the aggregated outcome a decision f will be made .

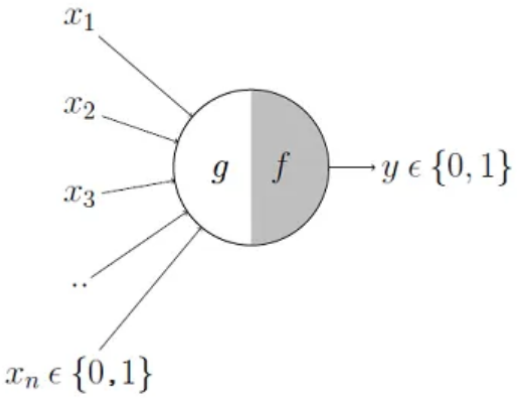


Figure 6.1: Diagram of MP Neuron

This can be depicted formally as:

$$g(x_1, x_2, x_3, \ldots, x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i. \tag{6.1}$$

$$y = f(g(\mathbf{x})) = \begin{cases} 1 & \text{if } g(\mathbf{x}) \geq \theta \\ 0 & \text{if } g(\mathbf{x}) < \theta \end{cases} \tag{6.2}$$

While it was a revolutionary breakthrough in the field and laid the groundwork for later development in the field of neural networks, it was considered impractical. The 'MP neuron' has limitations that hinder its practical application, specifically in its ability to learn from data. In the MP neuron model, the threshold requires manual configuration, which inhibits its ability to adapt to different datasets. Furthermore, its binary input and output design restrict its capability to model complex relationships within data. Later advancements such as continuous-valued activation functions (e.g. ReLU) as well as backpropagation and gradient descent will address these shortcomings.

As a step towards this, Frank Rosenblatt (Rosenblatt, 1958), an American psychologist proposed a model of an artificial neuron, a perceptron, which allowed for flexibility and learning (supervised learning). This was achieved using the workings of McCulloch-Pitts as well as Donald O. Hebb, a psychologist who introduced revolutionary research on the interactions between neurons which provided the basis for self-organized learning. Rosenblatt's model was considered one of the first models to depict machine learning. Application of the perceptron can be found in a variety of fields such as image processing, natural language processing, feature detection etc. Using adaptation from Rosenblatt's work, a diagram of a simple perceptron is depicted below.

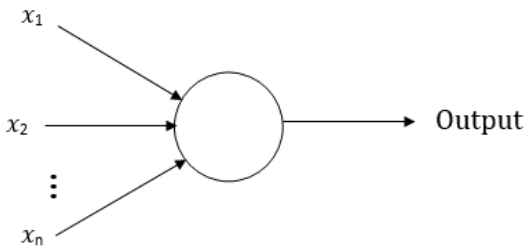


Figure 6.2: A Diagram of a Simple Perceptron

Hebb's work, referred to as 'Hebbian Theory' proposed the concept that neural connections could adapt based on experience. The phrase "Cells that fire together, wire together" was coined based off of Hebb's work, and was used to explain how the brain forms connections between neurons (Hebb, 1949). This inspired neural network models that adjust connection weights over time, and formed a crucial component of the foundation of the neural networks we use today.

While traditionally, the neural network model was used to understand the brain, over time it has been used extensively across a range of areas. We will be exploring the implementation of neural networks in finance.

We consider the use of neural networks in forecasting stock prices. Neural Networks, specifically artificial and deep neural networks, are favoured over linear models due to their ability to effectively grasp the non-linear relationships between stock returns and fundamentals. Kanas (2001) also notes that they are also more responsive to possible changes in the relationships between variables. Use of deep neural networks have also grown in popularity due to their predictive accuracy. The use of Long Short-Term Memory Networks (LSTM) was proposed by Zhang et al. (2021) as a model that surpasses classical neural networks in efficiency and prediction accuracy.

We can also consider the benefits of using neural networks in high-frequency trading (HFT). This is seen in a paper by DeSieno & Trippi (1992) where they combined multiple neural networks into a single Boolean decision rule system that outperforms single neural network models. A key benefit of using multiple neural networks is the improved prediction accuracy by combining the outputs of different networks, in comparison to classical trading methods or single neural network approaches. Each network was trained on a different subset of the data, and could therefore capture various aspects of the market dynamics.

In domains such as stock market forecasting and option pricing, the relationships between variables are often highly intricate. Neural networks are particularly well-suited to these applications, owing to their flexibility in modeling complex interactions. Their architecture enables them to surpass traditional models by effectively capturing certain dependencies and uncovering patterns within noisy datasets. A comprehensive exploration of the neural network architecture and its suitability in capturing these capabilities will be presented in the next chapter.

6.3 Neural Network Architecture

This section will examine the various components comprising the structure of a neural network. A detailed interpretation will be provided, encompassing both the mathematical and graphical rep-

resentations of the network at each stage, progressively building the model from its foundational perceptron to the complexities of a deep network. A thorough exploration of the neural network architecture is imperative, as it directly supports the achievement of our objectives—most notably, determining whether the selection of input parameters significantly impacts the network's performance in option pricing. The insights gained from this chapter will enable the formulation of an "ideal parameters" scenario, which will be presented in the results. We will draw upon the research from Pu (2021) to provide a general overview with detailed explanations to follow in the next section.

The general purpose of a neural network is to approximate a specific function, f, by defining a mapping $y = f(x;\theta)$ and determining the values of the parameters (θ) that represent this function in the best way possible. The mapping between inputs and outputs is governed by several critical factors, including the synaptic weights, the architecture of the neural network—specifically, the number of perceptrons within each layer, the total number of layers, and the arrangement of connections among them. By adjusting these parameters, various types of neural networks can be classified. For example, in feedforward neural networks (FNNs), each neuron in a layer is fully connected to all neurons in the subsequent layer, with no feedback loops that reintroduce the model's outputs into the network. In FNNs, information flows unidirectionally from the input x through the computations defining (f), ultimately producing the output y. This contrasts with recurrent neural networks (RNNs), where the network's output is fed back into itself, enabling the retention and processing of sequential information Owing to their universal approximation properties, feedforward neural networks (FNNs) have proven highly effective across a wide range of applications. Due to their universal approximation capabilities, this dissertation will employ FNNs for the pricing of American and European options (Bhoi et al., 2022).

The data in supervised learning typically consists of input features paired with labels, and it is divided into training, validation, and test datasets. The training dataset can be represented as:

$$\{(x_1, y_1), \ldots, (x_k, y_k)\}, \tag{6.3}$$

where x_i represents the input vector and y_i represents the corresponding label. Given input x_i, the output $f(x_i)$ should be as close as possible to the target y_i. To estimate this closeness, loss functions are defined and minimized using optimization algorithms during the training stage. An example of such a loss function is the Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{k}\sum_{i=1}^{k}(f(x_i) - y_i)^2. \tag{6.4}$$

This calculates the average squared difference between the predicted outputs and the actual labels.

During training, there is a risk of overfitting, where the model performs well on the training data but poorly on new, unseen data. Overfitting is often evident when the training loss decreases while the validation loss increases. To mitigate overfitting, a dropout layer may be applied during training. This involves randomly setting a portion of the input neurons to zero with a probability $p \in [0, 1]$, forcing the network to learn more robust features. Additionally, by reducing the number of epochs or increasing the number of training samples we can also help prevent overfitting. Once trained, the model can be evaluated on the test dataset to assess generalization. These features of neural networks will be discussed in more detail below.

6.3.1 Architecture and Notation

This chapter builds upon the derivations outlined by Lindholm et al. (2019) and Pu (2021), providing a comprehensive understanding of the mathematical framework underpinning neural networks. To enhance clarity, graphical representations will also be incorporated throughout the discussion. The following analysis of neural network architecture is important as it offers critical insights into the potential structural limitations inherent to neural networks, based on the complexity of the data or scenario they are modelling. In addition to addressing the objectives outlined at the beginning of this chapter, this exploration of the architecture will also improve the discussion on the limitations of neural networks within the broader context of machine learning, while highlighting avenues for further research later.

The analysis of neural network architecture is crucial, as it also provides insight into the possible structural limitations of neural networks, based on what they are modelling. Beyond the objectives highlighted at the start of this chapter, discussing the architecture will also provide insght into the section on neural network (machine learning) shortcomings and other areas of research.

A neural network (NN) is a nonlinear function that models the output variable y as a nonlinear function of input variables $x_1, ..., x_k$:

$$y = f(x_1, \ldots, x_k; \theta) + \epsilon, \tag{6.5}$$

with ϵ representing a residual term, and f is the function parameterized by θ. This nonlinear function can be parameterized in various ways.

In a neural network, the approach is to use multiple layers of linear regression models and nonlinear activation functions. This process is detailed below. For convenience, z is defined as the output without the residual ϵ:

$$z = f(x_1, \ldots, x_k; \theta). \tag{6.6}$$

We begin by defining f as a linear regression model:

$$z = \beta_0 1 + \eta_1 x_1 + \eta_2 x_2 + \cdots + \eta_k x_k. \tag{6.7}$$

The output z is defined as the sum of all terms $\eta_i x_j$. The value 1 is used as the input variable corresponding to the offset or bias term β_0.

To model the relationships between $\mathbf{x} = [1 \; x_1 \; x_2 \; \ldots \; x_k]^T$ and z, a nonlinear, scaler *activation function* $\Phi : \mathbb{R} \to \mathbb{R}$ is introduced.

Equation (6.7) is now transformed into a generalized linear regression model, where the linear combination of inputs is passed through the defined activation function:

$$z = \Phi\left(\beta_0 + \eta_1 x_1 + \eta_2 x_2 + \cdots + \eta_k x_k\right). \tag{6.8}$$

The adapted visualization of this process is depicted below, drawing upon guidance from Wrigglesworth (2021) and Lindholm et al. (2019).

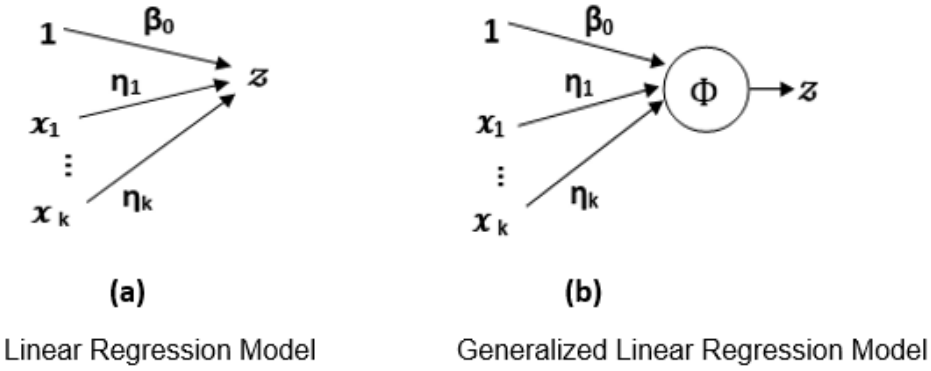


Figure 6.3: Generalized Linear Model

The following section explores the various options for the activation function, denoted as Φ. A thorough examination of these activation functions is crucial, as it informs the selection of parameters for our "ideal parameters" scenario. This scenario aims to rigorously assess whether the choice of input parameters significantly impacts the performance of neural networks, particularly in the context of option pricing.

6.3.2 Activation Function

If we recall, the activation function enables the network to learn complex patterns in the data by introducing non-linearity in the model. It is applied to the output of each neuron to ascertain whether the neuron should be activated, based on whether its input satisfies a specified threshold. The three essential purposes of using an activation function are provided below by drawing upon research by Jain (2024):

- Introduce Non-linearity: Incorporating non-linearity enables a neural network to capture and represent intricate and complex patterns within the data. This enhancement significantly improves both accuracy and efficiency. Without non-linearity, the model's capabilities would be limited to modeling only linear relationships.
- Differentiability: The differentiability of most activation functions is necessary for the backpropagation process. Backpropagation utilizes the derivative of the activation function to iteratively adjust the connection weights in each layer, thereby minimizing the error. A more detailed discussion of the backpropagation algorithm will be provided in a subsequent section.
- Controlling Output Range: Bounded output can assist with stabilizing learning of the model, and prevent extreme values in the network. This is seen with the activation functions hyperbolic tangent (tanh) and sigmoid, where their outputs are constrained to a certain range/bounds, namely sigmoid:(0,1) and tanh(-1,1).

During the training phase, the backpropagation algorithm is employed to compute gradients, facilitating the minimization of the loss function. As a result, continuous and differentiable activation functions are typically more suitable. The choice of activation function significantly

influences the network's capacity to learn underlying patterns. The loss function serves as a guiding mechanism, directing the adjustment of the network's parameters during training to effectively reduce error. A comprehensive list of activation functions and their corresponding properties is available in the Appendix, obtained from Pu (2021) and Pakkanen, et al., (2021), which will serve as a reference throughout this section.

One of the activation function we will be using in this dissertation is ReLU(rectified linear unit):

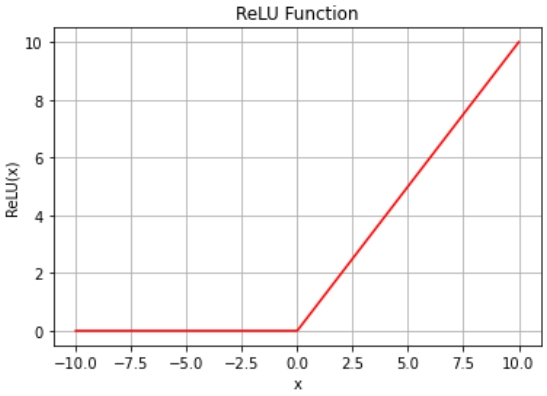


Figure 6.4: Graph of the Rectified Linear Unit Activation Function (ReLU)

$$\textbf{Definition} : \Phi(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0. \end{cases} \tag{6.9}$$

$$\textbf{Derivative} : \Phi'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0. \end{cases} \tag{6.10}$$

$$\textbf{Range} : [0, \infty).$$

1. ReLU: Rectified Linear Unit

The discussion surrounding the rectified linear unit activation function (ReLU) draws upon the research provided by Jain (2024) and Piepenbreier (2023).

ReLU only activates neurons when the input is positive. The output is set to zero for negative inputs resulting in sparse activation. This essentially helps reduce the vanishing gradient problem, which is encountered with activation functions such as sigmoid and tanh.

The vanishing gradient problem is a significant challenge that occurs when training deep neural networks. It is commonly seen when using specific activation functions such as sigmoid and

tanh due to the limited range. It arises when the gradients (the partial derivative of the loss function with respect to the model's parameters) become exceedingly small as they propagate backwards through the layers during the backpropagation process. This negatively impacts the training of the model. The updates to the weights become exponentially small (near-zero) and can either prolong the training time, or halt the learning process for those layers. The small gradients multiply between layers and decay significantly. The first layer will 'tear' away from the network or become ineffective, and this results in slower convergence. Overall, training for deep networks becomes slow and resource-intensive. The layers close to the output can learn and adapt, however the layers closer to the input stagnate. This hinders the networks ability to capture complex patterns. This issue is more prominent in deep networks compared to shallow networks.

The ReLU activation function helps alleviate the vanishing gradient problem by providing a consistent gradient of 1 for positive input values. This enables the straightforward updating of the weights. An additional advantage of employing ReLU over other activation functions is its simplicity. The function requires minimal computation which accelerates the model training. This efficiency is a crucial consideration, especially when working with a large dataset or complex structures such as deep neural networks.

2. Leaky ReLU: Leaky Rectified Linear Unit

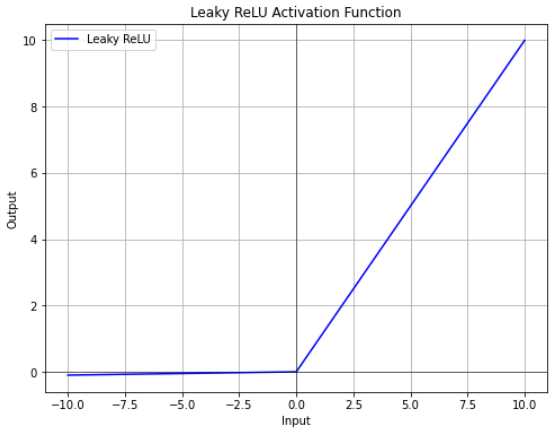


Figure 6.5: Graph of the Leaky Rectified Linear Unit Activation Function (LeakyReLU)

$$\textbf{Definition: } \Phi(x) = \text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0. \end{cases} \tag{6.11}$$

$$\textbf{Derivative: } \Phi'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x \leq 0. \end{cases} \tag{6.12}$$

$$\textbf{Range: } (-\infty, \infty).$$

A disadvantage of using ReLU is that it will not allow negative values to pass through. For negative input values, the ReLU activation function will provide a gradient of 0, which is ignored. The zero derivative of ReLU for negative arguments can be attributed to a number of reasons, as covered extensively by Aggarwal (2018).

Take the example where the neuron input is always nonnegative, yet all the weights have been set to negative values. In this case, the output will consistently be zero. Another instance occurs when a high learning rate is applied: this can cause the ReLU pre-activation values to reach a range where the gradient becomes zero, regardless of the input. Essentially, high learning rates can "disable" or deactivate ReLU units. When this happens, the ReLU fails to activate for any data sample. When a neuron is in this state, the gradient of the loss with respect to the weights preceding the ReLU will permanently be zero, preventing further updates to those neuron weights during training. Consequently, its output will be invariant to different inputs and will not contribute to distinguishing between instances. This is referred to as the 'dying/dead ReLU' problem. A way to fix this, is to use the "Leaky ReLU" activation function.

The Leaky ReLU function is a variation of ReLU that is used to address the dying ReLU problem. It does this by introduction of a small, non-zero gradient for negative inputs. This enables some information to pass through during backpropagation, even when there is negative input. The neurons will continue to learn even after receiving negative input values. Leaky ReLU activation function retains the benefits of ReLU, such as its straightford design and computational efficiency, while correcting any neuron activity. It is useful for DNN and ensures continouse learning of the neurons. In this dissertation we will be considering the performance of neural networks using the ReLU and LeakyReLU activation function, due to their efficiency and effectiveness.

3. Hyperbolic Tangent (tanh)

To discuss the hyperbolic tanh activation function, we will be drawing on research by Jain (2024) and Piepenbreier (2023).

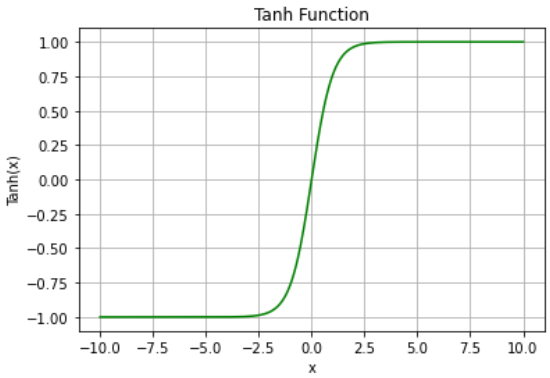


Figure 6.6: Graph of the hyperbolic tangent (tanh) activation function

$$\textbf{Definition} : \Phi(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{6.13}$$

$$\textbf{Derivative} : \Phi'(x) = 1 - \Phi(x)^2. \tag{6.14}$$

$$\textbf{Range} : (-1, 1).$$

The hyperbolic tangent (tanh) function is also a popular activation function to use in neural networks, especially with hidden layers. Values are mapped between the range -1 to 1, and there are advantages to using the tanh activation function in some cirumstances.

The tanh function is non-linear, thus allowing neural networks to learn complex patterns. The tanh function is also centered around zero, which can help in reducing issues related to bias shifts. This allows for faster convergence in training. The zero-centered nature of the tanh function aids optimisation, compared to the sigmoid function. However, Tanh still suffers with the vanishing gradient problem especially with inputs far from zero. This is seen particularly in deep networks and learning is slowed down, or even halted. Nonetheless, tanh can often be more effective than the sigmoid activation function for hidden layers due to its broader output range. It's properties can be beneficial for neural networks that are shallow, or in specific architectures such as RNNs.

4. Sigmoid

To discuss the sigmoid activation function, we will be drawing on research by Jain (2024) and Piepenbreier (2023).

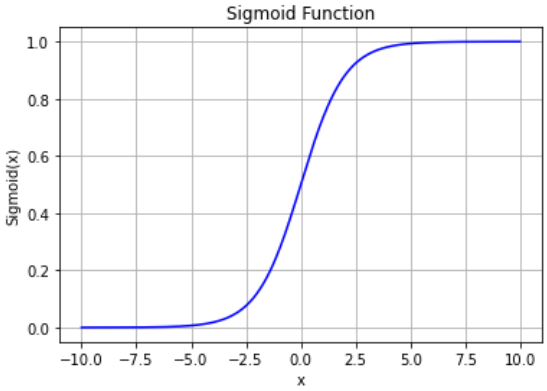


Figure 6.7: Graph of the Sigmoid Activation Function

$$\textbf{Definition} : \Phi(x) = \frac{1}{1 + e^{-x}}. \tag{6.15}$$

$$\textbf{Derivative} : \Phi'(x) = \Phi(x)(1 - \Phi(x)). \tag{6.16}$$

Range : $(0, 1)$.

The sigmoid activation function maps the input to an output range between 0 and 1, therefore making it appropriate for binary classification and probabilistic interpretations.

The sigmoid activation function was favoured in the past due to its smooth continuous output. However, as with the tanh activation function, for very large positive or negative values the gradient becomes increasingly small and approaches zero. This is the vanishing gradient problem, which makes it an unsuitable choice for deep learning models. Sigmoid outputs are also not zero-centered, which may present challenges during optimization. This could result in slower convergence due to the inefficient gradient updates.

It's properties makes it ideal for use in binary classification models, logistic regression, and in models where output is expected to be between 0 and 1. It may be used in certain layers of neural networks or in specific architectures wehre the vanishing gradient is not a significant concern.

4. Other Activation Functions

We will briefly discuss other activation functions. More information on the graphical depiction, full mathematical definition, derivative and range can be found in the Appendix (Pu, 2021). We will also be drawing on research from Jain (2024) and Piepenbreier (2023).

The Identity Id activation function essentially takes the output of each neuron as the input multiplied by weights. Therefore, it does not introduce non-linearity, and it will be unsuitable to use for complex patterns, such as in deep learning models. Note that as a result of the derivative being a constant, we also cannot consider backpropagation. It is used for regression tasks due to its simplicity and its unrestricted output values. It also does not have the vanishing gradient issue, as the derivative is always constant.

A similar situation occurs for the Heaviside (H) activation funtion where the derivatives are zero, otherwise undefined. The discontinuity and non-differentiability at certain points prevents it from being the ideal choice of an activation function.

Using our above findings, our ideal choices are the ReLU and LeakyReLU activation functions.

If we recall from the universal approximation theorem discussed by Hornik et al. (1989), provided that specific conditions regarding activation functions and network depth are satisfied, a neural network can approximate any continuous real-valued function on a compact set with arbitrary precision $\epsilon > 0$. This is referred to as the depth-bounded universal approximation theorem, which essentially focuses on how deep neural networks can approximate complex functions more accurately by stacking layers due to a certain depth requirement to ensure approximation.

Further research regarding the universal approximation theorem, specifically for ReLU activation functions, are given by Stinchcombe et al. (1989). Additionally, work by Lu et al. (2017) proposed the concept of a specific universal approximation function that, with ReLU activation functions, provides an upper bound to the width. This is an important result for this dissertation as we use the ReLU activation function variations in our neural networks.

The theorem is stated below:

Theorem 3: Universal Approximation Theorem (Width-Bounded) for ReLU Networks - (Anderson & Ulrych, 2023)

For any Lebesgue-integrable function $f : \mathbb{R}^n \to \mathbb{R}$ and any $\epsilon \geq 0$, $\exists$ a fully-connected ReLU network G with a width $d_L \leq n + 4$, such that the neural network function F_G approximating f satisfies:

$$\int_{\mathbb{R}^n} |f(x) - F_G| dx < \epsilon. \tag{6.17}$$

This theorem demonstrates that even with constraints on depth, a ReLU network can approximate a function to an arbitrary accuracy, provided it has sufficient width. This reinforces the idea that ReLU networks are particularly powerful approximators, even with width constraints.

From equation (6.8), we see that the generalized linear regression model is quite simple and on its own lacks the ability to represent complex relationships between the input $\mathbf{x}$ and the output $\mathbf{z}$. We implement two additional modifications in order to improve the models generalization. First, we employ multiple generalized linear regression models to construct a layer, giving rise to the multi-layer neural network. Next, these layers are stacked sequentially, resulting in a deep feed-forward neural network. By use of sequential stacking we ensure that the nodes in each layer are systematically connected to the nodes in the following later, as opposed to random connections. By 'feed-forward' we refer to the process where the input of each neuron in one layer serves as the input to the neurons in the subsequent layer.

To understand this visually, the diagram below adapted from Huang et al. (2022) depicts a perpetron consisting of two layers, namely a neuron and its connections. Each layer is composed of a group of nodes that aggregate all weighted inputs, applies an activation function, and produce a single output (as illustrated in Figure 6.8). Therefore, each node functions is a generalized linear model. The inputs are passed into the input layer, $x_1, x_2, ..., x_k$, with associated weights given by $\eta_1, \eta_2, ..., \eta_k$. These weights represent the 'strength' of the signal. Once aggregated, an activation function of choice is applied. The output will then become an input of another node in the following layer.

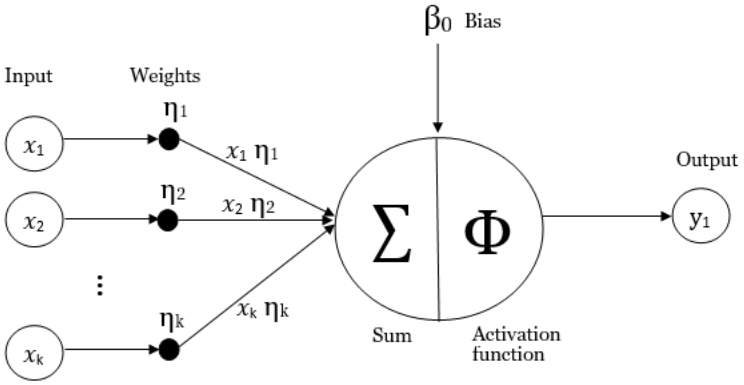


Figure 6.8: Flow of inputs through a neuron

To describe the process we draw on research by Wrigglesworth (2021). The intermediate layer between the input layer and output can also be referred to as a hidden layer. Input variables are received by each of the M nodes in the hidden layer. These inputs are received from the preceeding layer (input layer), weights are applied, and the generalized linear model (GLM) aggregates them with a unique set of weights and bias parameters for that node. The node then applies an activation function, and a single output is generated. This output then serves as input to each node in the subsequent layer, which has its own distinct set of parameters.

In Equation (6.8) above, the output is generated by a single scaler regression model. Let the output be a sum of M GLMs, each with its own set of parameters. The parameters for the i-th GLM of layer one are: $\beta_{0i}, \eta_{1i}, \ldots, \eta_{ki}$.

The output is denoted by h_i:

$$h_i = \Phi\left(\beta_{0i} + \eta_{1i}x_1 + \eta_{2i}x_2 + \cdots + \eta_{ki}x_k\right), \quad i = 1, \ldots, M. \tag{6.18}$$

The intermediate outputs h_i are known as the *hidden units*, and the M units $\{h_i\}_{i=1}^{M}$ serve as input variables to the following layer:

$$z = \beta_0 + \eta_1 h_1 + \eta_2 h_2 + \cdots + \eta_M h_M. \tag{6.19}$$

To differentiate between the layer equations above, we introduce superscripts (1) and (2), for layers 1 and 2 respectively. These equations describe a two-layer neural network (or alternatively, a neural network with one layer of hidden units), and will have increased flexiblility.

Therefore, we have:

$$\begin{aligned}
h_1 &= \Phi\left(\beta_{01}^{(1)} + \eta_{11}^{(1)}x_1 + \eta_{21}^{(1)}x_2 + \cdots + \eta_{k1}^{(1)}x_k\right), \\
h_2 &= \Phi\left(\beta_{02}^{(1)} + \eta_{12}^{(1)}x_1 + \eta_{22}^{(1)}x_2 + \cdots + \eta_{k2}^{(1)}x_k\right), \\
&\vdots \\
h_M &= \Phi\left(\beta_{0M}^{(1)} + \eta_{1M}^{(1)}x_1 + \eta_{2M}^{(1)}x_2 + \cdots + \eta_{kM}^{(1)}x_k\right), \qquad (6.20) \\
z &= \beta_0^{(2)} + \eta_1^{(2)}h_1 + \eta_2^{(2)}h_2 + \cdots + \eta_M^{(2)}h_M. \qquad (6.21)
\end{aligned}$$

To summarize the structure to this point, we start with an input layer, $\{x_i\}_{i=1}^{k}$ that consists of k inputs. Proceeding from this layer, there is a hidden layer (represented by the superscript (1)), which contains M nodes, or hidden units. Each hidden unit is an individual GLM model with its unique set of weights and bias parameters, represented collectively by $\theta_i^{(1)}$. The hidden layer is characterized by the parameter vector $\theta^{(1)} = \left[\theta_1^{(1)}, \ldots, \theta_M^{(1)}\right]$, and each node in this layer uses the activation function Φ. Each input layer node connects to each hidden unit in the hidden layer. After the hidden layer, there is an output layer consisting of a single node (as in the given example). It is also a GLM with its own activation function and a distinct parameter set $\theta^{(2)}$. The hidden units within the hidden layers are connected to the output node, and are weighted and aggregated across all inputs from the hidden units. The activation function is applied, and the final output of the network is produced.

The figure below illustrates the process, and is adapted from Lindholm et al.(2019).

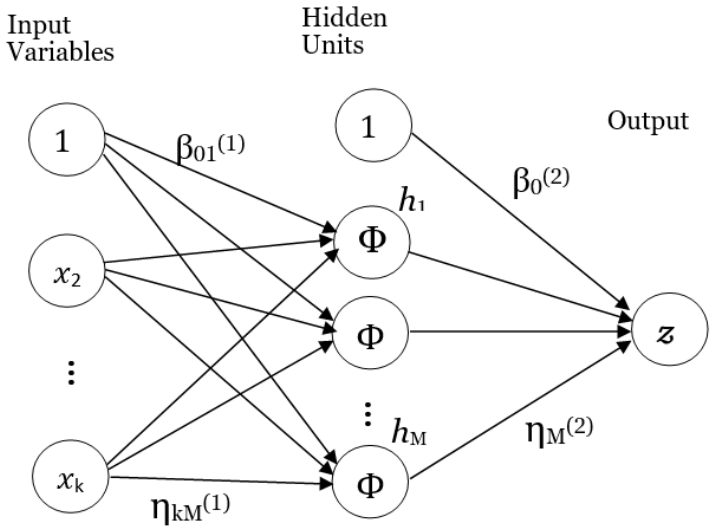


Figure 6.9: Multi-Layered Neural Network

6.3.3 Matrix Notation

The compact matrix notation depicted below is an adaptation from Lindholm et al. (2019) and Wrigglesworth (2021). This will assist us in forming a general theorem for neural networks.

$$\mathbf{b}^{(1)} = \begin{bmatrix} \beta_{01}^{(1)} & \cdots & \beta_{0M}^{(1)} \end{bmatrix}, \quad \mathbf{W}^{(1)} = \begin{bmatrix} \eta_{11}^{(1)} & \cdots & \eta_{1M}^{(1)} \\ \vdots & \ddots & \vdots \\ \eta_{k1}^{(1)} & \cdots & \eta_{kM}^{(1)} \end{bmatrix}, \quad \mathbf{b}^{(2)} = \begin{bmatrix} \beta_0^{(2)} \end{bmatrix}, \quad \mathbf{W}^{(2)} = \begin{bmatrix} \eta_1^{(2)} \\ \vdots \\ \eta_M^{(2)} \end{bmatrix}, \tag{6.22}$$

where $\mathbf{W}$ is a *weight matrix* and $\mathbf{b}$ is the *bias vector*.
The full model is given as:

$$\mathbf{h} = \Phi\left(\mathbf{W}^{(1)\top}\mathbf{x} + \mathbf{b}^{(1)\top}\right), \tag{6.23}$$

$$z = \mathbf{W}^{(2)\top}\mathbf{h} + \mathbf{b}^{(2)\top}, \tag{6.24}$$

where the components of $\mathbf{x}$ and $\mathbf{h}$ have been organized into vectors, depicted as $\mathbf{x} = [x_1, \ldots, x_p]^\top$ and $\mathbf{h} = [h_1, \ldots, h_M]^\top$ with Φ acting as our element-wise activation function.

The parameters of the model are depicted as:

$$\theta = \begin{bmatrix} \text{vec}(\mathbf{W}^{(1)})^\top & \text{vec}(\mathbf{W}^{(2)})^\top & \mathbf{b}^{(1)\top} & \mathbf{b}^{(2)\top} \end{bmatrix}^\top. \tag{6.25}$$

This completes the description of a nonlinear regression model in the form:

$$y = f(\mathbf{x}; \theta) + \epsilon.$$

6.3.4 Deep Neural Network Structure

Having explored the various structural components of a neural network, including its matrix notation, we now possess the foundational knowledge required to delve into deep neural networks. The comparison between the performance of artificial "shallow" neural networks and deep neural networks will serve as a pivotal aspect in achieving our objectives. Expanding on the structural and diagrammatic representation of deep neural networks is necessary for this analysis.

Building upon the aforementioned model architecture, we can now explore more intricate structures—namely, deep neural networks. These are fundamentally created by stacking multiple layers of generalized linear models (GLMs) to construct a deeper architecture. In deep neural networks, the depth of the network increases with the addition of multiple layers. A neural network is classified as "deep" when it comprises two or more hidden layers.

We start the process by enumerating the layers of the model with the index l. Each layer has defined parameters with a weight matrix $\mathbf{W}^{(l)}$ and a bias vector $\mathbf{b}^{(l)}$. Each layer is constructed from M_l hidden units $\mathbf{h}^{(l)} = [h_1^{(l)}, \ldots, h_{M_l}^{(l)}]^\top$.

Mapping from hidden layer $(l-1)$ to (l) is depicted as:

$$\mathbf{h}^{(l)} = \Phi\left((\mathbf{W}^{(l)})^\top \mathbf{h}^{(l-1)} + (\mathbf{b}^{(l)})^\top\right). \tag{6.26}$$

The output of each hidden layer is the input of the succeeding hidden layer e.g. output of $\mathbf{h}^{(1)}$ is the input to $\mathbf{h}^{(2)}$

We have constructed a DNN of L layers:

$$\begin{aligned}
\mathbf{h}^{(1)} &= \Phi\left((\mathbf{W}^{(1)})^\top \mathbf{x} + (\mathbf{b}^{(1)})^\top\right), \\
\mathbf{h}^{(2)} &= \Phi\left((\mathbf{W}^{(2)})^\top \mathbf{h}^{(1)} + (\mathbf{b}^{(2)})^\top\right), \\
&\vdots \\
\mathbf{h}^{(L-1)} &= \Phi\left((\mathbf{W}^{(L-1)})^\top \mathbf{h}^{(L-2)} + (\mathbf{b}^{(L-1)})^\top\right), \\
\mathbf{z} &= \mathbf{W}^{(L)\top} \mathbf{h}^{(L-1)} + \mathbf{b}^{(L)\top}.
\end{aligned} \tag{6.27}$$

Lastly, it is necessary to consider the varying dimensions of the parameters in each layer. Note that, in deep learning it is typical to have multi-dimensional outputs, so $\mathbf{z}$ could be $\mathbf{z} = [z_1, ..., z_P]$. This is summarized below as follows:

Layer	Weight Matrix	Offset Vector
$l = 1$	$\mathbf{W}^{(1)} : k \times M_1$	$\mathbf{b}^{(1)} : 1 \times M_1$
$l = 2, ..., L-1$	$\mathbf{W}^{(l)} : M_{l-1} \times M_l$	$\mathbf{b}^{(l)} : 1 \times M_l$
$l = L$	$\mathbf{W}^{(L)} : M_{L-1} \times P$	$\mathbf{b}^{(L)} : 1 \times P$

Table 6.1: Dimensions of the Parameter

Using the above information, we have a formalised definition of neural networks:

Definition 5: Neural Network - (Lu & Lu, 2020)
A feed-forward connected neural network of L hidden layers takes an input vector $\mathbf{x} \in \mathbb{R}^n$, outputs a vector $\mathbf{z} \in \mathbb{R}^K$, and has L hidden layers of sizes $M_1, M_2, \ldots, M_L$. The neural network is parametrized by the weight matrices $\mathbf{W}^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$ and bias vectors $\mathbf{b}^{(l)}$ with $l = 1, 2, \ldots, L+1$. The output $\mathbf{z}$ is defined from the input $\mathbf{x}$ iteratively according to the following:

$$\begin{aligned} \mathbf{h}^{(0)} &= \mathbf{x}, \\ \mathbf{h}^{(l)} &= \Phi\left((\mathbf{W}^{(l)})^\top \mathbf{h}^{(l-1)} + (\mathbf{b}^{(l)})^\top\right), \quad 1 \le l \le L, \\ \mathbf{z} &= \mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + (\mathbf{b}^{(L+1)})^\top. \end{aligned} \tag{6.28}$$

Here, Φ is a (nonlinear) activation function which acts on a vector $\mathbf{x}$ component-wise, i.e., $[\Phi(\mathbf{x})]_i = \Phi(x_i)$. When $M_1 = M_2 = \cdots = M_L = M$, we say the network has width M and depth L.
The neural network is said to be a deep neural network (DNN) if $L \ge 2$. The function defined by the deep neural network is denoted by $\mathrm{DNN}(\{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{L+1})$.

6.3.5 Graphical Differences Between ANN and DNN

In the subsequent section, we will examine the structural differences between artificial neural networks and deep neural networks. This comparison is essential as it provides context for understanding potential variations in performance and predictive accuracy between the two. Additionally, it lays the groundwork for addressing a critical, and arguably the most significant, objective: evaluating the computational efficiency of neural networks in comparison to traditional methods.

Artificial 'shallow' neural networks typically consist of three layers: an input layer, a single hidden layer, and an output layer. Their relatively limited depth makes them more suitable for less complex tasks, and they can be visualized with just one hidden layer connecting the input and output layers. In contrast, deep neural networks are defined by the presence of multiple hidden layers, which significantly increases their depth. The enhanced DNN structure allows the modelling of more complex patterns and representations. They can be visualized with several

hidden layers, emphasizing their greater complexity and capacity for more advanced tasks. Due to the greater complexity of deep neural networks, we anticipate that the approximations using deep neural networks may be more accurate. However, we also note that due to the more complex architecture, we anticipate increased computational usage and increased performance time for training the model, as studied intensily by Bianco et al. (2018), and Cheng et al. (2017).

The figures below depicting the structural differences are drawn, drawing upon guidance from Alves de Oliveira et al. (2023).

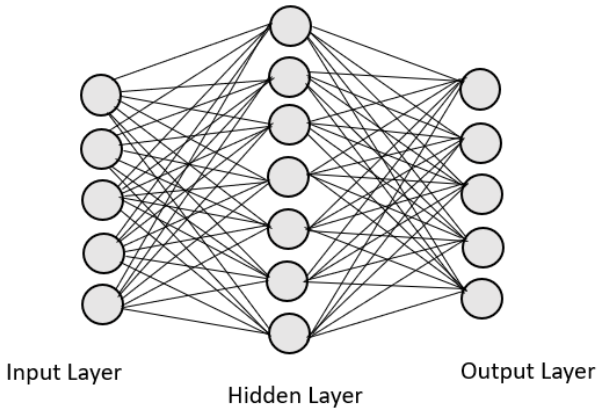


Figure 6.10: ANN 'Shallow' Diagram

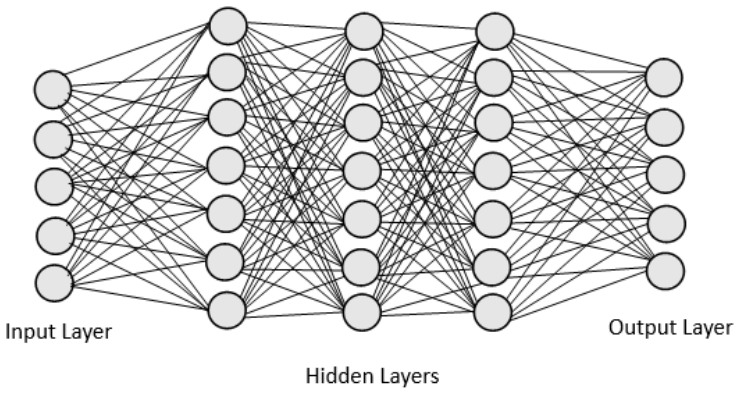


Figure 6.11: DNN Diagram

6.3.6 Loss Function

Peter Drucker, a famous businessman, is believed to have said

"You can't manage what you can't measure."

This powerful expression is relevant, especially in the case of neural networks. Our best approach to managing the effectiveness of the neural network, is to manage its performance in predicting the target values in the training data. This is done through the loss function. In this dissertation, our analysis of the loss function is important to neural networks as it serves as a key component for guiding the optimization process, thus enabling the network to learn by minimizing errors and improving its predictive performance over successive iterations. We will be drawing upon the research by Terven et al. (2023). A full list of other loss functions and their corresponding equations as depicted by Pu (2021) and Pakkanen et al. (2021) can be found in the Appendix. We will primarily be discussing the mean squared error and mean absolute error loss function.

The loss function quantifies how effectively the model's predicted values compare to the target output. It is essentially a measurable way of assessing the performance and the accuracy of a machine learning model. During the training stage, the network iteratively adjusts its parameters (weights and biases) in order to minimize the loss function, which would essentially guide the model to improve its predictions. The loss function essentially serves as a feedback mechanism, indicating the extent of deviation between the predictions and the actual results (prediction error). A lower loss value implies that the model is performing well and is desired. A higher loss function suggests a need for improvement through additional adjustments. Through the loss minimisation process the network learns patterns in the data helping it to generalize better on unseen data. An effective loss function would also assist in the managing of the trade-off between bias and variance (overfitting). This is also a critical step in ensuring the model's ability to generalize on unseen data.

Our chosen loss function is the mean squared error (MSE) loss function defined below:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(f(x_i) - y_i\right)^2. \tag{6.28}$$

This calculates the average squared differences between the predicted outputs and actual outputs.

Given input x_i the output $f(x_i)$ should ideally be as close as possible to the target y_i.

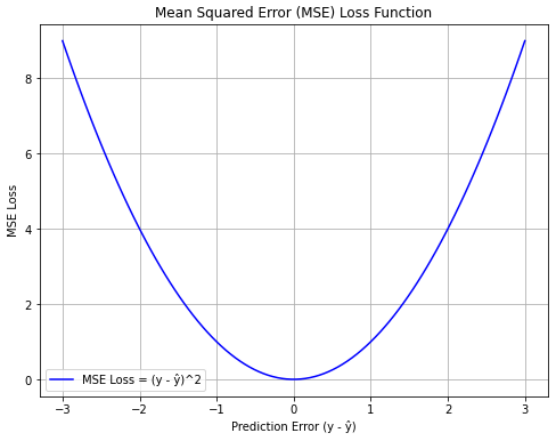


Figure 6.12: Mean Squared Error loss function

The MSE loss function holds the following properties Terven et al. (2023):

Property	**Description**
Non-negative	The MSE is always non-negative as it squares the differences between predicted and actual values. Higher values reflect greater discrepancies between predictions and actual outcomes. A perfect fit would be denoted by 0.
Outliers Sensitive	The MSE emphasizes larger errors more than smaller ones (quadratic growth), making it sensitive to outliers. As a result, outliers are given greater emphasis during training, which disproportionately affects the training outcome
Differentiable	MSE is a smooth and continuous function which enables efficient gradient computation. This differentiability is essential for optimization methods like gradient descent.
Convex	MSE is convex, having a unique global minimum. This trait facilitates the optimization process, enabling gradient-based optimization methods to converge to the global minimum without becoming caught in local minima.
Scale-dependent	The MSE value is dependent on the scale of the target variable, which complicates model performance comparisons across varying problems or scales.

Table 6.2: Properties of the Mean Squared Error (MSE) Loss Function

Its applications include: linear Regression, neural networks, support vector regression, gradient boosting etc.

While mean squared error loss or L_2 loss is a widely implemented method, the susceptibility to outliers is a limitation. Another loss function that is more robust to outliers is the mean absolute error (MAE) loss function

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|f(x_i) - y_i|. \tag{6.29}$$

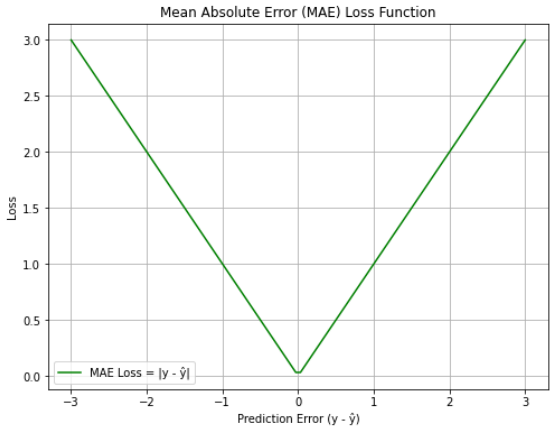


Figure 6.13: Mean Absolute Error Loss Function

Properties of the mean absolute error are discussed below.

Similar to the MSE, the MAE is also always non-negative with a perfect fit depicted by a value of 0. MAE is less sensitive to outliers compared to MSE, and all errors of all magnitudes are treated uniformly. This property makes MAE a more robust loss function than MSE, and is more suitable for scenarios where there are outliers. However, whilst MAE is a continuous function, it is non-differentiable at a prediction error of zero because of the presence of the absolute value function. The gradient of the absolute loss function is constant, and this is not ideal for gradient-based training and the optimization process. This makes MAE a less optimal choice for use with neural networks.

Other loss functions include the Hinge Loss function, and the Binary Cross-entropy Loss function, however these are more suitable for Binary Classification problems.

6.4 Training the Network

The training process forms the backbone of ensuring reliable and accurate predictions of neural networks. By drawing on the research by Lindholm et al. (2019) and Wrigglesworth (2021), this section outlines the methodology for estimating the parameter set θ which is a crucial step for calibrating the neural network to accurately model the option pricing dynamics. By defining the optimization problem the process can subsequently integrate numerical techniques, such as gradient descent, to iteratively refine θ and minimize the associated loss function. This section outlines the key steps: parameter initialization, iterative optimization, and stopping conditions, and these are integral to the backpropagation process. With backpropagation we compute the gradients of the loss function with respect to the parameters, and the optimization steps detailed below utilize these gradients to iteratively update the parameters. This will be discussed in the section to follow after outlining the optimization proces. By implementing the steps below we aim to ensure the network learns effectively and converges to a solution that we anticipate is capable of capturing the complexities of option pricing.

In order to define a model capable of producing reliable predictions, it is necessary to estimate

θ. This estimation problem can be formulated as:

$$\hat{\theta} = \arg\min_{\theta} J(\theta) \quad \text{where} \quad J(\theta) = \frac{1}{p}\sum_{i=1}^{p} L(X_i, y_i; \theta), \tag{6.30}$$

with $J(\theta)$ the cost function, $L(X_i, y_i; \theta)$ the loss function, and $arg\min_\theta J(\theta)$ the parameter value $\hat{\theta}$ that minimizes function $J(\theta)$ (Lindholm, 2019).

From the above equation, we note that J is being determined by the specific details of the problem, and $L(X_i, y_i; \theta)$ a function that measures the error for each data point (X_i, y_i) based on current set of parameters θ. The optimization problem in Eq. (6.30) cannot be solved in closed form and would require us to implement numerical optimization texhniques. Examples of this specific to deep neural networks include gradient descent, and stochastic gradient descent

The high level overview outlining the process is summarized as follows:

1. **Initialize parameters**: We begin by initializing θ_0. These initial values are chosen randomly to help the network learn diverse features from the data.

2. **Iterative Optimization**: After initialization, we update θ step by step to gradually reduce $J(\theta)$:

$$\theta_{t+1} = \theta_t - \lambda \nabla_\theta J(\theta_t), \quad \text{for } t = 1, 2, \ldots \tag{6.31}$$

   with learning rate λ.

3. **Stopping Condition**: After achieving a level of convergence, the process is halted with θ set to the final value θ_t. At this point, θ_t becomes the optimized θ for the network.

Selecting an initial parameter set θ_0 presents challenges, as the cost functions associated with neural network training are not necessarily convex. Non-convexity implies that training is highly sensitive to θ_0; convexity would be ideal as it guarantees convergence irrespective of θ_0. As noted by Brownlee (2019), when using the ReLU activation function, elements of θ_0 are typically initialized to small and non-negative values to facilitate operation within the positive domain of the ReLU activation function.

Backpropagation and Stochastic Gradient Descent

6.4.1 Backpropagation

Backpropagation is an important step in training process of neural networks. It enables neural networks to learn from data by minimizing prediction errors. By computing the gradients of the loss function with respect to the network's parameters, backpropagation thus provides the necessary information for optimizing weights and biases. This iterative process ensures that the neural network adjusts its internal structure to better capture any complex patterns, making it an essential component in option pricing. We will draw from research by Rumelhart et al. (1986) and Goodfellow et al. (2016).

When a feed-forward NN is used to take an input $\mathbf{x}$ and produce output $\hat{y}$, information propagates forward through the network. The input values x provide the initial values that are then passed through the hidden units in each layer, ultimately producing $\hat{y}$.

This process is known as *forward propagation*.

During training, the forward propagation process proceeds until it yields a scaler cost, $J(\theta)$. The *backpropagation* algorithm enables the flow of information from the cost function backward through the network, allowing for gradient computation. While calculating an analytical expression for the gradient is a simple process, the numerical evaluation can be quite computationally intensive. The backpropagation algorithm provides a straightforward and cost-efficient approach to this calculation. backpropagation refers specifically to the method of computing gradients. An additional algorithm, such as the stochastic gradient descent algorithm, utilizes these gradients for learning. Additionally, backpropagation can compute derivatives of any function (for certain functions, we indicate that the derivative is undefined).

We are interested in the process of computing the gradient $\nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{y})$ for an arbitrary function f, where $\mathbf{x}$ represents a set of variables with required derivatives, and $\mathbf{y}$ includes additional variables that are inputs to the function but do not require derivatives. The gradient often needed is that of the cost function with respect to the parameters, $\nabla_\theta J(\theta)$.

Numerous machine learning tasks necessitate the computation of derivatives, whether for optimizing the learning process or conducting model evaluations. The backpropagation algorithm extends beyond merely determining the gradient of the cost function with respect to model parameters and can also be applied to a broader range of derivative calculations. The methodology of propagating information through a neural network to derive such values is highly versatile, facilitating the computation of advanced mathematical constructs, such as the Jacobian matrix of a function f with multiple outputs.

The backpropagation (BP) algorithm is used in the training step of neural networks. It is a method for adjusting the weights of the connections between neurons, moving backwards through the network in order to minimize the difference between the actual and desired output. We use it to calculate the gradient of the loss function with respect to the weights of the network, which allows for more efficient optimization of gradient descent methods. We use these methods to minimize our loss by updating our weights using the chain rule. We then calculate the gradient for each layer and move from the last layer backwards. By use of the chain rule we can depict our error gradients as a sum of local gradient products over the many paths.

The full backpropagation process is concisely outlined below, using the variables and notations defined thus far:

Step 1: Forward Passing Phase

1. The input data is fed into the neural network, and the calculations are done layer by layer to generate predictions.
2. For each layer l, the pre-activation $z^{(l)}$ is computed as:

$$z^{(l)} = W^{(l)} h^{(l-1)} + b^{(l)}, \tag{6.32}$$

where:

- $W^{(l)}$ is the weight matrix for layer l,
- $h^{(l-1)}$ is the activation output from the previous layer $l-1$,
- $b^{(l)}$ is the bias vector for layer l,

3. The activation $h^{(l)}$ is then calculated by applying an activation function Φ to $z^{(l)}$:

$$h^{(l)} = \Phi(z^{(l)}). \tag{6.33}$$

Step 2: Loss Calculation

1. The network output $\hat{y}$ is compared to the actual target y using a loss function $J(\theta)$.
2. For Mean Squared Error (MSE) loss, the function is defined as:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)^2, \tag{6.34}$$

where:

- N is the number of training examples,
- $\hat{y}^{(i)}$ is the predicted output for the i-th example,
- $y^{(i)}$ is the actual output for the i-th example.

Step 3: Backward Pass Phase (BP)

1. Calculate the gradient of the loss function $J(\theta)$ with respect to each parameter using the chain rule.
2. Starting from the output layer, we compute the error term $\delta^{(L)}$ for each layer L as:

$$\delta^{(L)} = \frac{\partial J}{\partial z^{(L)}} = (\hat{y} - y)\,\Phi'\left(z^{(L)}\right). \tag{6.35}$$

3. For each hidden layer l, propagate the error backwards:

$$\delta^{(l)} = \left(W^{(l+1)}\right)^T \delta^{(l+1)} \circ \Phi'\left(z^{(l)}\right), \tag{6.36}$$

where $\circ$ denotes the element-wise product, and $\Phi'\left(z^{(l)}\right)$ is the derivative of the activation function.
4. The gradients of the weights $W^{(l)}$ and biases $b^{(l)}$ are computed as follows:

$$\frac{\partial J}{\partial W^{(l)}} = \delta^{(l)} \cdot \left(h^{(l-1)}\right)^T \tag{6.37}$$

$$\frac{\partial J}{\partial b^{(l)}} = \delta^{(l)}. \tag{6.38}$$

Step 4: Weight Adjustment

1. Update the weights and biases using the computed gradients and a learning rate λ:

$$W^{(l)} := W^{(l)} - \lambda\frac{\partial J}{\partial W^{(l)}} \tag{6.39}$$

$$b^{(l)} := b^{(l)} - \lambda\frac{\partial J}{\partial b^{(l)}}. \tag{6.40}$$

This step is performed to minimize the loss function by moving the parameters in the opposite direction of the gradient.

Step 5: Iterative Optimization

1. Repeat the forward and backward passess for a specified number of epochs, or until convergence occurs. 2. In each iteration (or batch, if using mini-batch gradient descent - to be discussed below), the weights and biases are updated to minimize the loss.

To ensure effective training, the convergence of the model should be monitored by tracking the loss or a relevant validation metric. If the loss fails to improve over multiple iterations early stopping can be employed to terminate the training process thus preventing overfitting. Once training is complete the final step involves evaluating the network on a separate test dataset to assess its generalization performance and ability to make accurate predictions on unseen data. The gradient descent technique will be discussed in more detail below.

6.4.2 Stochastic Gradient Descent

To recap, gradient descent is a key optimization algorithm used in training neural networks which enables the adjustment of weights and biases to minimize prediction errors. For option pricing gradient descent ensures that the network learns to accurately model complex pricing relationships by iteratively reducing the loss function. In this section, we draw upon the work by Lindholm et al. (2019), Liu (2022) and Wrigglesworth (2021). Other sources will be noted in context.

Gradient descent is an iterative process used to determine the optimal values of the parameters θ which correspond to the minimum cost function $J(\theta)$ value. The derivative of the cost function at each parameter value will provide the sensitivity of the function with respect to that particular variable. Gradient descent allows the learning process to iteratively adjust the estimated parameters, guiding the model toward an optimal set of values. In gradient descent, we refer to one iteration as 'one batch'. This provides us with an indication of the total number of samples from the total dataset that is used to approximate the gradient for each iteration.

The application of the conventional gradient descent optimization technique to our dataset has limitations. Specifically, it becomes computationally expensive, as each iteration requires processing the entire large dataset of samples. This computational burden must be repeated for each iteration until the model converges to the minimum point, making it inefficient in terms of time and cost. This issue can be addressed by employing stochastic gradient descent (SGD).

The term *stochastic* refers to processes involving random probability. SGD leverages this idea to accelerate the process of gradient descent. Unlike traditional gradient descent, which relies on the entire dataset in each iteration, SGD only uses a single sample to compute the gradient for each update. While using the entire dataset in standard GD can facilitate a smoother convergence to the minimum, it becomes impractical with very large datasets. However, SGD offers a computationally improved solution for this issue. A key benefit of SGD is its ability to avoid being trapped in local minima, which can be a risk when using standard gradient descent. This is because stochastic gradient descent, by updating with individual samples, introduces randomness that can help the algorithm escape local minima and better approximate the global minimum.

There are 3 main variants of gradient descent:

- Batch Gradient Descent (BGD): This approach calculates the error for each training example, but updates the model parameters only after all examples have been processed in a given iteration.
- Stochastic Gradient Descent (SGD): In SGD, the model parameters are updated after computing the error for each individual training example. This method introduces more fluctuation in parameter updates but can help in escaping local minima.
- Mini-Batch Gradient Descent: The dataset is divided into small batches, each used to compute the error and update the model parameters. This approach balances computational efficiency and convergence stability, making it the most commonly used variant in deep learning.

The learning rate, as noted by Liu (2022), is the hyperparamater that controls the size of the steps taken towards the minimization of the cost function. It assesses the speed at which a model updates its parameters in response to the computed gradient of the loss function, with respect to the parameters θ.

In stochastic gradient descent (SGD), the 'true' or actual gradient is approximated by $\hat{g}_t$:

$$\theta_{t+1} = \theta_t - \lambda \hat{g}_t, \tag{6.41}$$

where $\hat{g}_t$ is the gradient at a single instance.
During the training process, the algorithm updates the parameters after completing each full pass through the training set. The total number of these full passes is referred to as the number of epochs depicted as E. The data is shuffled after each epoch.

The algorithm for stochastic gradient descent is:

Algorithm 3: Stochastic Gradient Descent Algorithm - (Lindholm et al. 2019)

---

1. In the network, initialise all the parameters θ and set $t = 1$; For $k = 1$ to $Epoch(E)$
2. a.) Shuffle the training data randomly $\{(x_i, y_i)\}_{i=1}^{k}$;
   b.) Estimate the gradient of the loss function use of the following equation:

$$\hat{g}_t = \frac{1}{k} \sum_{i=1}^{k} \nabla_\theta L(x_i, y_i; \theta)$$

   c.) Perform a gradient update step $\theta_{t+1} = \theta_t - \lambda \hat{g}_t$
   d.) Increment the iteration counter $t = t + 1$.

6.4.3 Mini-Batch Gradient Descent

We evaluate both standard stochastic gradient descent and minibatch gradient descent, comparing their performance to inform our objective of identifying the "ideal parameter" scenario. This analysis is grounded in the foundational work of Bengio (2012).

Mini-batch gradient descent is a variantion of the gradient descent algorithm where the training data is divided into smaller 'batches'. These mini-batches are then used to compute the prediction error of the model and subsequently update the model's parameters.

Consider the scenario where we are training a network with a very large dataset of size k. Running the backpropagation algorithm could be computationally expensive. To alleviate this, we can assume that many data points in the dataset are similar, allowing us to approximate the gradient for half of the dataset as being identical to the other half:

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \sum_{i=1}^{\frac{k}{2}} \nabla_\theta L(x_i, y_i, \theta) \\ &\approx \sum_{i=\frac{k}{2}+1}^{k} \nabla_\theta L(x_i, y_i, \theta),\end{aligned} \tag{6.42}$$

where all variables are defined previously.

Consequently, computing the gradient over the entire dataset is unnecessary. Alternatively, we calculate the gradient on the first half of the data, perform a parameter update, and then calculate the gradient on the second half of the data. The advantage of this is the computational time reduction of approximately half. A mini-batch size k_{mb}, represents the number of training examples per mini-batch. Thus, the gradient is updated based on the mini-batches:

$$\theta_{t+1} = \theta_t - \lambda \sum_{i=1}^{k_{mb}} \nabla_\theta L(x_i, y_i, \theta_t). \tag{6.43}$$

When using mini-batches, it is essential that each mini-batch is balanced and representative of the entire dataset. To achieve this we select $k_m b$ training samples randomly from the dataset to create each mini-batch. This can be achieved pratically by shuffling the training dataset randomly before dividing it into sequential mini-batches. We will discuss the advantges of using minibatch stochastic gradient descent below, drawing upon the research of Brownlee (2019), Lindholm, et al (2019) and Olamendy (2023)

Advantages of minibatch stochastic gradient descent:

- **Faster Training**: Minibatch updates are computationally more cost effective than full-batch updates, which enhances the training process speed.
- **Memory Efficiency**: Minibatches allow the processing of large datasets that wouldn't fit into memory if processed in a single batch.
- **Better Generalization**: The noise introduced by minibatch updates can assist in preventing overfitting and improve generalization by enabling the model to explore different paths during optimization.

Algorithm 4: Mini-Batch Gradient Descent Algorithm -(Lindholm et al. 2019)

1. In the network, initialize the parameters θ and set $t = 1$.
2. For $i = 1$ to $Epoch(E)$:
   (a) Shuffle the training data randomly $\{(x_i, y_i)\}_{i=1}^{k}$.
   (b) For $j = 1$ to $\frac{k}{k_{mb}}$:
      i. Compute the gradient of the loss function by utilizing the mini-batch $\{(x_i, y_i)\}_{i=(j-1)k_{mb}+1}^{jk_{mb}}$:

$$\hat{g}_t = \frac{1}{k_{mb}} \sum_{i=(j-1)k_{mb}+1}^{jk_{mb}} \nabla_\theta L(x_i, y_i, \theta)\Big|_{\theta=\theta_t}.$$

      ii. Perform a gradient update step $\theta_{t+1} = \theta_t - \lambda \hat{g}_t$.
      iii. Increment the iteration counter $t = t + 1$.

6.4.4 Learning Rate

The learning rate, λ, is an important hyperparameter in the optimization of neural networks. It determines the step size at which the network's parameters are updated during the training process, directly influencing the convergence speed and the stability of the model. A rate too high risks overshooting the optimal solution, while a rate too low may lead to excessively slow convergence. Balancing this trade-off ensures that the network effectively learns the underlying pricing dynamics. We will briefly discuss the learning rate, drawing upon researh by Lindholm et al. (2019), Nesterov (2013), and Wrigglesworth (2021).

To find our optimal learning rate λ, we follow the strategy:

- Reduce the learning rate if the error continues to worsen or oscillates significantly
- Increase the learning rate should the error remain fairly stable but increases gradually rises over time.

By using a constant learning rate λ the gradient will approach zero when the optimum is reached. This is demonstrated by step $\lambda \nabla_\theta J(\theta)\big|_{\theta=\theta_t}$ also approaching zero, thus resulting in convergence. However, it is noted that for stochastic gradient descent this is not the case. The gradient $\hat{g}_t$ may not approach zero as $J(\theta)$ reaches its minimum, since $\hat{g}_t$ is only an approximation of the true gradient $\nabla_\theta J(\theta)\big|_{\theta=\theta_t}$. As a result, our adjustments will be too large as we near the optimum thus preventing convergence of the SGD algorithm.

In practice, the learning rate is adjusted dynamically. We begin with a relatively high learning rate and gradually reduce it to a specified level. When selecting λ, it is essential to balance the rate of convergence, and the potential for overshooting the global minima. While the direction of descent is determined by $\nabla_\theta J(\theta)$, λ controls the magnitude of each step in that direction as

noted by Nesterov (2013). For stochastic gradient descent, a common approach is to choose a fixed λ. This approach is risky as with too low of a step size, the estimate θ_t will not change significantly between interations, and the learning progress rate will slacken. The convergence will be excessively slow with too low of a learning rate, as noted above. With too large of a learning rate, the estimate may exceed the optimum and will fail to converge due to too long of a step size. For an optimal rate of learning, the convergence is achieved in a fair number of iterations.

The following graphs are obtained from Lindholm et al.(2019) and illustrate the behavior of the optimization process for the function $J(\theta)$ using different learning rates. The red markings on the curves indicate the incremental updates of the parameter estimate θ which is being optimized.

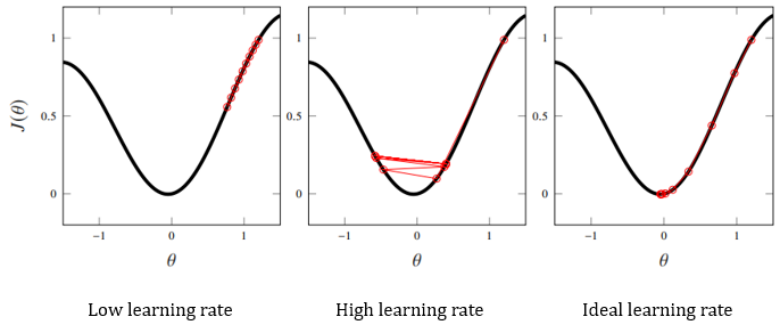


Figure 6.14: Optimization of $J(\theta)$ using Different Learning Rates

In the first graph the optimization process is very slow due to small step sizes. The red markings are closely spaced indicating minimal progress toward minimizing $J(\theta)$. While convergence is possible it would require a significant number of iterations, and is quite slow resulting in an inefficient training process. In contrast, the second graph demonstrates step sizes that are excessively large, causing the algorithm to overshoot the minimum of $J(\theta)$. This results in oscillatory behavior, with the red markings oscillating widely on both sides of the curve. The algorithm struggles to converge and may fail to achieve an optimal solution. The third graph depicts the ideal learning rate where the learning rate is well-calibrated. The optimization process is efficient and stable with the red markings showing steady progress toward the minimum of $J(\theta)$, without overshooting or unnecessary delays. This represents the optimal behavior striking a balance between convergence speed and accuracy.

The choice of the ideal learning rate is critical for effective optimization. The above graphs emphasise the importance of refining the learning rate for neural network training.

6.4.5 Optimization Algorithms

Optimization algorithms are vital for the effective training of neural networks. This enables them to model complex financial patterns accurately and efficiently. In option pricing the data relationships are non-linear and high-dimensional, and algorithms like Adam and RMSprop may improve convergence speed and stability. The correct algorithm may enhance the network's ability for option pricing. In this section we discuss various optimization algorithms at a high-level, discussing their structure and highlighting their shortcomings and strengths.

1. AdaGrad

Drawing upon the research by Ruder(2016) and Duchi, et al.(2011), and Vyacheslav(2023) we will discuss the AdaGrad algorithm.

AdaGrad is a gradient-based optimization algorithm that dynamically adjusts the learning rate for each parameter individually. By applying smaller updates to frequently occurring parameters and larger updates to less frequent ones, it is highly effective for sparse data scenarios. If the gradients associated with a specific weight vector component are large, the corresponding learning rate is reduced. Conversely, smaller gradients result in a larger learning rate. By this adaptive mechanism AdaGrad is able to effectively mitigate the issues of vanishing and exploding gradients. This method has improved the robustness of stochastic gradient descent (SGD) and has been widely employed in the training of large-scale neural networks, demonstrating its effectiveness in complex machine learning tasks.

Initially, all parameters θ used a single, shared learning rate λ for each θ_i. However, AdaGrad adjusts the learning rate for each parameter θ_i individually at every time step t. To illustrate AdaGrad's per-parameter update, we first define $g_{t,i}$ as the gradient of the objective function with respect to parameter θ_i at t:

$$g_{t,i} = \nabla_{\theta_i} J(\theta_{t,i}). \tag{6.44}$$

The SGD update for every θ_i at time t then becomes:

$$\theta_{t+1,i} = \theta_{t,i} - \lambda \cdot g_{t,i}. \tag{6.45}$$

Adagrad modifies the learning rate λ individually for each parameter θ_i at time step t, utilizing the sum of past gradients for θ_i:

$$\lambda_{t,i} = \theta_{t,i} - \frac{\lambda_t}{\sqrt{G_{t,ii} + \epsilon}}, \tag{6.46}$$

where $G_t \in \mathbb{R}^{d \times d}$ is a matrix, with the diagonal entry i, i representing the sum of the squares of the gradients with respect to θ_i up to time t. The term ϵ is a small positive term added for the prevension of division by zero. The square root in the denominator of the parameter update formula serves to stabilize the learning process by weakening the influence of accumulated squared gradients. It ensures that the learning rate adjustment does not become excessively large, which could lead to unstable updates. As G_t represents the cumulative sum of squared past gradients for all parameters θ, element-wise multiplication is performed between G_t and g_t:

$$\theta_{t+1} = \theta_t - \frac{\lambda_t}{\sqrt{G_t + \epsilon}} \odot g_t. \tag{6.47}$$

A key advantage of AdaGrad lies in its ability to adapt the learning rate dynamically during training, eliminating the need for manual tuning. The default of λ is 0.01 for most implementations, streamlining its use. However, a significant shortcoming is the squared gradients accumulation in the denominator, which causes the effective learning rate to decay progressively. Over time, this decay can render the learning rate exceedingly small, thus slowing convergence in later iterations and ultimately limiting the algorithm's capacity for further learning.

2. AdaDelta

Drawing upon the work by Ruder (2016) and Zeiler (2012) we will discuss the AdaDelta algorithm.

AdaDelta is an extension of AdaGrad that addresses the issue of its aggressively, monotonically decreasing learning rate. In contrast to AdaGrad, which relies on the cumulative sum of squared gradients, AdaDelta utilizes a running average of squared gradients. This approach prevents the learning rate from diminishing to negligibly small values thus enabling the algorithm to maintain its learning capacity effectively throughout the training process. The algorithm replaces the cumulative sum of squared gradients with an exponential moving average $E[g^2]_t$, providing greater weight to recent gradients while still considering historical values. This is expressed as:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2, \tag{6.48}$$

where:

- $E[g^2]_t$: The exponential moving average of squared gradients at time step t,
- γ: The decay rate, typically set to 0.9, similar to the momentum term
- g_t: The gradient at time step t.

The standard SGD update rule can be now be expressed in terms of the $\Delta\theta_t$ such that:

$$\Delta\theta_t = -\lambda_t \cdot g_{t,i}$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t, \tag{6.49}$$

where $\Delta\theta_t$ is the parameter update vector.
The AdaGrad update vector takes the form:

$$\Delta\theta_t = -\frac{\lambda_t}{\sqrt{G_t + \epsilon}} \circ g_t. \tag{6.50}$$

With the replacement of the matrix G_t with $E[g^2]_t$ we have the root mean squared error criterion for the gradient, which simplifies to:

$$\Delta\theta_t = -\frac{\lambda_t}{\text{RMS}[g]_t} g_t, \tag{6.51}$$

where $\text{RMS}[\Delta\theta]_t$ is initially unknown and is approximated using the RMS of previous parameter updates. With the substitution of λ with $\text{RMS}[\Delta\theta]_{t-1}$, the AdaDelta update rule becomes:

$$\Delta\theta_t = -\frac{\text{RMS}[\Delta\theta]_{t-1}}{\text{RMS}[g]_t} g_t, \tag{6.52}$$

where:

- $\text{RMS}[g]_t = \sqrt{E[g^2]_t + \epsilon}$, representing the RMS of the gradient at time step t,
- ϵ: A small constant to avoid division by zero.

AdaDelta scales the parameter updates by the ratio of the root mean squared (RMS) of past updates to the RMS of the gradients, thus ensuring that updates remain proportional and are not dominated by extreme values. While AdaDelta provides adaptive learning rates and eliminates the need for manual tuning, it does have certain limitations. A challenges lies in its sensitivity to hyperparameters, particularly the choice of the decay rate γ and the initialization of parameters. This can significantly influence its performance. Additionally, in some cases AdaDelta may converge more slowly than other optimizers like Adam, which utilize adaptive learning rates

alongside momentum to achieve faster convergence (Brownlee, 2021).

3. RMSProp

RMSProp is an optimization algorithm that was first proposed by Geoff Hinton (2012) in a Coursera Class lecture. We will discuss RMSProp briefly, drawing from work by Ruder (2016) and Bushaev (2018).

There are 2 ways to view RMSProp:

- We can view RMSProp as an adaptation of mini-batch learning.
- We can view RMSProp as a method to deal with AdaGrad's radically diminishing learning rates.

RMSProp and Adadelta were both developed around the same time independently in order to address the learning rate decay problem of Adagrad. Similar to AdaDelta, RMSProp introduces an exponentially weighted moving average of squared gradients in place of AdaGrad's cumulative sum, which allows it to emphasize recent gradient values while diminishing the influence of older gradients. This adjustment enables faster convergence. Like AdaDelta, RMSProp avoids the excessive decay of the learning rate over iterations thus ensuring it remains effective throughout training (Vyacheslav, 2023).

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2,$$

where Hinton (2012) suggests γ= 0.9, λ=0.001.

$$\theta_{t+1} = \theta_t - \frac{\lambda}{\sqrt{E[g^2]_t + \epsilon}} g_t. \tag{6.53}$$

While RMSProp may be effective for some scenarios, it has several limitations. It requires precise tuning of the hyperparameters like the learning rate and decay rate, and its lack of momentum makes it less effective at escaping local minima, in comparison to Adam. Additionally, its reliance on single-step gradient updates can lead to slower convergence in complex spaces. Unlike Adam, RMSProp does not dynamically adapt the global learning rate which limits its flexibility in some scenarios. Adam often surpasses RMSProp by integrating momentum with adaptive learning rates thus enabling faster convergence and improved robustness across a range of scenarios.

4. Adam

Our selected stochastic optimizer is Adam, or Adaptive Moment Estimation an advanced optimization algorithm specifically designed for training deep learning models. Adam is an "adaptive moment" estimator as it utilizes the first and second moments of gradients to compute dynamic learning rates for each parameter, ensuring efficient and precise optimization. It is important to note that Adam addresses several challenges commonly encountered with stochastic gradient descent, including instability and slow convergence. Combining the strengths of both AdaGrad and RMSProp, Adam has been recognized as one of the most effective optimization algorithms for neural networks by Oppermann (2022). Oppermann notes that Adam stands out as the ideal choice for most deep learning tasks due to its performance across a wide range of tasks. The diagram below, adapted from Soraemon (2022), illustrates the taxonomy of the SGD optimization algorithms. While momentum-based methods, such as Nesterov Accelerated Gradient and Nadam, build upon the momentum gradient approach, they are not included here for the sake of brevity.

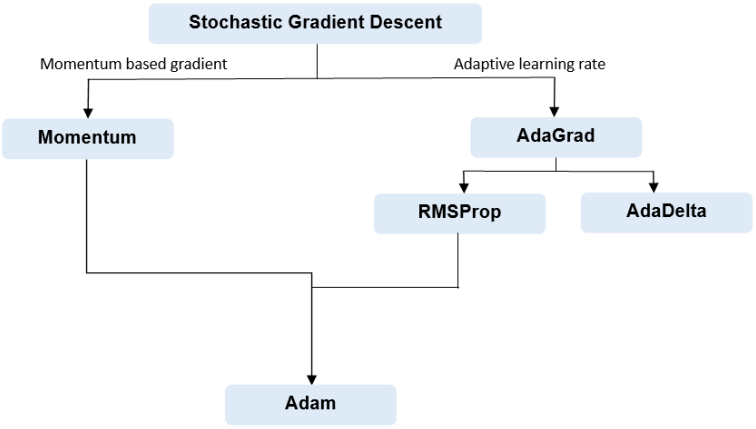


Figure 6.15: Stochastic Gradient Descent Optimization Algorithm Taxonomy

Define the parameters:

- θ_t: Parameter at time step t.
- g_t: Gradient of the loss function with respect to the parameters at time step t.
- m_t: First moment (mean) estimate at time step t.
- v_t: Second moment (variance) estimate at time step t.
- λ: Learning rate.
- ϵ: Constant to prevent division by zero.
- γ_1, γ_2: Exponential decay rates for the moment estimates, where $\gamma_1, \gamma_2 \in [0, 1]$.
- At $t = 0$, initialize $m_0 = 0$ and $v_0 = 0$.

Set $t = t + 1$. The objective is to minimize the expected value with respect to θ. To depict the derivation, we follow the steps for Adam optimisation below drawing from research by Pu (2021):

$$g_t = \nabla_\theta f_t(\theta_{t-1}) \tag{6.54}$$

We update the biased first moment estimate:

$$m_t = \gamma_1 m_{t-1} + (1 - \gamma_1) g_t \tag{6.55}$$

We update the second moment estimate:

$$v_t = \gamma_2 v_{t-1} + (1 - \gamma_2) g_t^2 \tag{6.56}$$

Using the bias-corrected first moment estimate:

$$\hat{m}_t = \frac{m_t}{1 - \gamma_1^t} \tag{6.57}$$

Using the bias-corrected second moment estimate:

$$\hat{v}_t = \frac{v_t}{1-\gamma_2^t} \tag{6.58}$$

such that:

$$\theta_{t+1} = \theta_t - \lambda \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \tag{6.59}$$

Mean(1^{st} moment): The moving average m_t depicts the mean of the gradients. The decay rate γ_1 controls how much of the past gradients information is kept (control exponential decay rate). **Variance(2^{nd} moment)**: v_t depicts the exponential moving averages of the variances of these gradients (squared gradients). γ_2 controls how much of the past squared gradients information is kept (control exponential decay rate).

Algorithm 5: Adam Optimization Algorithm - (Pu,2021)

---

1. For each iteration $t = 1, 2, \ldots$:
   (a) Compute the gradient of the objective function $f(\theta_t)$ with respect to the parameters:
   $$g_t = \nabla_\theta f(\theta_t)$$
   (b) Update the biased first moment estimate (mean of gradients):
   $$m_t = \gamma_1 m_{t-1} + (1-\gamma_1) g_t$$
   (c) Update the biased second moment estimate (uncentered variance of gradients):
   $$v_t = \gamma_2 v_{t-1} + (1-\gamma_2) g_t^2$$
   (d) Correct the bias in moment estimates:
   - Corrected first moment estimate:
   $$\hat{m}_t = \frac{m_t}{1-\gamma_1^t}$$
   - Corrected second moment estimate:
   $$\hat{v}_t = \frac{v_t}{1-\gamma_2^t}$$
   (e) Update the parameters:
   $$\theta_t = \theta_{t-1} - \lambda \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$
2. Stop when the parameters converge or after a fixed number of iterations.

We outline the key advantages and the default parameters used for Adam (Oppermann, 2022):

Key Advantages:

- Adaptive learning rates for each parameter based on the gradient's magnitude.
- Works well for sparse gradients and non-stationary objectives.
- Efficient and straightforward to implement.

Default Hyperparameters:

- $\lambda = 0.001 - 0.0001$ (learning rate)
- $\gamma_1 = 0.9$ (decay rate for first moment)
- $\gamma_2 = 0.999$ (decay rate for second moment)
- $\epsilon = 10^{-8}$ (to prevent division by zero)

This concludes the chapter on the structure and training of neural networks. To summarize, we began by providing a comprehensive background on neural networks, tracing their origins from the initial work of McCulloch and Pitts (1943) and Rosenblatt (1958), and contextualized their relevance within the financial domain. We then conducted an in-depth exploration of neural network architecture, detailing the mathematical and diagrammatic representations at each stage of their design. This included a discussion on commonly used activation functions, justifying our selection of ReLU and Leaky ReLU, and extending the analysis to deep neural networks (DNNs). We also highlighted the distinctions between artificial neural networks (ANNs) and DNN architectures, which are critical to the interpretation and presentation of our results.

Further, we presented the matrix notation and formal definition of a neural network and introduced the training process, focusing on the loss function to be minimized. This was linked to an extensive discussion of backpropagation and stochastic gradient descent (SGD), including the examination of both mini-batch SGD and standard SGD techniques. The chapter concluded with an overview of optimization algorithms, resulting in our selection of Adam as the optimizer for this dissertation.

This chapter forms the foundation of the dissertation, supporting our primary objectives: to demonstrate the potential of neural networks as a viable alternative to traditional methods in option pricing, and to evaluate the extent to which the choice of input parameters impacts neural network performance for option pricing. Special attention will be given to identifying "ideal" input parameters and analyzing the performance differences between ANNs and DNNs. In the next chapter, we will explore Monte Carlo methods as an additional benchmark for accuracy, a crucial step toward achieving our stated objectives.

Chapter 7

Monte Carlo Techniques

7.1 Monte Carlo Simulation Method

Monte Carlo simulations form a core method in financial analysis due to their flexibility and robust applications (Pu, 2021). They are utilized in the pricing of options and securities, as well as in improving the efficiency and reliability of pricing methods often serving as a benchmark for validating alternative approaches. John (2020) notes that while using Monte Carlo simulations may seem uneccesary when the Black-Scholes equation can be implemented, they provide a solution for cases where a closed form solution is not available. These simulations offer a probabilistic framework for modeling the uncertainty and randomness (McFarland & DeCarlo, 2020). This is inherent in financial markets and allows users of Monte Carlo methods to account for the complex stochastic nature of asset price movements.

While the application of Monte Carlo simulations for pricing American options is inherently complex due to their early exercise feature, the method offers numerous notable advantages (Longstaff & Schwartz, 2001). One significant benefit noted by Pu (2021) is that the computational time increases linearly with the number of factors influencing the option's value. Moreover, Monte Carlo simulations exhibit versatility and are capable of accommodating sophisticated stochastic processes such as jump diffusion. In practice, these simulations can harness parallel computing, thereby substantially improving computational efficiency.

A variety of Monte Carlo techniques have been developed for pricing American options. This section focuses on the Least-Squares Monte Carlo (LSM) method pioneered by Longstaff & Schwartz (2001). A detailed discussion of the Least-Squares Monte Carlo approach will follow in the next section.

In the context of neural networks for this dissertation, implementation of Monte Carlo simulations to option pricing will play a dual role in helping us meet our objectives. First, they provide a comparative benchmark to evaluate the accuracy and efficiency of neural network-based models (artificial and deep neural networks) for option pricing. Neural networks excel at approximating non-linear relationships and uncovering patterns in high-dimensional data as discussed thoroughly in the neural network chapter preceeding this. Monte Carlo simulations excel in generating precise, probabilistic estimates through repeated random sampling (Pu, 2021). This complementary nature of neural networks and Monte Carlo simulations allow us to assess the viability of neural networks as a computationally efficient alternative to option pricing using Monte Carlo simulations.

The second role of utilizing Monte Carlo simulations for option pricing in this dissertation is to present the argument that neural networks are effective approximators, even in volatile and dynamic market environments. Monte Carlo methods will enhance the training process of the neural networks by generating synthetic data to expand the dataset. This is done for the Heston stochastic volatility model, which forms the stochastic volatility component of the results section. By simulating a wide range of potential market scenarios, Monte Carlo techniques enable the neural network to learn a broader spectrum of pricing dynamics, therefore improving generalization and robustness in real-world applications.

This dissertation leverages Monte Carlo simulations, more specifically Least-Squared Monte Carlo simulations, not only as a benchmark for accuracy but also as a tool to highlight the strengths and limitations of neural networks in the financial domain. By comparing results from Monte Carlo methods with those generated by artificial and deep neural networks, we aim to validate the performance of neural networks and explore their potential to improve on traditional pricing methodologies, including in a dynamic market environment. This will also provide insight into whether the choice of neural network architecture, input parameters, and optimization algorithms significantly influences pricing accuracy and computational efficiency. This discussion reinforces the central objectives of this dissertation, paving the way for a comprehensive evaluation of neural networks in financial modeling and their practical implications in modern finance.

We are able to determine multi-dimensional integrals which are essentially the security price's expected discounted payouts over the sample paths by using the following steps outlined by Boyle, et al. (1996):

1. Generate sample paths of the underlying asset under the assumption of risk-neutral measures.
2. Calculate the discounted cashflows of the security along each sample path.
3. Compute the average of the discounted cashflows across all sample paths.

In order to understand the Monte Carlo method, we consider an integral:

$$\mu = \int g(x) f(x)\, dx, \tag{7.1}$$

where $f(x)$ is a probability density function. The integral is interpreted as the expected value and is depicted as:

$$\mathbb{E}[g(x)] = \int g(x) f(x) dx. \tag{7.2}$$

Drawing upon guidance from Hammersley & Handscomb (1965), we note that if an independent and identically distributed sequence of random numbers $\{x_i\}_{i=1}^{n}$ is generated from the probability density function, we have an approximation to the expected value depicted below as:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} g(x_i). \tag{7.3}$$

With $g(x)$ an integrable function and using Durrent (2010) we apply the Central Limit Theorem such that:

$$\hat{\mu} \to \mu \quad \text{as} \quad n \to \infty.$$

The variance (square integrable) can be depicted as

$$\sigma^2 = \text{var}[g(x)] = \int (g(x) - \mu)^2 f(x)\, dx. \tag{7.4}$$

Using the Central Limit Theorem,

$$\frac{\sqrt{n}(\hat{\mu} - \mu)}{\sigma} \to \mathcal{N}(0, 1).$$

The error distribution shown as $\hat{\mu} - \mu \sim \mathcal{N}(0, \frac{\sigma}{\sqrt{n}})$.

Drawing from work by Korn et al. (2010) we have the convergence rate for Monte Carlo methods, which is depicted as $O\left(\frac{1}{\sqrt{n}}\right)$. The rate of convergence is relatively slow compared to other numerical methods, but it does not increase with increasing dimensions (curse of dimensionality) as seen with numerical methods (Hammersley and Handscomb, 1965). Monte Carlo Methods maintain the same convergence rate regardless of the number of dimensions, and this is beneficial when dealing with high-dimensional problems. However, the slow convergence is still a limitation of the method and for increased accuracy there is still the requirement for a large number of samples, which is also computationally intensive. Boyle (1977) implemented Monte Carlo simulations in valuing derivatives. This approach essentially involved calculating the value of the option by estimating the discounted payoff's expected value under the risk-neutral measure. By generating a large set of payoff values and taking their average, an estimate of the expected value is achieved. This research is still applicable today, and provided a unique implementation of Monte Carlo simulations.

The graphs below illustrate the Monte Carlo simulations of stock price paths for a stock over a 1 year period with the following parameters: $S_0 = 100$, $r = 0.05$, $\sigma = 0.2$, $T = 1$, $N = 252$, $M = 1000$. This is depicted solely for visualization purposes

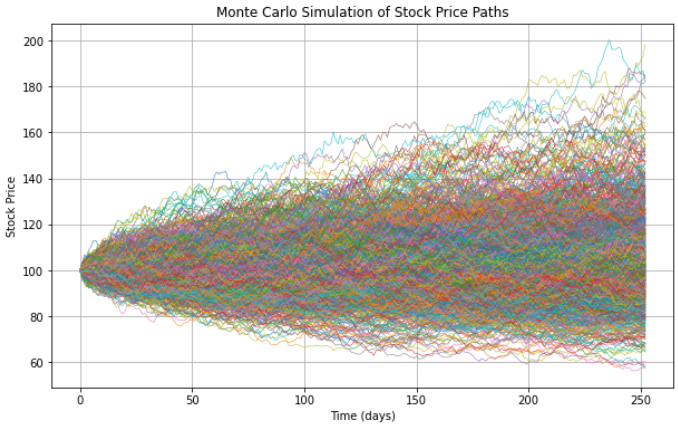


Figure 7.1: Monte Carlo simulations of stock price paths

7.2 Least-Squares Monte Carlo Simulation

In the paper "Valuing American Options by Simulation: A Simple Least-Squares Approach", Longstaff & Schwartz (2001) used simulation to determine the approximate value of American Options. The holder of an American Option can exercise the option at any time up to and including the date of expiration. After comparing the payoff from immediate exercise with the expected payoff from continuing to hold the option, a decision is made whether to exercise early, or to hold the option. Therefore to assess the optimal exercise price, Longstaff and Schwartz (LS) considered the conditional expectation of the payoff when the option-holder chooses to hold the option. The important concept undelying this approach is given by Longstaff & Schwartz (2001) as "this conditional expectation can only be estimated from the cross-sectional information in the simulation by using least squares."

In order to regress the actual payoffs from holding the option, a set of basis functions are used where the arguments of the basis functions are obtained from the underlying asset. The fitted values obtained from the regression will give a direct approximation of the conditional expectation function. With a clear approximation of the conditional expectation function for each exercise date, we can compare it to the immediate exercise values and derive an optimal exercise decision or "stopping rule" along each path. This process is repeated recursively and the cashflows discounted backward in time to zero. This will give us the price of the American option.

For the valuation, Longstaff and Schwartz (2001) had an underlying assumption of finite time horizon $[0, T]$, and the underlying probability space $(\Omega, \mathcal{F}, \mathbb{P})$, and equivalent martingale measure $\mathbb{Q}$. It was defined as:

"...where the state space Ω is the set of all possible realizations of the stochastic economy between time 0 and T and has typical element ω representing a sample path, $\mathcal{F}$ is the sigma field of distinguishable events at time T, and P is a probability measure defined on the elements of $\mathcal{F}$."

Consider the notation $C(\omega, s; t_k, T)$ indicating the path of cash flows generated by the option and is conditional on:
1. The option not being exercised before or at time t
2. The holder of the option following the optimal stopping strategy at every time (s) after t, $(t < s \leq T)$.

We are essentially using the Least-Squares Monte Carlo algorithm in order to maximize the American Option value by obtaining the pathwise approximation to the optimal stopping rule. The American Option can only be exercised at $0 < t_1 \leq t_2 \leq t_3 \leq ... \leq t_K = T$ for K discrete times, and taking into account the optimal stopping policy at each exercise date. For "continuously exercisable" American Options, the Least-Squares Monte Carlo method can be used to determine the value of the options by taking a sufficiently large K. At expiry, the option-holder chooses to either hold the option, or exercise the option based on whether it is in the money or out of the money. At t_k, the exercise date before the final expiration date, the option-holder has the choice to either exercise immediately or hold the option until the next exercise date where the holder has to make the same decision. The cash-flow at t_k is known to the option-holder, however the cash-flows from continuing the life of the option are not known. By applying the assumption of no-arbitrage, the value of the option when choosing not to exercise is equal to the risk-neutral expectation of the discounted remaining cash-flows, $C(\omega, s; t, T)$ This is discussed in detail in the research by Longstaff & Schwartz (2001).

The above is depicted as:

$$F(\omega, t_k) = \mathbb{E}_Q[\sum_{m=k+1}^{K} e^{-\int_{t_k}^{t_m} r(\omega,s)ds} C(\omega, t_m; t_k, T)|\mathcal{F}_{t_k}] \tag{7.5}$$

with a risk-free discounting rate of $r(\omega, t)$, and the expected value taken conditional on $\mathcal{F}_{t_k}$ being the information set at time t_k.

The conditional expectation function can be approximated by least-squares regression for each date $t_{k-1}, ..., t_1$. Because the option is defined recursively we note that $C(\omega, s; t_{k+1}, T)$ may differ from $C(\omega, s; t_{k+2}, T)$ because the optimal time to exercise may be different, and therefore the successive cashflows may change along realized path ω. The assumption made is that at time t_{K-1}, the function $F(\omega, t_{K-1})$ can be expressed as a linear combination of the orthonormal basis function $(f_m(X))$. These include: Laguerre, Hermite etc. and are discussed in the next section.

Therefore,

$$F(\omega; t_{K-1}) = \sum_{m=0}^{\infty} a_m f_m(X), a_m \in \mathbb{R} \tag{7.6}$$

with the approximation given:

$$F_P(\omega; t_{K-1}) = \sum_{m=0}^{P} a_m f_m(X), a_m \in \mathbb{R} \tag{7.7}$$

using the first $P < \infty$ basis functions.

The process is repeated backwards, considering only in-the-money paths, until the first exercise date. Once we have an exercise decision strategy, we can obtain the option cashflows with an approximation. We only consider the in-the-money paths for our approximation due to the fact that the exercise decision is only relevant when the option is in-the-money. As a result, the region is limited over which the conditional expectation is approximated, and fewer basis functions are required to get a correct approximation to the conditional expectation function value.

As $N \to \infty$, the approximation function $F_P(\omega; t_{K-1})$ converges in both mean square and in probability to the function $F_P(\omega; t_{K-1})$, which makes the approximation a very suitable unbiased estimator of the actual function. We will be using Laguerre Polynomials (order 3) as the basis function, as was demonstrated by Longstaff & Schwartz (2001).

Definition 6: Laguerre Polynomial The recursive formula depicting the analytical solution of the sequence:

$$La(n, x) = \frac{(2n-1-x)La(n-1,x)-(n-1)La(n-2,x)}{n},$$

where

$$La(0, x) = 1 \text{ and } La(1, x) = 1\text{-}x.$$

7.2.1 Robustness of the Least Squares Monte Carlo Algorithm

To test the robustness of the LSMC algorithm in American option valuation, we consider the method of adjusting the othonormal basis functions. We essentially want to test how effective the Least Squares Monte Carlo Algorithm is in valuing American options when adjusting a realistic number of paths and adjusting the orthonormal basis functions. This process was outlined by Moreno & Navas (2003). Their research focused on how different selections and modifications of these basis functions influence the accuracy and stability of the LSMC method, which is widely used for pricing American-style derivatives. By systematically testing various configurations of the basis functions, they demonstrated that the choice of basis significantly affects the algorithm's performance, particularly in capturing the complexities of early exercise features in American options. This work emphasizes the importance of basis function selection in ensuring the reliability of LSMC-based pricing models, offering valuable insights into optimizing this widely used simulation technique for practical financial applications.

As the results derived from implementing the Least-Squares Monte Carlo (LSMC) algorithm serve as the benchmark for accuracy when evaluating the performance of artificial and deep neural networks, examining the robustness of the LSMC algorithm through the adjustment of basis functions is highly pertinent. Nevertheless, the application and detailed exploration of this adjustment fall beyond the scope of this dissertation.

In the function space, continuous functions can be portrayed as a linear combination of basis functions. The same concept can be applied to the vector space, where every vector can be represented as a linear combination of basis vectors. A basis function can therefore be understood as an element of a specific basis (Kohn, et al., 2001).

We would utilize the following basis functions with the general formula given as:

$$a_{n+1}f_{n+1}(x) = (a_n + b_n x)f_n(x) - a_{n-1}f_{n-1}(x). \tag{7.8}$$

The table of orthonormal basis functions is depicted below, with the full equations given in the Appendix (Moreno & Navas, 2003).

Table 7.1: Basis functions Recurrence Law

	f_n	a_{n+1}	a_n	b_n	a_{n-1}	$f_0(x)$	$f_1(x)$
Chebyshev1 Kind A	$CH_1(x)$	1	0	2	1	1	x
Chebyshev2 Kind A	$CH_2(x)$	1	0	2	1	1	$2x$
Hermite	$H(x)$	1	0	2	2n	1	$2x$
Laguerre	$La(x)$	n+1	2n+1	-1	n	1	1-x
Legendre	$Le(x)$	n+1	0	2n+1	n	1	x
Powers	$P(x)$	1	0	1	0	1	x

With the above table, the basis function equations and using the research by Moreno and Navas (2003), the robustness of the Least-Squares Monte Carlo (LSMC) algorithm can be assessed.

7.3 Backwards Dynamic Programming Principle

The optimal stopping problem involves finding the best time to exercise an option with the intention to maximize expected profit. The solution follows a dynamic programming algorithm where the optimal stopping time $\tau*$, is defined as the first time that the option value equals its intrinsic value. The continutation value of an American option represents the value of continuing to hold rather than exercising the option. A discussion surrounding the optimal stopping time and continuation value is necessary to adequately understand American option pricing, and provides a foundation to build on this knowledge with machine learning, particularly neural network application.

In the following section we will provide a detailed analysis on the backward dynamic programming priciple.

7.4 Least Squares Monte Carlo Optimal Stopping

The Longstaff-Schwartz Least Squares Monte Carlo method calculates the optimal stopping times directly, without computing the option price at each step. According to the research conducted by Oussama (2018), the Longstaff-Schwartz approach is relatively more computationally time-efficient that the CRR method for high-dimensional problems. The process outlined below draws upon research by Korn et al. (2010), Tang (2015), Moutzouris (2021).

To understand the stopping rule and continuation value of an american option, we utilize the following definitions adapted from Korn et al. (2010):

Definition 7: American Contingent Claim - (Korn et al. 2010)
An *American contingent claim* consists of a progressively measurable stochastic process $X = \{(X_t, \mathcal{F}_t)\}_{t\in[0,T]}$, where $X_t \geq 0$ and represents the value of the claim at time t, with a final payment X_τ received at the exercise time $\tau \in [0, T]$, chosen by the holder of the contingent claim.
We assume that τ is a stopping time, and that $\{(X_t, \mathcal{F}_t)\}_{t\in[0,T]}$ possesses continuous paths. Additionally, we assume:

$$\mathbb{E}\left(\sup_{0\leq s\leq T}(X_s)^\alpha\right) < \infty \quad \text{for some } \alpha > 1. \tag{7.9}$$

Theorem 4: Fair value of an American Contingent Claim - (Korn et al. 2010)
The fair price $\hat{p}$ of an American contingent claim X is given by:

$$\hat{p} = \sup_{\tau \in \mathcal{S}[0,T]} \mathbb{E}^{\mathbb{Q}} \left(e^{-r\tau} X_\tau \right), \tag{7.10}$$

where:

- $\mathcal{T}[0,T]$ denotes the set of all stopping times within the interval $[0,T]$.
- $\mathbb{E}\left(e^{-r\tau}X_\tau\right)$ is the expected discounted payoff at stopping time τ, with X_τ representing the payoff of the claim at time τ.

There exists an optimal stopping time τ^* such that the supremum is attained at $\tau = \tau^*$.

The stock process $S(t)$ is modeled as a Markov process in $\mathbb{R}^d$, where P represents the payoff function. Consider an American put option, where $B(t) = P(S(t))$ denotes the value of the payoff if the holder decides to exercise the option at time t. Additionally, $g(S(t))$ denotes the discounted time-0 value of the payoff if the option holder exercises at time t, where $g(S(t)) = e^{-rt}B(t) = e^{-rt}P(S(t))$.

We denote the fair price of a vanilla American put option by utilizing an optimal stopping time τ^*, depicted by:

$$\tau^* = \inf\{t \geq 0 : S(t) \leq b^*(t)\}, \tag{7.11}$$

where $b^*(t)$ depicts the optimal exercise boundary.

This means that, when pricing American options, we must determine at each potential exercise opportunity $\{t_1, \ldots, t_m\}$ whether to exercise the option or not. We achieve this by utilizing our knowledge of the optimal exercise strategy τ^* in advance, leading us to the use of a Monte Carlo approach for pricing American options. The concept behind the backward dynamic programming principle is to start at the point of maturity where the exercise decision is known. We then proceed backwards by single steps until the initial starting value is reached, while updating the optimal exercise decision at each step.

The following process is outlined by drawing upon work from Tang (2015) and Moutzouris (2021):

Let $\tilde{V}(S(t_i))$ depict the value of an American option at time t_i, assuming that the option is exerciable at t_i, for $i \in \{1, 2, \ldots, m\}$. We compute the option value $\tilde{V}(S(t_0))$ using the following equations:

$$\tilde{V}(S(t_m)) = P(S(t_m)) \tag{7.12}$$

$$\tilde{V}(S(t_i)) = \max\left(P(S(t_i)), E[e^{-r\tau}\tilde{V}(S(t_{i+1}))|S(t_i)]\right) \tag{7.13}$$

for $i = m-1, m-2, \ldots, 1$ and for $\tau = t_{i+1} - t_i$.

From equation (7.12), we note that the payoff function P depicts the option's value at maturity. From equation (7.13), the option's value is determined as the maximum of its intrinsic value $P(S_{t_i})$ and the discounted expected continuation value, $\mathbb{E}[e^{-r\tau}\tilde{V}(S_{t_{i+1}})|S_{t_i}]$ at t_i.

The majority of methods for valuing an American option are based on the formulation presented

in equations (7.12) and equation (7.13) above. This is outlined by Korn et al. (2010) and is demonstrated in the binomial tree method, where the conditional expectation in equation (7.13) is calculated as the average of the two succeeding nodes. The conditional expectation of American options is a recognisable challenge which is reinforced by the difficulty in approximating the nested conditional expectations in the equation (7.13). To determine the options value at t_i for $i = 1, 2, \ldots, m$, we use the equations (7.12) and equation (7.13) above. The option value at every step is approximated with respect to the t_0 value. The discounted payoff function is given by: $g(\cdot) = e^{-rt_i}(\cdot)$.

We depict the t_0 dynamic programming principle $V(S_{t_i}) = e^{-rt_i}\tilde{V}(S_{t_i})_{i=1,2,\ldots,m}$ by:

$$V(S_{t_m}) = g(S_{t_m}), \tag{7.14}$$

and

$$V(S_{t_i}) = \max(g(S_{t_i}), \mathbb{E}[V(S_{t_{i+1}})|S_{t_i}]) \tag{7.15}$$

for $i = m-1, m-2, \ldots, 1$.

We proceed to prove the equality of the time t_i and time t_0 dynamic programming formulations.

Proof:

For $i = 1, 2, \ldots, m$

$$P(S_{t_i}) = e^{-rt_i}P(S_{t_i}) \tag{7.16}$$

$$V(S_{t_i}) = e^{-rt_i}\tilde{V}(S_{t_i}). \tag{7.17}$$

Therefore, for $i = 0$:

$$V(S_{t_0}) = \tilde{V}(S_{t_0}) = e^{-rt_1}\tilde{V}(S_{t_1}) = V(S_{t_1}). \tag{7.18}$$

For $i = m$:

$$V(S_{t_m}) = e^{-rt_m}V(S_{t_m}) = e^{-rt_m}P(S_{t_m}) = g(S_{t_m}). \tag{7.19}$$

For $i = m-1, m-2, \ldots, 1$, we have :

$$\begin{aligned}
V(S_{t_i}) &= e^{-rt_i}\tilde{V}(S_{t_i}) \\
&= e^{-rt_i}\max\left(P(S_{t_i}), \mathbb{E}\left[e^{-r(t_{i+1}-t_i)}\tilde{V}(S_{t_{i+1}})\middle|S_{t_i}\right]\right) \\
&= \max\left(e^{-rt_i}P(S_{t_i}), \mathbb{E}\left[e^{-rt_i}e^{-r(t_{i+1}-t_i)}\tilde{V}(S_{t_{i+1}})\middle|S_{t_i}\right]\right) \\
&= \max\left(g(S_{t_i}), \mathbb{E}\left[e^{-rt_{i+1}}\tilde{V}(S_{t_{i+1}})\middle|S_{t_i}\right]\right) \\
&= \max\left(g(S_{t_i}), \mathbb{E}\left[V(S_{t_{i+1}})\middle|S_{t_i}\right]\right).
\end{aligned} \tag{7.20}$$

Continuation Value

The choice of exercise vs retention occurs at each potential exercise date for an American option. The **continuation value** is the value of holding the option, which is defined using time-t_0 as:

$$C(S_{t_i}) = \mathbb{E}[V(S_{t_{i+1}})|S_{t_i}], \quad i = 1, 2, \ldots, m-1. \tag{7.21}$$

Therefore, the t_0 value-based BDPP (7.14)-(7.15) can be depicted as:

$$C(S_{t_m}) = 0, \tag{7.22}$$

and

$$C(S_{t_i}) = \mathbb{E}\left[\max(g(S_{t_{i+1}}), C(S_{t_{i+1}}))\middle|S_{t_i}\right] \tag{7.23}$$

for $i = m-1, m-2, \ldots, 0$ and where $C(S_{t_0})$ depicts the American Option fair price.

From equations (7.15) and (7.21), we observe that the discounted process $V(S_{t_i})$ determines the continuation value by using work from Glasserman (2003):

$$V(S_{t_i}) = \max(g(S_{t_i}), C(S_{t_i})) \tag{7.24}$$

for $i = 1, 2, \ldots, m$.

Stopping Rule

The dynamic programming recursions at time-t_i specified by (7.12)-(7.13), and at time-t_0 specified by (7.22)-(7.23) bring attention to option values. It can also be redefined in terms of the **stopping rules** and the **optimal exercise region** as follows:

$$\tau^*(m) = t_m, \tag{7.25}$$

and

$$\tau^*(i) = \begin{cases} t_i, & g(S_{t_i}) \geq \mathbb{E}[g(S_{\tau^*_{i+1}})|S_{t_i}] \\ \tau^*(i+1), & g(S_{t_i}) < \mathbb{E}[g(S_{\tau^*_{i+1}})|S_{t_i}] \end{cases} \tag{7.26}$$

for $i = m-1, m-2, \ldots, 1$.
The optimal regions are defined below:

Definition 8: Optimal Regions- (Tang, 2015)
The optimal region for **exercising** the option at each potential exercise time t_i is defined as the set:

$$\{S(t_i) : g(S(t_i)) \geq \mathbb{E}[g(S(\tau^*(i+1)))|S(t_i)]\} \tag{7.27}$$

while the optimal region for **holding** the option is defined as:

$$\{S(t_i) : g(S(t_i)) < \mathbb{E}[g(S(\tau^*(i+1)))|S(t_i)].\} \tag{7.28}$$

The stopping rule τ^* can be interpreted as the first time the stock price $S(t_i)$ enters the optimal exercise region.

7.5 Convergence

The American option value, computed using the Longstaff-Schwartz LS method, $V_k^N(S_{t_0})$ closely estimates the "true" value of the American option, $V(S_{t_0})$. The approximation, $V_k^N(S_{t_0})$ is influenced by the number of paths simulated (N) and the number and the type of the basis functions used (k). By drawing from Tang (2015), the difference between $V_k^N(S_{t_0})$ and $V(S_{t_0})$ arises from two primary sources of error:

1. The first source of error arises from the use of Monte Carlo simulation, as the option's value, being an expectation, is approximated using an arithmetic mean. By an increase in the number of simulated paths (N), the error can be decreased. Table 7.2 depicts this effect, and it is depicted graphically below in figure 7.2.

Table 7.2: Monte Carlo Error Analysis for American Put Option Valuation

Number of Simulations	**Option Value**	**Standard Error**
10	33.8419	15.0148
100	29.3642	4.5393
1000	29.0456	1.7255
10 000	26.3452	0.4945
100 000	26.1644	0.1579
1 000 000	26.3187	0.0498

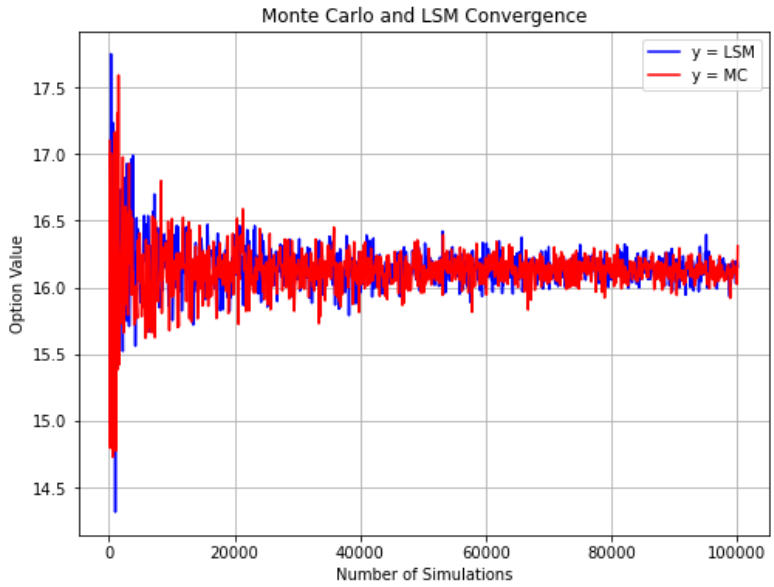


Figure 7.2: LSMC and MC Convergence

2. The second source of error arises from utilizing the set of finite basis functions $\{B_1, B_2, \ldots, B_k\}$ to approximate the continuation value of the option:

$$C(S_t) = \mathbb{E}[V(S_{t+1}) \mid S_t]. \tag{7.29}$$

With the selection of the appropriate number and type of basis functions (k), this error can be minimized.

Convergence Conditions:
Clément, et al. (2002) established the convergence properties of the Least-Squares Monte Carlo method as follows. They define the term $V_k(S_{t_0})$ by:

$$V_k(S_{t_0}) = \sup_{\tau \in \Gamma(B_1,\ldots,B_k)} \mathbb{E}\left(e^{-r\tau} g(S_\tau)\right), \tag{7.30}$$

where τ represents the optimal stopping time and $\Gamma(H_1, \ldots, H_k)$ is the set of exercise strategies derived from solving the least-squares regression problem with the basis functions $\{B_1, B_2, \ldots, B_k\}$.

The following is also noted by Clément, et al. (2002):

(a) With the number of basis functions kept constant, whilst increasing N (number of generated paths) results in the convergence of $V_k^N(S_{t_0})$ to $V_k(S_{t_0})$ which is the supremum of all option prices:

$$V_k^N(S_{t_0}) \xrightarrow{N\to\infty} V_k(S_{t_0}), \tag{7.31}$$

almost surely. This holds true as long as the basis functions are contained within the L^2-function space.

(b) As the number of basis functions $k \to \infty$, the option value $V_k(S_{t_0})$ shows convergance to the true option value $V(S_{t_0})$:

$$V_k(S_{t_0}) \xrightarrow{k\to\infty} V(S_{t_0}). \tag{7.32}$$

Remarks - (Tang, 2015)

1. With the number of basis functions kept constant, the value of the option calculated using the Longstaff-Schwartz algorithm can be depicted as $V_k^N(S_{t_0})$ only converges to $V_k(S_{t_0})$ as $N \to \infty$. It is important to note that it does not converge to the true option value $V(S_{t_0})$, which is an indication that the Longstaff-Schwartz algorithm provides an underestimation of the option price by utilizing a suboptimal exercise strategy.

2. Identifying the optimal number of basis functions for ensuring convergence can be difficult. Glasserman and Yu (2004) demonstrated that as k increases, achieving convergence may require exponential growth in N.

7.6 Simple Numerical Example by Longstaff & Schwartz : American Put Option

In the paper by Longstaff & Schwartz (2001), a simple numerical example is presented in order to understand the fundamentals of American option pricing. This dissertation follows the same underlying principles and understanding, however for a different set of data than that presented by Longstaff and Schwartz.

We analyze an American put option on a non-dividend-paying underlying asset. The option can be exercised at time $t = 1, 2, 3$ where $t = 3$ indicates the expiry date and has a strike price of 1.1. We assume a riskless continuously compounded interest rate of 7%. We simulate 8 different sample paths of the underlying under the risk neutral measure.

Table 7.3: Stock Prices for Paths

Path	t=0	t=1	t=2	t=3
1	1.00	1.079	1.056	1.427
2	1.00	1.175	1.431	1.887
3	1.00	1.389	1.094	1.011
4	1.00	0.871	0.865	0.792
5	1.00	1.725	1.945	1.675
6	1.00	0.987	0.626	0.790
7	1.00	0.654	0.763	1.004
8	1.00	0.989	1.532	1.784

Our goal is to identify the optimal stopping times that will maximize the American option value at time t_i for $i = 1, 2, 3, ...$ along every simulated path, as described by Longstaff & Schwartz (2001). The option pay-off at t_3 is depicted by the following matrix:

Table 7.4: Cash Flow at time 3

Path	t=0	t=1	t=2	t=3
1	-	-	-	0.00
2	-	-	-	0.00
3	-	-	-	0.089
4	-	-	-	0.308
5	-	-	-	0.00
6	-	-	-	0.31
7	-	-	-	0.096
8	-	-	-	0.00

The cash flows received at time $t = 3$ in this matrix are identical to those that would be obtained if the option were European. By utilizing this approach, we only consider the in-the-money paths and will maximize the option value along all the exercise dates for every path. If the put option is in-the-money at time 2, the holder of the option has the choice to either exercise immediately, or to continue holding the option until final expiration. At time t=2, there are five

in-the-money option paths. We choose X to be the discounted cash flows received at a future date should the option not be exercised, and Y indicates the price of the underlying. We regress Y on a constant X and X^2. We note this below for X and Y in the table showing regression at time 2.

Table 7.5: Regression at time 2

Path	Y	X
1	$0.00\ x\ e^{-0.07}$	1.056
2	-	-
3	$0.089\ x\ e^{-0.07}$	1.094
4	$0.308\ x\ e^{-0.07}$	0.865
5	-	-
6	$0.31\ x\ e^{-0.07}$	0.626
7	$0.096\ x\ e^{-0.07}$	0.763
8	-	-

A comparison is made between the intrinsic value and the estimated continuation value, where the intrinsic value is 1.1- $Path_{t_i}$ for $i = 1, 2, 3, ...$ By the regression of the discounted future cashflows, Y, on X and X^2 such that the conditional expectation can be determined by $\mathbb{E}[Y|X] = 0.31975 + 0.10072X - 0.32066X^2$.

Table 7.6: Optimal Exercise decision at time 2 for Early Exercise

Path	Exercise	Continuation Value	Decision
1	0.044	0.0685	Hold
2	-	-	-
3	0.006	0.04616	Hold
4	0.235	0.1669	Exercise
5	-	-	-
6	0.474	0.257	Exercise
7	0.337	0.20992	Exercise
8	-	-	-

As seen in the table depicting optimal exercise decision at time 2 for early exercise, it is optimal to exercise the option at time 2 for the fourth, the sixth and the seventh paths. In the table depicting the cash flow at time 2, we see the cashflows received by the holder of the option which is conditional on not exercising the option before time 2.

Table 7.7: Cash Flow at time 2

Path	t=1	t=2	t=3
1	-	0.00	0.00
2	-	0.00	0.00
3	-	0.00	0.089
4	-	0.235	0.00
5	-	0.00	0.00
6	-	0.474	0.00
7	-	0.337	0.00
8	-	0.00	0.00

If an option is exercised at time 2, the cashflow in final column time t=3 will become zero since an option can only be exercised once, and when exercised there will be no additional cashflows. We now consider where the option should be exercised for time 1 (Longstaff & Schwartz, 2001).

When considering the initial stock price matrix we see that there are five in-the-money paths at time 1. These paths, as seen previously will have Y as the discounted value of the successive cashflows for the option. The cashflows received at time 2 or time 3 are used for each path. It should be noted that if cashflows are received at time 2 they will have to be discounted back by one time period, and no further cashflows can occur since the cashflows can occur at either time 2 or 3, but not for both periods. The cashflow matrix at time 3 for the path becomes 0 (Longstaff & Schwartz, 2001).

Table 7.8: Regression at time 1

Path	Y	X
1	$0.00\ x\ e^{-0.07}$	1.079
2	-	-
3	-	-
4	$0.235\ x\ e^{-0.07}$	0.871
5	-	-
6	$0.474\ x\ e^{-0.07}$	0.987
7	$0.337\ x\ e^{-0.07}$	0.654
8	$0.00\ x\ e^{-0.07}$	0.989

By regressing Y on a constant X and X^2 at t=1 the conditional expectation can be determined. The estimated conditional expectation function is given by:
$\mathbb{E}[Y|X] = -1.1076 + 3.8413X - 2.5750X^2$ We substitute X into this regression equation in order to determine the conditional expectation function. We can now assess the table below depicting the optimal exercise decision at time 1.

Table 7.9: Optimal Exercise decision at time 1 for Early Exercise

Path	Exercise	Continuation Value	Decision
1	0.021	0.0392	Hold
2	-	-	-
3	-	-	-
4	0.229	0.28467	Hold
5	-	-	-
6	0.113	0.1753	Hold
7	0.446	0.30324	Exercise
8	0.111	0.17278	Hold

Therefore from the matrix we see that it is optimal to exercise at time 1 for path seven. We have now effectively identified the exercise strategy at times 1, 2, 3 and we will construct a stopping rule table below where a "1" indicates the exercise dates at which the option is exercised for the paths and for t=1,t=2,t=3 (Longstaff & Schwartz, 2001).

Table 7.10: Stopping Rule

Path	t=1	t=2	t=3
1	0	0	0
2	0	0	0
3	0	0	1
4	0	1	0
5	0	0	0
6	0	1	0
7	1	0	0
8	0	0	0

It is now possible to determine the cashflows realized by using the table 7.10 which indicates stopping rules. Exercise the option with a one in the matrix, which leads to the cashflow matrix depicting the option cash flow matrix.

Table 7.11: Option Cash Flow Matrix

Path	t=1	t=2	t=3
1	0.00	0.00	0.00
2	0.00	0.00	0.00
3	0.00	0.00	0.089
4	0.00	0.235	0.00
5	0.00	0.00	0.00
6	0.00	0.474	0.00
7	0.446	0.00	0.00
8	0.00	0.00	0.00

At time 1, the option is exercised at the seventh path, at time 2 the option is exercised at the fourth and sixth path, at time 3 the cashflow is received at the third path. There are no

cashflows for paths when the option is out-the-money. Using the process, we see that the value for the American put option is 0.13805. This value is significantly more than for the European put option we obtained from discounting the first cashflow matrix for time 3.

With this simple numerical example we have a better understanding of how the conditional expectation function can be determined by using the cross-sectional information in the simulated paths. We can then identify the exercise decision that optimizes the value of the American option at each time step and for every path.

Drawing upon the work by Tang (2015) and Moutzouris (2021), we have the following formalized algorithm:

Algorithm 6: Longstaff-Schwartz Least-Squares Monte Carlo Algorithm - (Tang, 2015)

---

1. Generate N independent simulated paths of the underlying at each exercise date:

$$\{S_{t_1}^{(n)}, S_{t_2}^{(n)}, \ldots, S_{t_m}^{(n)}\} \quad \text{where } n = 1, 2, \ldots, N \text{ and } t_i = \frac{T}{m} \times i, \quad i = 1, 2, \ldots, m.$$

2. The derivative's discounted terminal value at its maturity, for each path $n = 1, 2, \ldots, N$ is expressed as:

$$V(S_{t_m}^{(n)}) = g(S_{t_m}^{(n)}).$$

3. Compute iteratively by moving backward at each exercise time t_i for $i = m-1, m-2, \ldots, 1$:

   (a) Select basis functions: $\{B_1, B_2, \ldots, B_k\}$.

   (b) Determine the subset of in-the-money paths, $\omega \subset \{1, 2, \ldots, N\}$ such that $g(S_{t_i}^{(n)}) > 0$ $\forall n \in \omega_N$.

   (c) Perform the least-squares regression analysis:

$$\min_{\alpha_j \in \mathbb{R}} \frac{1}{N} \sum_{n=1}^{N} \left(V(S_{t_i}^{(n)}) - \sum_{j=1}^{k} \alpha_j B_j(S_{t_i}^{(n)}) \right)^2$$

   and the optimal coefficient vector α^*:

$$\alpha^* := [\alpha_1^*, \alpha_2^*, \ldots, \alpha_k^*]^T = (X^T X)^{-1} X^T Y \in \mathbb{R}^{k \times 1},$$

   where

$$Y := \left[V(S_{t_i}^{(1)}), V(S_{t_i}^{(2)}), \ldots, V(S_{t_i}^{(N)}) \right]^T \in \mathbb{R}^{N \times 1}$$

   and

$$X := \begin{bmatrix} B_1(S_{t_i}^{(1)}) & \ldots & B_k(S_{t_i}^{(1)}) \\ \vdots & \ddots & \vdots \\ B_1(S_{t_i}^{(N)}) & \ldots & B_k(S_{t_i}^{(N)}) \end{bmatrix} \in \mathbb{R}^{N \times k}.$$

(d) Evaluate the continuation value $C^*(S_{t_i}^{(n)})$ and the discounted exercise value $g(S_{t_i}^{(n)})$ for every path $n \in \omega_N$:

$$C^*(S_{t_i}^{(n)}) = \sum_{j=1}^{k} \alpha_j^* B_j(S_{t_i}^{(n)}).$$

(e) Make a comparison between the estimated value $C^*(S_{t_i}^{(n)})$ and the discounted exercise value $g(S_{t_i}^{(n)})$ to make a decision on exercising:

$$V(S_{t_i}^{(n)}) = \begin{cases} g(S_{t_i}^{(n)}), & \text{if } n \in \omega_N \text{ and } g(S_{t_i}^{(n)}) \geq C^*(S_{t_i}^{(n)}) \\ V(S_{t_{i+1}}^{(n)}), & \text{otherwise.} \end{cases}$$

4. Determine the American option price:

$$V_N^h(S_{t_0}) = \frac{1}{N} \sum_{n=1}^{N} V(S_{t_1}^{(n)}).$$

7.7 Heston Stochastic Volatility Modelling Process

This section will cover the detailed explanation of implementing the Heston stochastic volatility model with the Least-Squares Monte Carlo method, with application to artificial and deep neural networks. As this process serves to assist us in meeting one of our core objectives: using the Heston stochastic volatility model, the dissertation demonstrates robustness of neural networks in handling of volatile market conditions. As this process is not as straightforward as the constant volatility process, it is outlined in more detail in the following section.

The process we descibe below is the integration of the **Heston stochastic volatility model** with the Least Squares Monte Carlo (LSMC) method to generate synthetic data and evaluate the performance of neural networks (ANNs and DNNs) in option pricing. The **Heston model** serves as the foundation for simulating realistic financial market conditions. This is done by modeling both asset price and variance as stochastic processes. This enables the generation of dynamic data that is essential for robust neural network training.

The Heston model provides simulated paths for asset prices and variances using parameters: the initial stock price (S_0), strike price (K), risk-free rate (r), and stochastic volatility parameters (κ, θ, σ, ρ, v_0). These paths are derived from the model's stochastic differential equations (SDEs), which capture both mean-reversion behavior and correlation between asset price and variance. This simulation step ensures that the neural networks are exposed to complex and realistic market dynamics.

The **LSMC method** complements the Heston model by acting as a pricing mechanism. Once the Heston model has generated paths for asset prices as described above, the LSMC model uses these paths to calculate option prices. Specifically, LSMC applies regression techniques to approximate continuation values, facilitating the identification of optimal early exercise strategies for American-style options. The regression uses basis functions, specifically the Laguerre polynomial in this dissertation, to estimate the continuation value at each timestep. This iterative process produces a benchmark option price for comparison with neural network predictions.

The generated datasets, which include simulated paths and LSMC-calculated option prices, are preprocessed using scalers such as `MinMax Scaler` or `Standard Scaler`, which will be discussed in the methodology section below. This normalization ensures that input features are scaled appropriately, allowing the neural networks to learn efficiently. In the case of the Heston stochastic volatility model, we will only be considering the ideal paramters, i.e. the MinMax Scaler, LeakyRelu, and minibatch gradient descent. Two distinct neural network architectures are trained on this data:

- **Artificial Neural Network (ANN)**:
  - A simpler structure comprising 32 neurons per layer across three layers.
- **Deep Neural Network (DNN)**:
  - A more complex architecture with 64 neurons per layer across four layers.
  - Incorporates dropout layers to mitigate overfitting by randomly deactivating neurons during training, encouraging model generalization. This will be discussed in more detail in the methodology section below

Both networks utilize the Adam optimizer and mean squared error (MSE) loss function to minimize the difference between predicted and benchmark prices. Predictions from these networks are then evaluated across varying conditions, such as changes in underlying stock price (S_0), volatility (σ), and time to maturity (T). Sensitivity analyses are conducted to visualize the influence of these parameters on neural network performance.

7.7.1 Relationship Between Heston Model and LSMC

The Heston model provides the stochastic paths that serve as input for the LSMC algorithm. LSMC, in turn, uses these paths to compute benchmark option prices, allowing for the evaluation of neural network predictions. This complementary relationship ensures that the Heston model will generate realistic scenarios, while LSMC provides an accurate pricing benchmark.

This integration of the Heston model and LSMC aligns closely with the dissertation's objectives:

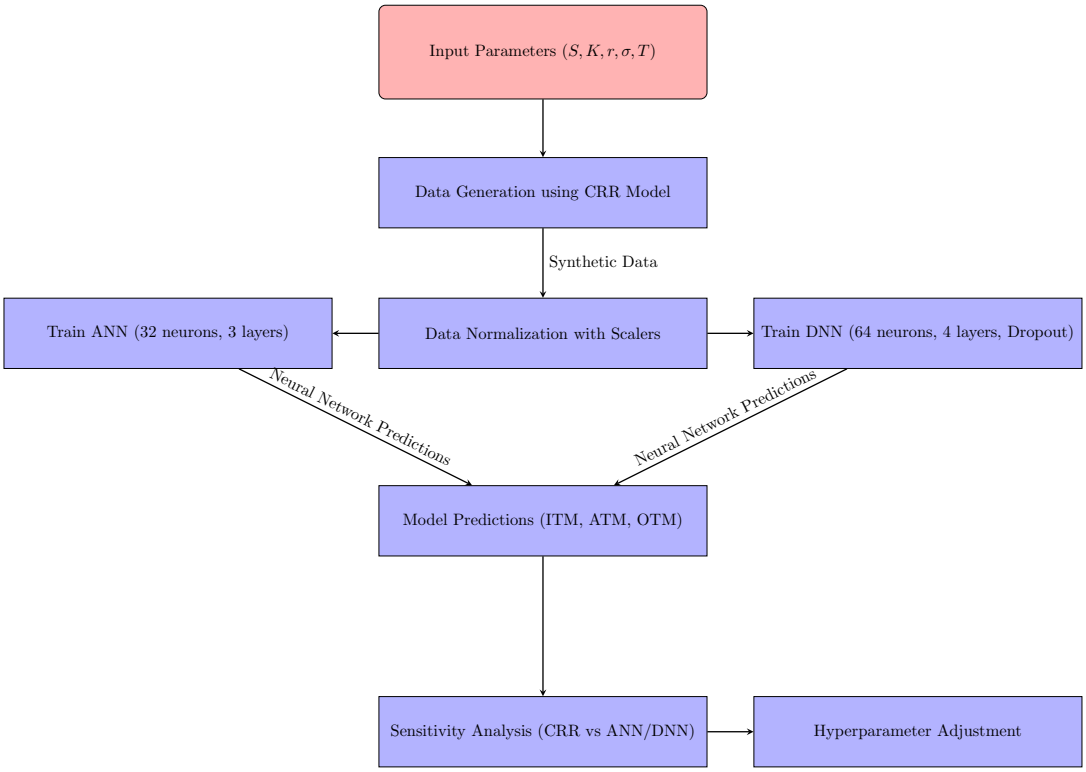
- **Exploring Volatile Market Conditions**: By simulating paths under the Heston model, neural networks are tested in dynamic environments, demonstrating their robustness in handling stochastic volatility.
- **Comparative Analysis**: LSMC serves as one of our traditional benchmarks for comparing neural network performance, enabling the evaluation of ANNs and DNNs as viable alternatives.
- **Optimal Parameter Identification**: Sensitivity analyses and the use of scalers and dropout layers highlight the impact of input parameters on model performance, contributing to the identification of an "ideal parameters" scenario. For the stochastic volatility component of the disseration, we already have the ideal parameteres, and will be using those to test the performance of the neural networks.

By combining the strengths of the Heston model, LSMC, and neural networks, we intend to thoroughly evaluate the applicability of neural networks in option pricing and their potential to outperform traditional methods in accuracy and computational efficiency.

Chapter 8

Methodology

8.1 Diagramatic Representation



8.2 Diagram Workflow Explanation

The diagrammatic representation above illustrates the end-to-end process of utilizing neural networks to price options, specifically comparing the CRR binomial tree method with artificial neural networks (ANN) and deep neural networks (DNN). This diagram is specific to the 'ideal parameters' scenario, but can be adjusted and applied to all testing scenarios in the results section

Initially, synthetic training data is generated using the CRR model. This data relies on input parameters such as the initial stock price (S), strike price (K), risk-free interest rate (r), volatility (σ), and time to maturity (T). The generated data is then normalized using scaling techniques like MinMaxScaler or StandardScaler to standardize input features, ensuring efficient model training.

The next phase involves training two neural network architectures: an ANN with 32 neurons across three layers and a DNN with 64 neurons across four layers, incorporating two 30% dropout layers for regularization. Both models use the Mean Squared Error (MSE) loss function and the Adam optimizer for parameter updates, with 20% of the data reserved for validation during training. This step ensures model accuracy and allows generalization.

Once trained, the ANN and DNN models predict option prices for various scenarios, including in-the-money (ITM), at-the-money (ATM), and out-of-the-money (OTM) situations. The CRR method serves as a benchmark to evaluate the performance of these neural networks.

Finally, sensitivity analyses are conducted to examine how variations in key parameters—such as share price, volatility, and time to maturity—affect model predictions. The results inform hyperparameter adjustments for improved performance in subsequent iterations.

8.3 Python Overview

As discussed, the methods we consider include:

- The Cox-Ross-Rubinstein (CRR) method for both American and European options.
- The Black-Scholes (BS) method.
- The Longstaff-Schwartz Least Squares Monte Carlo Simulation (LSM) method.

Packages Used

The following Python packages are utilized throughout the analysis for both European and American option pricing, as well as for neural network modeling:

- **numpy**: For efficient numerical computations, including array manipulations and matrix operations.
- **tensorflow**: A platform for machine-learning applications, used for defining and training neural networks.
- **keras.models**: The `Sequential` class from Keras, utilized to create and train neural network models.
- **from keras.layers import Dense**: The `Dense` layer class from Keras, used to construct layers in neural networks.
- **sklearn.preprocessing.MinMaxScaler**: Used for feature scaling of the training dataset.
- **sklearn.preprocessing.StandardScaler**: Alternative feature scaler for normalizing the dataset.
- **scipy.special.eval_laguerre**: Used to compute Laguerre polynomials, which serve as basis functions in the Longstaff-Schwartz method.
- **scipy.integrate.quad**: Used for numerical integration where applicable.
- **scipy.stats.norm**: Provides methods related to the normal distribution, such as cumulative distribution functions.
- **matplotlib.pyplot as plt**: For generating graphs, plots, and visualizations of results.
- **time**: Used for measuring computational time for different methods.

Standard Parameters

The following parameters are standard across option pricing methods, such as Black-Scholes and the Cox-Ross-Rubinstein models:

- S_0: Initial stock price.
- K: Strike price.
- r: Risk-free interest rate.
- T: Time to maturity.
- σ: Volatility of the stock.

Heston-Specific Parameters

The Heston stochastic volatility model introduces additional parameters for capturing the dynamics of volatility:

- κ: Speed of mean reversion for the volatility process.
- θ: Long-term mean (or equilibrium level) of variance.
- ρ: Correlation between the Brownian motions driving the asset price and variance processes.
- v_0: Initial variance of the underlying asset.

Simulation Parameters

The following simulation parameters are use:

- M: Number of time steps in the simulation.
- I: Number of paths simulated in the Monte Carlo process.
- `num_samples`: Number of samples for generating training data.

Functions and Workflow

1. **Generate Price Paths:** Functions are created to simulate price paths for the underlying asset, e.g., geometric Brownian motion.
2. **Calculate Payoffs:** Functions calculate the payoff of the option at each time step, particularly for methods like LSM.
3. **Define Pricing Methods:** Functions are defined for pricing both American and European options.
4. **Training Data Generation:**
   - Training data is generated for neural networks.
   - Using the LSM method, the function computes training data by pricing options.
5. **Neural Network Training:** The neural networks are defined and trained using the computed training data.

8.4 Scaling

We use the functions MinMaxScaler and StandardScaler from sklearn.preprocessing. A scaler is used as a preprocessing tool to normalize or standardize the input features before training neural networks. By scaling the data to a uniform range or distribution, the scaler ensures that all input parameters, such as stock price, strike price, volatility, and time to maturity, are treated with equal importance during the training process. This step is crucial in application to neural network as features with varying scales can lead to uneven gradient updates, slowing down convergence or causing instability in the optimization process. To recap our thesis objectives, we aim to evaluate the performance of neural networks in option pricing and determine the impact of input parameters on accuracy. Thus, the use of a scaler plays a pivotal role in meeting our

objectives. It not only improves the training efficiency but also allows for a fair comparison of our results across different parameter combinations, ensuring the robustness and reliability of our findings. We will draw from the research by Brownlee (2020) and Pelletier (2024).

As stated by Christopher Bishop in his book 'Neural Networks for Pattern Recognition' (Bishop, 1996):

"In practice it is nearly always advantageous to apply pre-processing transformations to the input data before it is presented to a network. Similarly, the outputs of the network are often post-processed to give the required output values"

- Standardization (Z-score normalization): Standardization transforms the data to achieve a mean of zero and a standard deviation of one. This is useful when the features have different units or vastly different ranges.
- Min-Max Scaling: This technique normalizes the data to a specified range, usually [0, 1]. This is useful when you need the input values to be within a specific range.

The table below depicts the equations for the Standard and the MinMax Scaler, adapted from Reddy (2018).

Standard Scaler	$\frac{x_i - \bar{x}}{\sigma_x}$
MinMax Scaler	$\frac{x_i - \min(x)}{\max(x) - \min(x)}$

We have chosen to compare results from the use of both a Standard Scaler and a Min-Max Scaler.

Benefits of MinMax Scaler

- Keeps Features within a Fixed Range: By scaling features to a specific range, most likely [0,1], it ensures that all features contribute equally towards the model.
- Preserves the Shape of the Distribution: Unlike with standardization, using MinMax Scaling doesn't distort the original distribution of data values. It is an ideal choice when the data has no strong outliers or is already in a fairly uniform range.
- Useful for neural networks: MinMax Scaler is commonly utilized in deep learning with the intention that neural networks perform best when inputs are scaled to a range.
- Given that option pricing models involve input features such as stock prices, strike prices, and volatilities, which is uniformly simulated in our case, we anticipate that the MinMax Scaler could be more appropriate.

Benefits of Standard Scaler

- Centers Data Around Zero: Standardization centers the data around the mean, making it a suitable choice for algorithms based on the assumption of a Gaussian distribution.

- Handles Data with Outliers: By scaling based on the standard deviation, the Standard Scaler is less sensitive to extreme values than the MinMax Scaler. This makes it a more robust choice when the data has outliers.
- Improves Convergence for Some Algorithms: Many ML models converge faster and produce better results when features are standardized. This will improve overall performance.

8.5 Dropout

Dropout is an important regularization technique in deep learning that randomly "drops out" or deactivates a fraction of neurons during training, preventing over-reliance on specific features and reducing the risk of overfitting (Goodfellow et al. 2016). By the term 'regularization' we refer to the ML concept aimed at mitigating overfitting and improving a model's ability to generalize. Overfitting occurs when the model learns details and patters of limited training data very closely, and rather than capturing the general trend it also captures sampling noise. Consequently, the model performs very well on the training data, however, it performs poorly on unseen data. This renders the model (or DNN) ineffective for real-world application. Regularization techniques incorporate constraints or adjustments during training to find the balance between underfitting and overfitting, thereby enhancing the model's performance on unseen data. This balance trade-off is explained in the graph below obtained from Tewari (2021):

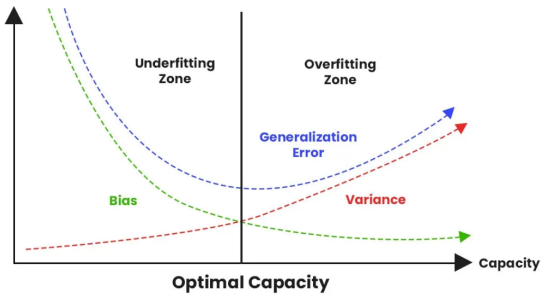


Figure 8.1: Trade-off graph for bias vs variance

This graph illustrates the bias-variance tradeoff, highlighting how model complexity affects error. Simpler models in the underfitting zone have high bias as they fail to capture data patterns, while complex models in the overfitting zone have high variance due to overfitting. The optimal capacity lies where the generalization error, (the sum of bias and variance) is minimized. Achieving this balance ensures that the model generalizes well to unseen data. Techniques like regularization and early stopping assist in maintaining this optimal balance.

Dropout layers play a critical role in enhancing the performance and generalization capabilities of neural networks by mitigating overfitting during training. In this dissertation, dropout layers are strategically incorporated into the neural network architectures in order to achieve one of the key objectives: evaluating the robustness and accuracy of neural networks in option pricing by adjusting input parameters. Our use of dropout layers aligns with this dissertation's goal of determining how neural network structures and parameter configurations, including regularization techniques, influence the model's predictive performance and computational efficiency. From our analysis of our results, we will then create an 'ideal parameters' scenario. We will draw upon

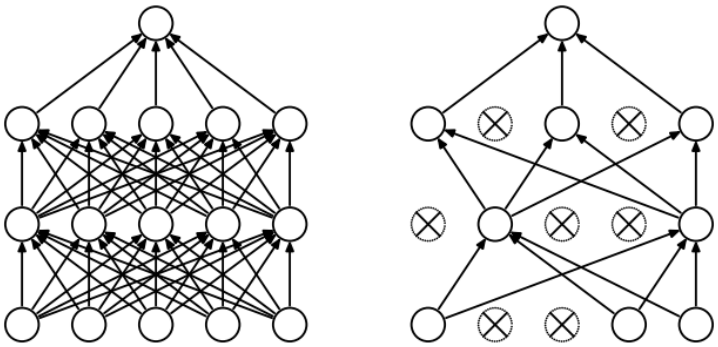
the work by Lindholm et al. (2019), Srivastava et al. (2014), and Wrigglesworth (2021).

Deep neural networks are often favoured for their accuracy and ability to learn complicated relationships between data inputs and outputs. However, a significant shortcoming of the process is overfitting. To mitigate this, a dropout function is implemented.

A dropout function is a regularization technique that is designed to mitigate overfitting by preventing units from co-adapting excessively. A dropout function essentially "drops out" a subset of neurons and their connections randomly within a layer during each forward pass of the training process. During each training iteration, individual neurons are 'disabled' with a specified probability p, usually between 0.2 to 0.5. With this random selection when deactivating the neurons, the odds of over-relying on particular neuron outputs are reduced, and a more distributed representation of the data is encouraged.

Applying dropout in a neural network essentially involves creating a thinned version of the network. This thinned network after dropout was applied consists of all units that remain active, essentially a sub-network of the original network. The collections of dropped units are independent between sub-networks.

The diagram below obtained from Srivastava et al. (2014) depicts the dropout process. The dropped units are crossed-out on the graph to the right.



(a) Standard Neural Net (b) After applying dropout.

Figure 8.2: Dropout

The following steps are repeated during the training process Aggarwal (2018):

- Sampling the Network: Generate a neural network by sampling from the base network. All nodes are sampled independently from each other. If a node is dropped, all its connections are also removed to ensure that it doesn't contribute to the forward or backward pass in that training iteration.
- Sampling Training Data: Select a single training example or a mini-batch of training data. This is used to update the weights of the retained nodes. This mini-batch approach helps stabilize training.
- Updating Weights: After randomly sampling which nodes remain active in this particular

training iteration, backpropagation is applied to update the weights of the active nodes based on the error from the training data. This process repeats, with different nodes being dropped in each iteration, leading to a network that learns more robust and generalized patterns.

The process is depicted mathematically Thevenot (2020):

Training Phase:

$$\mathbf{y} = \Phi(\mathbf{W}\mathbf{x}) \circ \mathbf{m}, \quad m_i \sim \text{Bernoulli}(p)$$

Testing Phase:

$$\mathbf{y} = (1-p)\Phi(\mathbf{W}\mathbf{x}), \tag{8.1}$$

where

- Φ: An activation function.
- $W(x)$: The matrix including bias.
- $\circ$: Element-wise multiplication.
- p: Dropout probability.

Mathematically, each neuron's probability of being omitted follows a Bernoulli distribution with probability p.

This results in an element-wise multiplication between the neuron (layer) vector and a mask, where each element in the mask is a random variable governed by the Bernoulli distribution. The mask is a vector of i.i.d Bernoulli random variables.

In the testing phase dropout is not applied, so all neurons remain active. To adjust for the increased information compared to the training phase, we weight the output by the probability of each neuron being retained. Therefore, the probability that a neuron is not dropped is $1-p$.

8.6 ANN vs DNN

In this dissertation, we compare the performance and characteristics of artificial neural networks (ANNs) and deep neural networks (DNNs) in option pricing, with the predominant goal of evaluating their potential as viable alternatives to traditional pricing methods. Table 8.1 presented highlights key differences between ANNs and DNNs across several critical aspects, including complexity, computational requirements, and training performance. These comparisons align with our primary objectives: to assess the suitability of neural networks for capturing complex patterns. We also identify and discuss configurations such as network depth, units per layer, and regularization techniques, that optimize their performance. The table also emphasizes how the computational trade-offs and structural choices of neural networks *generally* influence their ability to generalize and handle complex financial data, thereby providing a foundation for addressing the dissertation's core research questions to be investigated in the results section.

Aspect	DNN (Deep Neural Network)	ANN (Artificial Neural Network)
Complexity and Capacity	• Layer Units: 3 hidden layers of 64 units each • Learning Capacity: DNN exhibitis a higher capacity with more units per layer, allowing improved capturing of complex data patterns. • Complexity: Higher complexity due to more units per layer • We also consider the addition of another 64 unit layer to the DNN and discuss the results	• Layer Units: 3 hidden layers of 32 units each. (An ANN requires atleast 1 hidden layer, and we will be considering the scenario with 3 hidden layers for comparability to DNN) • Learning Capacity: Lower capacity than DNN due to fewer units per layer • Complexity: Simpler due to fewer units per layer
Computational Requirements	• Higher computational requirements; requires powerful hardware and longer training time	• Lower computational requirements; suitable for standard hardware and shorter training time
Training and Performance	• Training Time: Longer due to more parameters • Performance: Better on complex tasks due to higher capacity but more prone to overfitting without regularization	• Training Time: Shorter due to fewer parameters • Performance: Suitable for simpler tasks; performs better on smaller datasets
Summary	Higher capacity, complexity, and suited for intricate tasks with more computational resources; prone to overfitting- this is addressed with dropout layers.	Lower complexity, faster training, suited for straightforward tasks with fewer computational resources.

Table 8.1: Comparison between DNN and ANN

8.7 Relative Error Formulas and Explanation

In our results section, we will also be considering the following error terms in the "Ideal Case Scenario".

8.7.1 1. Relative Error Formula for ANN

$$\text{error\_ANN}[i] = \left| \frac{\text{ANN}[i] - \text{CRR}[i]}{\text{CRR}[i]} \right| \tag{8.2}$$

Explanation:

- ANN$[i]$ is the ANN model's prediction for the i^{th} test case.
- CRR$[i]$ is the true value computed using the CRR method.
- The difference (ANN$[i]$ − CRR$[i]$) represents the absolute error in the ANN model.
- Dividing by CRR$[i]$ normalizes the error relative to the magnitude of the true value.
- Taking the absolute value ensures all errors are positive.

8.7.2 2. Relative Error Formula for DNN

$$\text{error\_DNN}[i] = \left| \frac{\text{DNN}[i] - \text{CRR}[i]}{\text{CRR}[i]} \right| \tag{8.3}$$

This formula follows the same logic but uses predictions from the DNN model (DNN$[i]$) instead.

8.7.3 3. Error Sensitivities

The code computes relative errors for sensitivities to:

- **Share Price** (S),
- **Volatility** (σ),
- **Time to Maturity** (T).

We present the neural network algorithm for vanilla American options, a crucial algorithm for this dissertation.

Algorithm 7: Neural Network Algorithm for Vanilla American Options - (Anderson & Ulrych, 2023)

---

1. **Generate Training Data**
   - Define Parameters: Set a range for the stock prices (S), strike prices (K), risk-free rates (r), volatilities (σ), and time to maturities (T).
   - Simulate the Market scenarios: Randomly generate a set of initial conditions for S, K, r, σ, and T within the specified ranges.
   - Calculate Option Prices: Choose a method (CRR or LSMC) to calculate the true American option prices for the generated market scenarios. These are our base values we will use as a point of comparison with our neural network (ANN/DNN) results.

1.2. **Normalize the data**

   - Scaling: Using a scaler function of choice (Standard Scaler or MinMax Scaler) Normalize S, K, r, σ, and T.

2. **Neural Network Design**

2.1. Define the architecture of the chosen neural network
   - Input Layer: Accept the 5 inputs of S, K, r, σ, and T.
   - Hidden Layers:
     - Add multiple dense layers with activation functions e.g., ReLU or LeakyReLU.
     - Hidden layers can be e.g., 2-4 layers with between 32-64 neurons each. This depends on the chosen NN (ANN or DNN).
   - Output Layer: A single neuron with a linear activation (in the case of e.g. ReLU) to output the option price.
   - Loss function: e.g. Mean-Squared Error (MSE) to assess the discrepancy between the predicted option price and the true value.
   - Optimizer: e.g. Adam optimizer for efficient training of the NN.

3.2. **Train the Model**

   - Split Data: Divide the data into training and validation sets.
   - Fit the Model:
     - Train the model on the training data.
     - Monitor the training process on the validation data.

4. **Model Evaluation**

4.1. Assess the model performance
   - Metrics: Apply error metrics to assess the model's performance on the validation set.

5. **Prediction**

5.1. Predict the Option Prices

* Scenarios: Input new market scenarios (ITM, ATM, OTM) with new sets of S, K, r, σ, T into the trained model.
* Predict: Obtain the predicted American Option prices, considering early exercise.

Practical Component

Chapter 9

Introduction to Results

Prior to our main section of results, we will be covering the following necessary item required for the understanding of our results.

- We will provide a simple numerical example outlining the process that will be followed for our main results. This will give the reader an understanding of the complete method followed, prior to adding more complex scenarios in the main results section.

Our main results will be divided into 3 main chapters:

1. Constant volatility results
2. Stochastic volatility results
3. Time Performance

This split is done with the intention of highlighting the key objectives of our reseach:

- This dissertation aims to demonstrate the potential of neural networks as a viable alternative to traditional methods for option pricing. This is done by use of a numerical results table and scatter plots to show the performance of neural networks in comparison to traditional 'benchmark' methods.
- The disseration also aims to evaluate whether the choice of input parameters significantly influences neural network performance in option pricing. This is achieved by:
  - **Scaler Variation:** Comparing sensitivity graphs generated using different scalers, such as MinMax and StandardScaler.
  - **Testing Dropout Layers:** Examining the impact of varying dropout layers through sensitivity analyses.
  - **Ideal Parameters Scenario:** Presenting an ideal parameters case, combining insights from the above analyses and prior research to identify optimal parameter settings for comparing traditional Cox-Ross-Rubinstein (CRR) models with artificial and deep neural network (ANN/DNN) architectures.
    We will depict a scatterplot, sensitivity graphs, and errors from these results.
- The dissertation also aims to explore the performance of neural networks in specific areas, focusing on:

- **Unstable Market Environments:** Using the Heston Stochastic Volatility model, the dissertation demonstrates the robustness of neural networks in handling volatile market conditions.
- **Computational Efficiency:** Training and execution time metrics are analyzed to highlight the superior time efficiency of neural networks compared to traditional benchmark methods.

This dissertation will meet these objectives, specifically highlighting differences between the performance of the artificial 'shallow' neural network structure, and the deep neural network structure.

By providing a thorough analysis of the aforementioned items, we are able to present a logical argument and conclusion to follow in the results section. To better understand and visualize the split of constant volatility vs stochastic volatility, the following figure 9.1 will provide a high-level overview of what is to follow in each section:

Note: The volatility in the 'constant volatility' section is chosen arbitrarilty and provided in the table of numerical results. The volatility in the 'stochastic volatility' section, however, follows the dynamics defined by the Heston stochastic volatility model.

Constant Volatility		Stochastic Volatility	
European	Table of numerical results	American	Table of numerical results
	Scatterplot		Sensitivity graphs
	Sensitivity Graphs		
	Scalar Results • Standard Scalar • MinMax Scalar		
	Dropout Results • One layer • Two layers		
American	Table of numerical results		
	CRR vs ANN/DNN		
	Scatterplot		
	Scalar Results • Standard Scalar • MinMax Scalar		
	Dropout Results • One layer • Two layers		
	LSMC vs ANN/DNN		
	Scatterplot		
	Scalar Results • Standard Scalar • MinMax Scalar		
	Dropout Results • One layer • Two layers		
	Ideal Parameter Scenario		
	Scatterplot		
	Sensitivity Graphs		
	Errors		

Figure 9.1: Breakdown of Results

9.1 Comparison

To perform a fair assessment of the performance across all models, the same range of parameters are used for the training step. The models are then assessed by analyzing training and testing metrics, robustness through in-sample and out-of-sample predictions, the time taken for the training of the model as well as the time required for a trained model to generate option prices.

Table 9.1: Parameter Ranges for Model Comparison

Parameter	**Range**
Initial Stock Price (S_0)	[80, 120]
Strike Price (K)	[80, 120]
Volatility (σ)	[0.1, 0.5]
Maturity (T)	[0.25, 2]
Risk-Free Rate (r)	[0.01, 0.1]

The parameters above are then used to generate data which will then be fed into the various neural network models for American and European options. We tested this using 10 000 samples with the ranges as depicted in the table above.

9.2 A Simple Example: Neural Networks with a European Call Option

In this section, we outline the steps involved in pricing a European option using a neural network model. This process involves generating synthetic training data, utilizing the Black-Scholes model for theoretical pricing, and constructing a neural network for prediction. We will draw upon the research by Cotto (2019). This serves as a representation of the most straightforward example, designed to introduce the reader to the process in a clear and accessible manner, paving the way for the more complex results discussed later in this section.

Packages Utilized

The following Python libraries and tools are used throughout the implementation:

- **numpy**: For numerical computations, including matrix and array manipulations.
- **keras**: To build, train, and evaluate the neural network model.
- **sklearn**: Used for splitting the dataset into training and testing subsets, ensuring proper validation.
- **scipy.stats**: Provides the cumulative distribution function (CDF) of the normal distribution as required by the Black-Scholes model.

Data Generation

We generate synthetic training data through a user-defined function to create values for the following parameters:

- S: Underlying asset price.

- K: Strike price.
- r: Risk-free interest rate.
- σ: Volatility of the stock.
- T: Time to maturity.

This synthetic data is then used as input to another function that utilizes the Black-Scholes model. The function computes theoretical call option prices based on these parameters, providing the training dataset for the neural network.

Neural Network Architecture

The architecture of the neural network used for pricing is summarized in the table below:

Table 9.2: Summary of Neural Network Configuration

Feature	**Description**
Hidden Layer	1 hidden layer
Neurons	64 neurons.
Activation Function	ReLU (Rectified Linear Unit).
Output Layer	Single neuron with linear activation to predict the call option price.
Optimizer	Adam optimizer for gradient-based optimization.
Loss Function	Mean Squared Error (MSE) to minimize prediction error.

Training the Model

The training dataset (`x_train`) and target values (`y_train`) are prepared such that 20% of the data is reserved for validation. The model is trained over 100 epochs using the Adam optimizer, ensuring efficient convergence with minimal computational cost.
Recall: An epoch refers to one complete pass through the entire training dataset during the training process.

Context and Objective

This neural network implementation aims to evaluate the effectiveness of machine learning models in predicting option prices compared to traditional methods, such as the Black-Scholes model. By leveraging a structured architecture and training process (as we've defined above), this approach contributes to the dissertation's broader objective of demonstrating the viability of neural networks in the financial domain, particularly for option pricing.

Results

The Black Scholes call option price graph is depicted as:

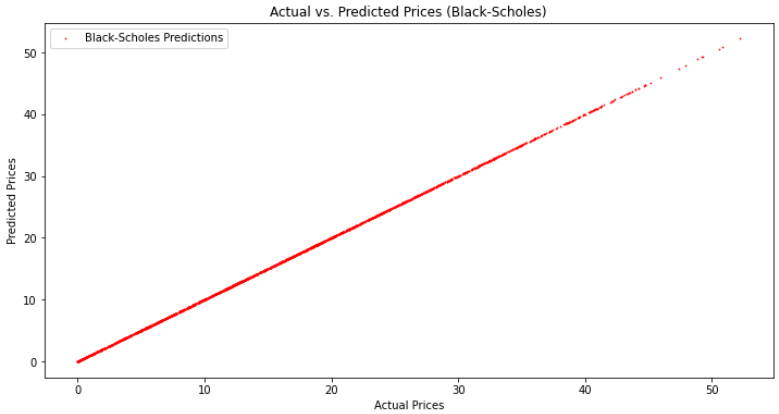


Figure 9.2: Call option price graph using BS

The ANN predicted call price is depicted as:

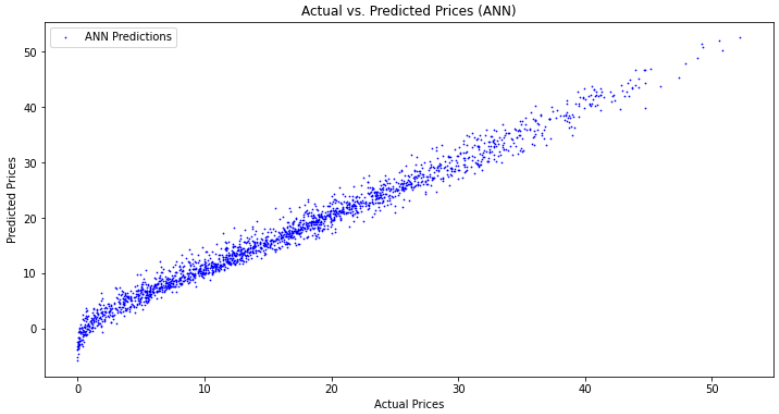


Figure 9.3: Call option price graph using ANN

While the ANN model provides fairly accurate predictions, as depicted in linear trend of the scatter plot, there is a noticeable spread around the ideal line reflecting approximation error. This variance can partly be attributed to the network's structural simplicity, with only one hidden layer and limited depth. The use of a DNN with more layers, and use of LeakyReLU as the activation function might improve the accuracy.

We can infer that while the ANN approximation does not achieve the precision of the Black-Scholes formula, the ANN model generalizes better to scenarios beyond the assumptions of the Black-Scholes model. This includes situations with more complex market dynamics or stochastic volatility.

Chapter 10

Results: Constant Volatility

10.1 European Options

We ran the simulation for European Call options comparing the following methods:

- Cox-Ross-Rubinstein model
- Black-Scholes model
- Artificial neural networks
- Deep neural networks

This was done for the following paths:

- Out-the-Money (OTM)
- at-the-Money (ATM)
- In-the-Money (ITM)

For our standard scenario without the consideration of adjusting scalers and dropout, our parameters are:

- Activation function: ReLU
- ANN structure: 3 layers of 32 neurons
- DNN structure, 3 layers of 64 neurons

Our results are shown in Table 10.1 for K=100, r=0.05:

S	σ	T	Black-Scholes (EU)	CRR (EU)	ANN (EU)	DNN (EU)
80	0.2	1	1.8594	1.8631	1.9080	1.9130
80	0.2	2	5.2318	5.2081	5.2352	5.2623
80	0.4	1	7.5782	7.5836	7.5496	7.7116
80	0.4	2	14.1772	14.1344	14.3439	14.1467
90	0.2	1	5.0912	5.1022	5.3167	5.1307
90	0.2	2	9.9081	9.9008	9.8377	9.7190
90	0.4	1	12.2497	12.2731	12.3034	12.3288
90	0.4	2	19.8561	19.8278	19.9306	19.6550
100	0.2	1	10.4506	10.4306	10.3618	10.4737
100	0.2	2	16.1268	16.0986	16.1157	16.1444
100	0.4	1	18.0230	17.9841	18.0672	18.0633
100	0.4	2	26.2902	26.2366	26.4515	26.2338
110	0.2	1	17.6630	17.6755	17.7086	17.6211
110	0.2	2	23.5910	23.6071	23.6419	23.6263
110	0.4	1	24.7413	24.7485	24.6472	24.7393
110	0.4	2	33.3599	33.3615	33.5380	33.3221
120	0.2	1	26.1690	26.1777	25.9682	25.9871
120	0.2	2	31.9844	31.9687	32.0109	32.0385
120	0.4	1	32.2343	32.2527	32.4443	32.2646
120	0.4	2	41.0020	41.0154	41.2024	40.9819

Table 10.1: Table of Numerical Results for Black-Scholes, CRR, ANN, and DNN using Various Parameters

Figure 10.1: Scatterplot of Black-Scholes vs ANN and DNN predicted prices

The scatter plot illustrates a comparison between the predicted prices from deep neural network (DNN) and artificial neural network (ANN) models against actual prices obtained from the Black-Scholes (BS) model across various scenarios. The points for both DNN and ANN predictions are closely aligned with the perfect fit line, which indicates that both models provide very accurate predictions compared to the BS model. We can infer by the close proximity of the points to the perfect fit line that the NNs have learned the BS pricing function well across different scenarios. Both DNN and ANN predictions lie fairly close to each other and the perfect fit line, which suggests that the models perform similarly in terms of accuracy. From the consistency we can infer that both DNN and ANN can approximate the BS prices well, with no significant deviations between the two models.

The scatterplot results are further supported by our ITM, OTM and ATM scenarios above. We also note that the Black-Scholes price is very close to the CRR binomial price for European options. This was included as an additional source of comparison.

The sensitivity graphs illustrated below (Figure 10.2-10.7) depict how the predicted option prices for European options (using the Black-Scholes model as our benchmark) vary with changes in three factors: volatility, time to maturity, and share price.

10.1.1 Black-Scholes Scaler Sensitivities

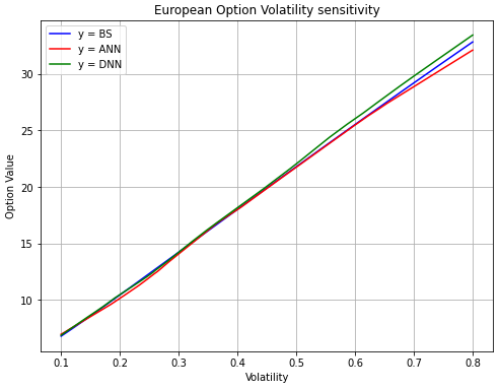


Figure 10.2: Black Scholes European Option Volatility Sensitivity (Standard Scaler)

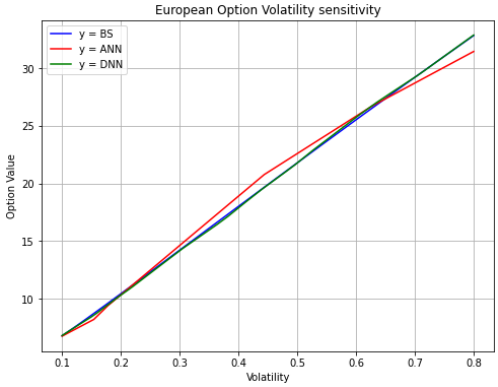


Figure 10.3: Black Scholes European Option Volatility Sensitivity (MinMax Scaler)

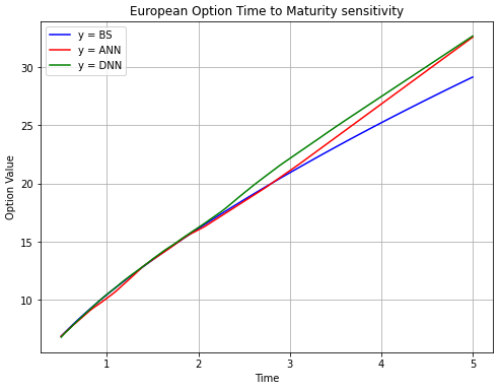


Figure 10.4: Black Scholes European Time to Maturity Sensitivity (Standard Scaler)

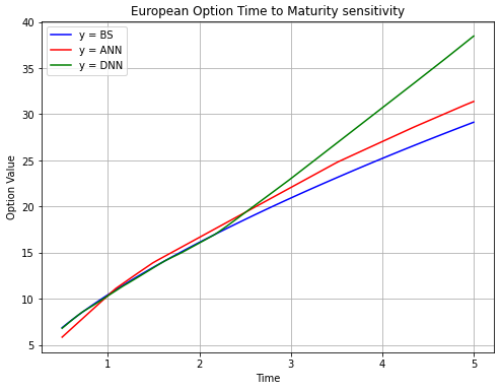


Figure 10.5: Black Scholes European Time to Maturity Sensitivity (MinMax Scaler)

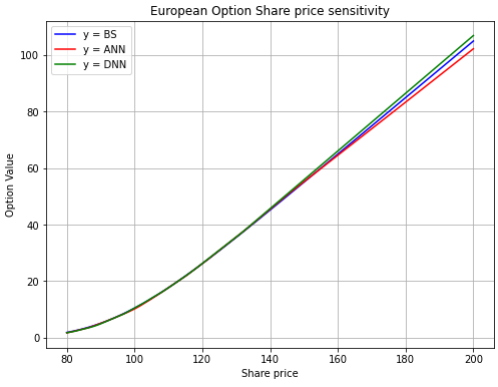


Figure 10.6: Black Scholes European Option Share Sensitivity (Standard Scaler)

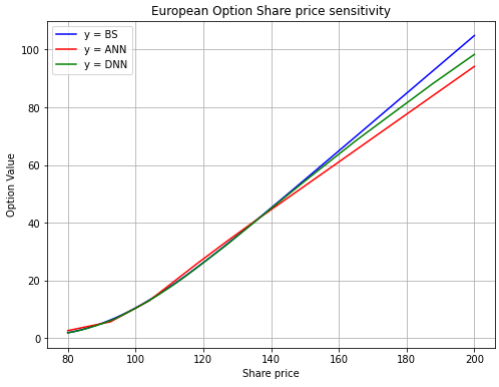


Figure 10.7: Black Scholes European Option Share Sensitivity (MinMax Scaler)

Volatility
If we refer to Figures 10.2, 10.3 we see that all three models (BS, ANN, and DNN) exhibit a positive, linear relationship, where the value of the option increases as volatility rises. This is consistent with our expectations: a higher volatility increases the possible range of outcomes, which raises the option's value. The results from ANN and DNN align near perfectly to the Black-Scholes model, indicating that the neural networks are capable of accurately approximating the option pricing function.

Time to Maturity
The above graphs (Figure 10.4, 10.5) show the sensitivity of the option value for various times to maturity (T). A general positive trend is observed, with the option value rising as time to maturity increases. With BS model as the benchmark, the ANN and DNN are more or less exactly on the same plotting. However we note deviations between the models, particularly between the BS model and the neural networks as maturity increases. This occurs around approximately T=3, and is due to the fact that for T=3-5 the data is out of our training range. It is an indication of how the models react to unseen data.

Option Share Price
The last graphs (Figure 10.6, 10.7) show the sensitivity of the option value to changes in the share price . We note a clear upward, convex relationship, where the option value increases with the share price. This is expected behavior for European call options. Based on the overlap of the three models (BS, ANN, DNN) we deduce that both ANN and DNN models closely follow the CRR model's share price sensitivity. They accurately capture the non-linear relationship between share price and option value.

The performace using a MinMax scaler is near identical to that of the Standard scaler, for all three sensitivity graphs: volatility, time to maturity, option share price. We take note of the deviation in the Time to Maturity sensitivity graph around T=3. We note that the ANN graph predicts higher option values than the CRR model, particularly as time to maturity increases. This could imply that the ANN model is more sensitive to time as a factor in option value, and could potentially be overestimating the effect of longer maturities. The DNNs predictions lie between those of the BS and ANN models, acting as a middle ground. The DNNs values are closer to the BS predictions at shorter maturities but diverge at longer maturities. This implies that the DNN may be slightly better at capturing the relationship between time and option value, even imperfectly.

10.1.2 European Dropout Function Results

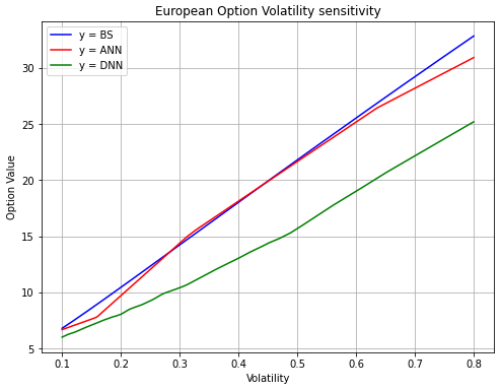


Figure 10.8: Black Scholes European Option Volatility Sensitivity (1 Layer Dropout)

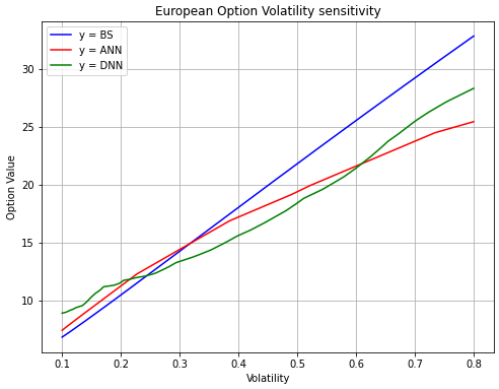


Figure 10.9: Black Scholes European Option Volatility Sensitivity (2 Layers Dropout)

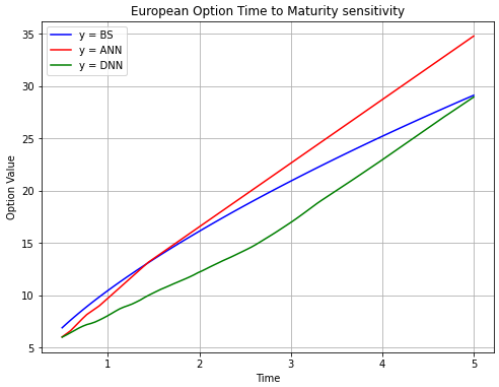


Figure 10.10: Black Scholes European Time to Maturity Sensitivity (1 Layer Dropout)

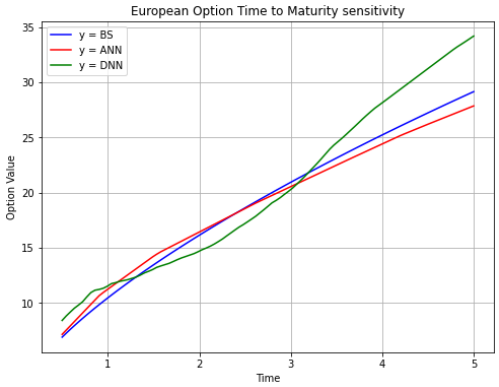


Figure 10.11: Black Scholes European Time to Maturity Sensitivity (2 Layer Dropout)

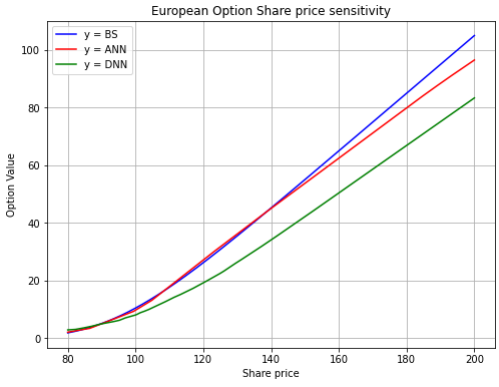


Figure 10.12: Black Scholes European Option Share Sensitivity (1 Layer Dropout)

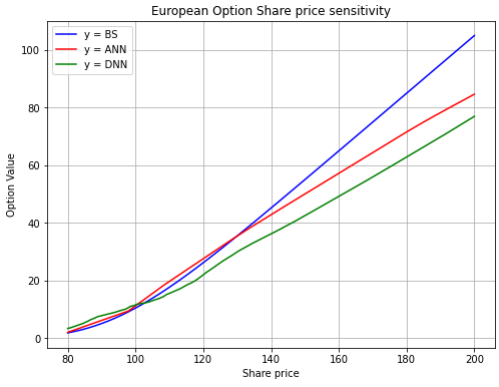


Figure 10.13: Black Scholes European Option Share Sensitivity (2 Layers Dropout)

In the 1-layer dropout configuration (Figure 10.8), both ANN and DNN exhibit a roughly linear sensitivity to volatility, with the ANN aligning somewhat with the Black-Scholes (BS) model. The DNN notably underperforms, producing lower sensitivity values compared to both the ANN and BS models, suggesting that a single dropout layer limits the DNN's ability to fully capture volatility sensitivity.

In the 2-layer dropout configuration (Figure 10.9), there is minimal improvement in NN volatility sensitivity. While the DNN shows slight improvements with values more aligned with the BS model, the ANN's performance slightly declines. Both models continue to struggle with out-of-range data.

For time sensitivity (Figure 10.10, 10.11), the ANN and DNN perform significantly better with the 2-layer dropout configuration, even for longer maturities (out-of-range data). The ANN values align closely with the BS model, while the DNN shows a very slight improvement compared to its 1-layer performance, indicating better generalization.

In terms of share price sensitivity (Figure 10.12, 10.13), both ANN and DNN models display a linear trend for the 1-layer and 2-layer configurations. The DNN underestimates option values from the BS model, particularly at higher share prices. The ANN aligns more closely with the BS model than the DNN, though it still falls short, especially in the 2-layer configuration.

Overall, the NNs do not accurately capture the sensitivities in both the single and double layer configurations for European options.

10.2 American Options

We compare the following methods with American option Pricing:

- Cox-Ross-Rubinstein
- Artificial neural networks (using comparison base of CRR)
- Deep neural networks (using comparison base of CRR)
- Least-Squares Monte Carlo
- Artificial neural networks (using comparison base with LSM)
- Deep neural networks (using comparison base with LSM)

This was done for the following paths:

- At-the-Money (ATM)
- In-the-Money (ITM)
- Out-the-Money (OTM)

For our standard scenario without the consideration of adjusting scalers and dropout, our parameters are:

- Activation function: ReLU
- Gradient Descent: Standard stochastic gradient descent
- ANN structure: 3 layers of 32 neurons
- DNN structure, 3 layers of 64 neurons

Our results are shown in Table 10.2 for K and r chosen arbitrarily, with K=100, r=0.05:

S	σ	T	CRR (AM)	ANN (AM)	DNN (AM)	LSMC	LSMC (ANN)	LSMC (DNN)
80	0.2	1	1.8306	1.8394	2.1098	1.8306	1.8274	2.0511
80	0.2	2	5.2247	5.5308	5.0925	5.0907	5.5022	5.3310
80	0.4	1	7.5188	7.7044	7.4324	7.5781	7.4709	7.7702
80	0.4	2	14.2448	14.4469	14.3799	14.7199	14.3912	14.4918
90	0.2	1	5.0812	4.9218	5.0798	5.1781	4.6886	5.2712
90	0.2	2	9.9324	9.9464	9.7326	10.2141	10.0114	9.9588
90	0.4	1	12.0965	12.0965	12.7252	12.4966	12.0126	12.1287
90	0.4	2	19.9255	20.0557	20.0352	19.5832	19.6727	20.3313
100	0.2	1	10.4107	10.5521	10.4093	10.5531	9.8916	10.5319
100	0.2	2	16.0704	16.033	15.8576	16.4935	15.9516	16.3548
100	0.4	1	17.4951	17.8232	18.0319	17.6961	17.7341	18.1923
100	0.4	2	26.1831	26.5685	26.5634	26.5521	26.3876	26.8361
110	0.2	1	17.6851	17.4121	17.7571	17.6232	17.3099	17.7159
110	0.2	2	23.5945	23.9326	23.4654	23.9757	23.2177	23.8652
110	0.4	1	24.7457	24.1567	24.6493	24.6773	24.3779	24.9329
110	0.4	2	33.4509	33.3383	33.7788	33.2449	33.1024	34.0974
120	0.2	1	26.1715	25.2929	26.2942	26.0413	25.7544	26.0752
120	0.2	2	31.9799	32.3142	32.1997	32.1957	31.9463	32.4035
120	0.4	1	32.2432	32.0882	32.3872	32.0487	32.2246	32.1548
120	0.4	2	40.9447	41.0744	40.2343	41.0243	41.0223	41.8759

Table 10.2: Comparison of American option prices using CRR, ANN, DNN, and LSMC methods for various parameters

10.3 CRR vs ANN and DNN

Note: Our initial results were obtained without using a scaler function. The graphs for this are provided in the Appendix.

Figure 10.14: Scatterplot of CRR vs ANN and DNN predicted prices

The scatter plot illustrates a comparison between predicted prices from deep neural network (DNN) and artificial neural network (ANN) models against actual prices obtained from the Cox-Ross-Rubinstein (CRR) model across various scenarios. The points for both DNN and ANN predictions are closely aligned with the perfect fit line, which indicates that both models provide very accurate predictions to the CRR model. We can infer by the proximity of the points to the perfect fit line that the NNs have learned the pricing function well across different scenarios. Both DNN and ANN predictions lie close to each other and the perfect fit line, which suggests that the models perform similarly in terms of accuracy. We can infer that both ANN and DNN architectures are effective in capturing the patterns in the CRR-based prices based on the comparison of the predictions to the CRR model.

This observation is further supported by our numerical results above. We also note that the CRR matches the LSMC pricing method extremely well.

The sensitivity graphs below illustrate how the predicted option prices for American options (using the CRR model as our benchmark) vary with changes in three factors: volatility, time to maturity, and share price.

10.3.1 CRR Scaler Sensitivities

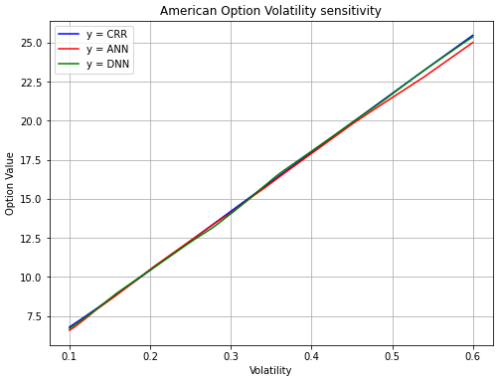


Figure 10.15: CRR American Option Volatility Sensitivity (Standard Scaler)

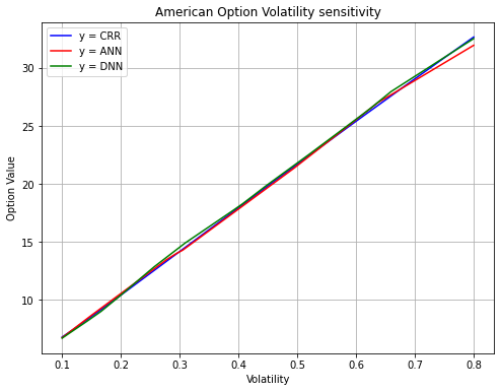


Figure 10.16: CRR American Option Volatility Sensitivity (MinMax Scaler)

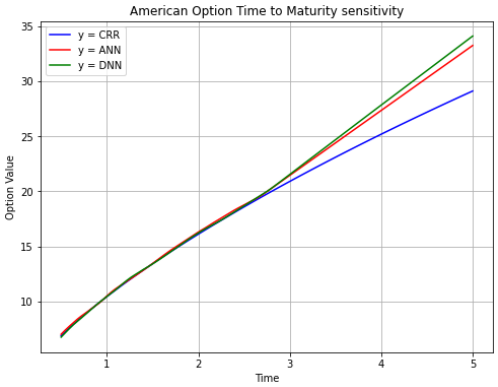


Figure 10.17: CRR American Option Time to Maturity Sensitivity (Standard Scaler)

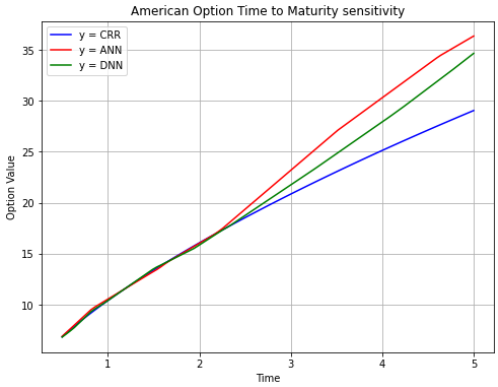


Figure 10.18: CRR American Option Time to Maturity Sensitivity (MinMax Scaler)

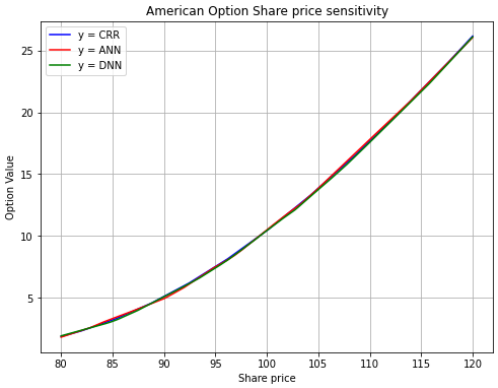


Figure 10.19: CRR American Option Share Price Sensitivity (Standard Scaler)

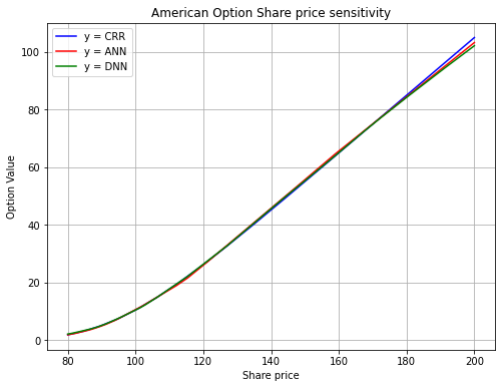


Figure 10.20: CRR American Option Share Price Sensitivity (MinMax Scaler)

Volatility
All three models (CRR, ANN, and DNN) exhibit a positive, linear relationship, where the value of the option increases as volatility rises, as seen in Figures 10.15 and 10.16. This is consistent with our expectations: a higher volatility increases the possible range of outcomes, which raises the option's value.

Time to Maturity
The above graphs (Figures 10.17 and 10.18) shows the sensitivity of the option value for various times to maturity (T). A general positive trend is observed, with the option value rising as time to maturity increases. With CRR model as the benchmark, the ANN and DNN are more or less exactly on the same plotting. However we note deviations between the models, particularly between the CRR model and the neural networks as maturity increases. This occurs around approximately T=3, and is due to the fact that for $T \in [3, 5]$ the data is out of our training range. It is an indication of how the models react to unseen data.

Option Share Price
The last graphs (Figures 10.19 and 10.20) shows the sensitivity of the option value to changes in the share price. We note a clear upward, convex relationship, where the option value increases with the share price. This is expected behavior for American call options. Based on the overlap of the three models (CRR, ANN, DNN) we deduce that both ANN and DNN models closely follow the CRR model's share price sensitivity. They accurately capture the non-linear relationship between share price and option value.

The performace using a MinMax scaler is near identical to that of the Standard Scaler, for all three sensitivity graphs: volatility, time to maturity, option share price. We take note of the deviation in the Time to Maturity sensitivity graph around T=3. We note that the ANN graph predicts higher option values than the CRR model, particularly as time to maturity increases. This could imply that the ANN model is more sensitive to time as a factor in option value, and could potentially be overestimating the effect of longer maturities. The DNNs predictions lie between those of the CRR and ANN models, acting as a middle ground. The DNNs values are closer to the CRR predictions at shorter maturities but diverge at longer maturities. This implies that the DNN may be slightly better at capturing the relationship between time and option value, even imperfectly.

10.3.2 CRR Dropout

We consider two scenarios:

- 1-dropout layer (30%)
- 2-dropout layers (30%)

The graphs below (Figures 10.21 - 10.26) illustrate the sensitivity of American option prices to changes in volatility, time to maturity, and share price across three models: CRR, artificial neural network (ANN), and deep neural network (DNN) when 1-layer and 2-layer dropouts are added.

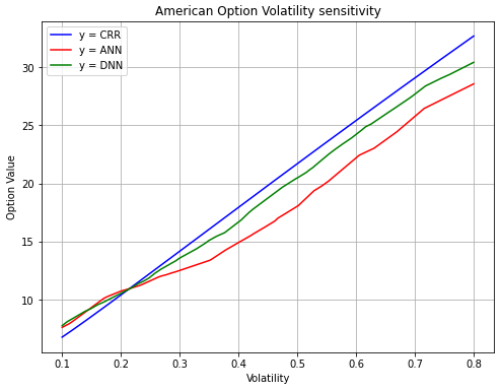


Figure 10.21: CRR American Option Volatility Sensitivity (1 Layer Dropout)

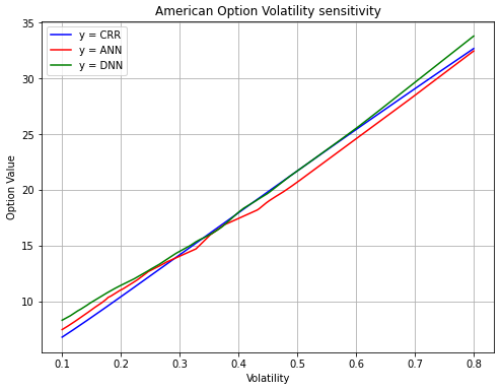


Figure 10.22: CRR American Option Volatility Sensitivity (2 Layers Dropout)

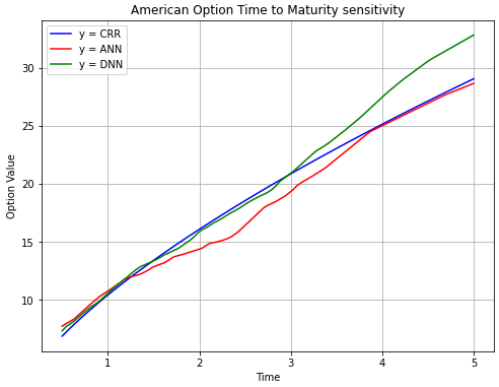


Figure 10.23: CRR American Time to Maturity Sensitivity (1 Layer Dropour)

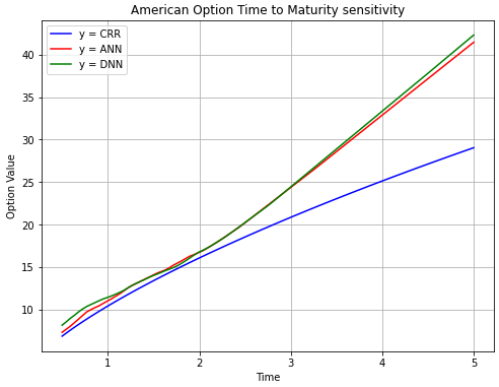


Figure 10.24: CRR American Time to Maturity Sensitivity (2 Layer Dropout)

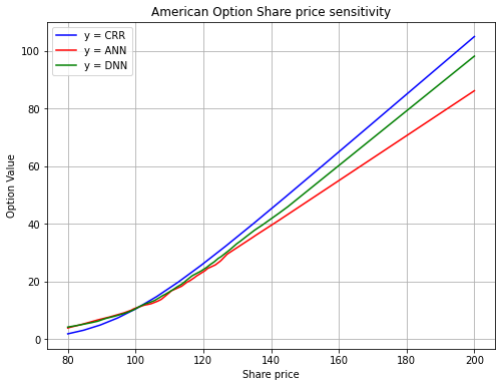


Figure 10.25: CRR American Option Share Sensitivity (1 Layer Dropout)

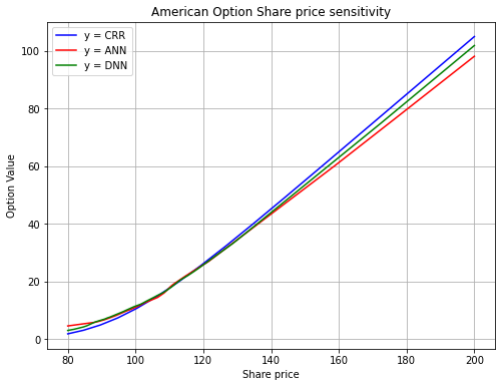


Figure 10.26: CRR American Option Share Sensitivity (2 Layers Dropout)

In the 1-layer dropout volatility graph (Figure 10.21), both the ANN and DNN exhibit sensitivity to volatility that somewhat aligns with the CRR model. However, there are notable deviations, with the DNN outperforming the ANN in accuracy. While both neural networks capture the overall trend, they fall short of completely matching the CRR model's sensitivity to volatility.

When analyzing the 2-layer dropout volatility graph (Figure 10.22), both the ANN and DNN align much more closely with the CRR model. This alignment is near perfect, even for out-of-range data that the model was not trained on. This demonstrates that introducing an additional dropout layer significantly enhances the neural network's performance, particularly in capturing sensitivity to volatility.

A similar trend is observed in the share price sensitivity graphs (Figure 10.25, 10.26). With the inclusion of a second dropout layer, the neural network provides an almost exact approximation to the CRR values. The DNN consistently performs better than the ANN across all dropout configurations for share price sensitivity.

For the time sensitivity graphs (Figure 10.23, 10.24), neither ANN nor DNN approximates the CRR values as accurately for both dropout configurations. While the ANN and DNN values are closely aligned with each other, they show a significant divergence from the CRR model at longer times to maturity, particularly at $T \in [3, 5]$. This is likely because the neural networks were not trained on data for these time intervals, causing the models to struggle with out-of-range predictions.

10.4 LSMC vs ANN and DNN

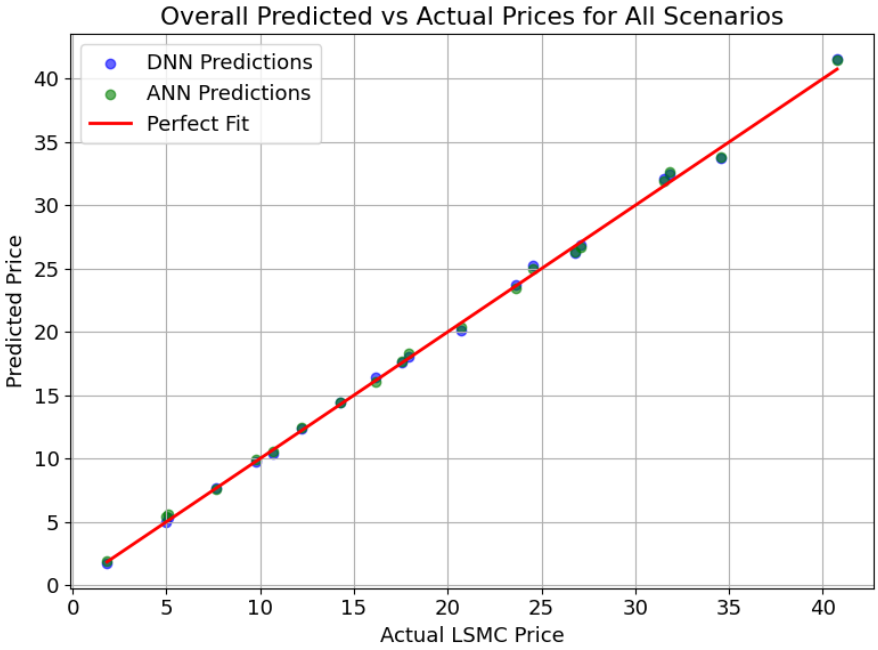


Figure 10.27: Scatterplot of LSMC vs ANN and DNN predicted prices

The scatter plot illustrates a comparison between predicted prices from deep neural network (DNN) and artificial neural network (ANN) models against actual prices obtained from the LSMC model across various scenarios. The points for both DNN and ANN predictions are closely aligned with the perfect fit line, which indicates that both models provide very accurate predictions compared to the LSMC model. We can infer by the proximity of the points to the perfect fit line that the neural networks have learned the pricing function well across different scenarios. Both DNN and ANN predictions lie in close proximity to each other and the perfect fit line, which suggests that the models perform similarly in terms of accuracy. We infer that both ANN and DNN architectures are effective in capturing the patterns in the LSMC-based prices.

The graphs below (Figures 10.28 - 10.33) illustrate the sensitivity of American option prices to changes in volatility, time to maturity, and share price across three models: Least Squares Monte Carlo (LSMC), artificial neural network (ANN), and deep neural network (DNN). Our analysis includes the use of both the Standard Scalar and the Min Max Scalar, due to their similarities in graph sensitivities.

10.4.1 LSMC Scaler Sensitivities

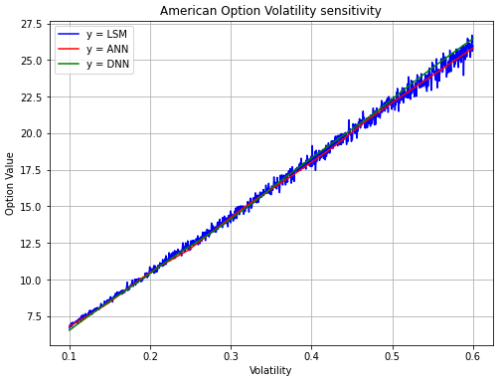


Figure 10.28: LSMC American Option Volatility Sensitivity (Standard Scaler)

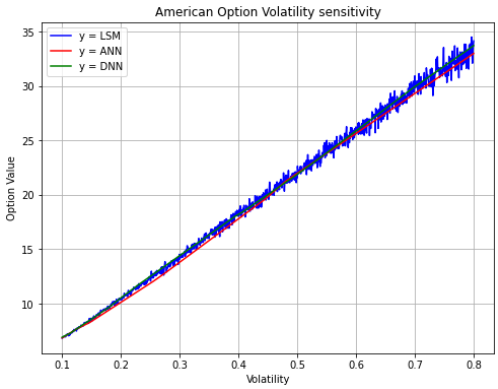


Figure 10.29: LSMC American Option Volatility Sensitivity (MinMax Scaler)

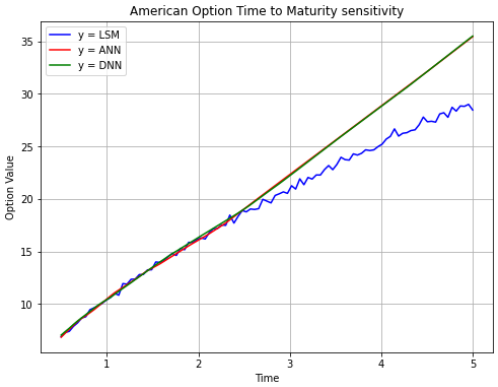


Figure 10.30: LSMC American Option Time to Maturity Sensitivity (Standard Scaler)

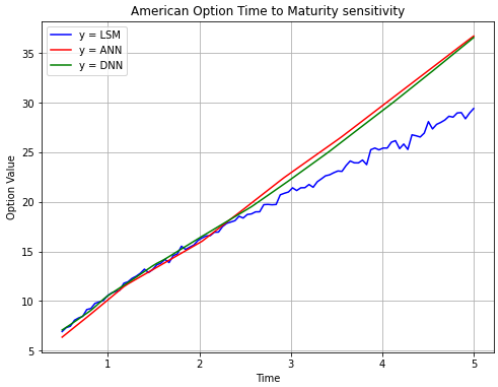


Figure 10.31: LSMC American Option Time to Maturity Sensitivity (MinMax Scaler)

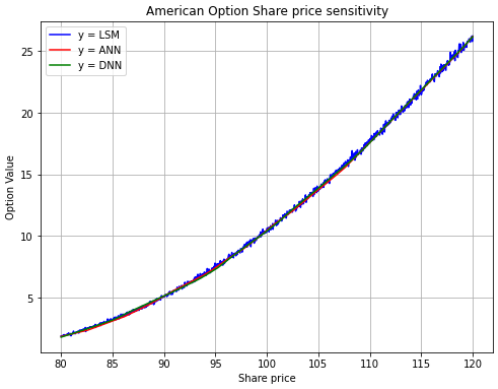


Figure 10.32: LSMC American Option Share Price Sensitivity (Standard Scaler)

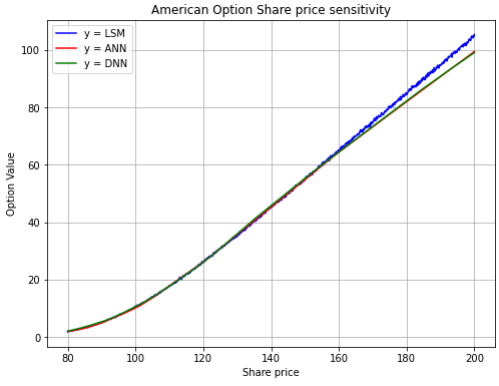


Figure 10.33: LSMC American Option Share Price Sensitivity (MinMax Scaler)

Volatility
In these graphs (Figure 10.28, 10.29), option values increase as volatility rises, which aligns with our expectations of the model. Higher volatility generally leads to higher option values because it increases the chances of favorable price movements. The LSM model exhibits significant fluctuations (noise) as volatility increases. These fluctuations suggest that the Monte Carlo approach, which relies on random sampling, introduces variability into the option value predictions, especially at higher levels of volatility. If the underlying share price is volatile, as in our case by using GBM a stochastic model, it introduces variability into the true estimated value of the option. Both ANN and DNN models depict a smoother relationship between volatility and the option value, indicating that they are less affected by random sampling noise. The lines for ANN and DNN closely overlap, which suggest that both NNs effectively capture the volatility sensitivity of option values without the noise observed in the LSMC model.

Time to Maturity
The graphs (Figure 10.30, 10.31) show the relationship between option value and time to maturity. The LSMC option values show a steady increase as time to maturity grows but with some fluctuations at higher maturities. Both ANN and DNN models produce a smooth, upward-trending lines that depicts a positive relationship between time to maturity and option value. The divergence around T=3 is due to the extrapolated values performing less accurately, as discussed above.

Option Share Price
The graphs (Figure 10.32, 10.33) show the relationship between the option value and the share price. Typically, as the share price increases the call option value also increases as there is a greater chance of the option finishing in the money. Both NN models display a very similar convex pattern to the LSMC method, with a smooth increase in option value as share price rises. ANN and DNN values align closely, which indicates that both NN models are accurately capturing the share price sensitivity.

10.4.2 LSMC Dropout

- 1-dropout layer (30%)
- 2-dropout layers (30%)

The graphs below (Figures 10.34 - 10.39) illustrate the sensitivity of American option prices to changes in volatility, time to maturity, and share price across three models: Least Squares Monte Carlo (LSMC), artificial neural network (ANN), and deep neural network (DNN) when 1-layer and 2-layer dropouts are added.

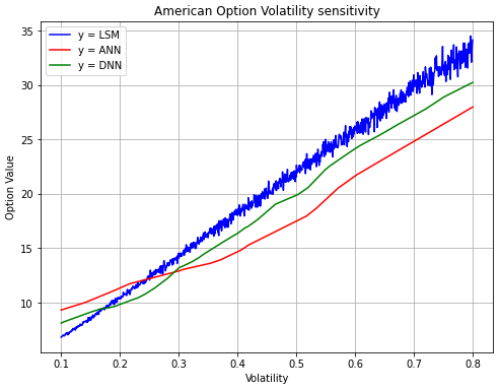


Figure 10.34: LSMC American Option Volatility Sensitivity (1 Layer Dropout)

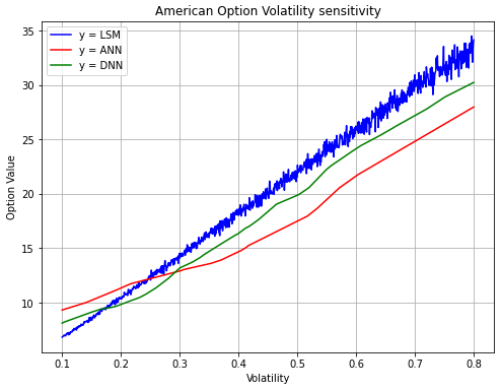


Figure 10.35: LSMC American Option Volatility Sensitivity (2 Layers Dropout)

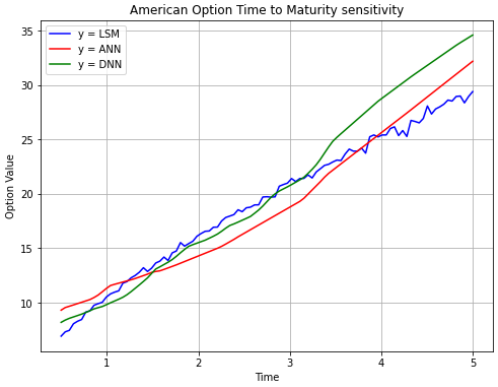


Figure 10.36: LSMC American Time to Maturity Sensitivity (1 Layer Dropour)

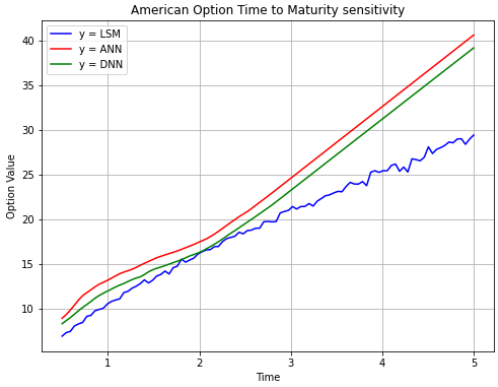


Figure 10.37: LSMC American Time to Maturity Sensitivity (2 Layer Dropout)

Figure 10.38: LSMC American Option Share Sensitivity (1 Layer Dropout)

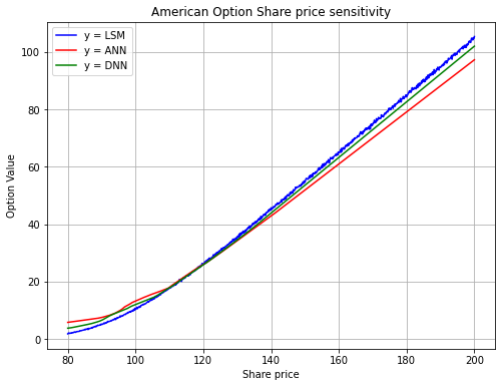


Figure 10.39: LSMC American Option Share Sensitivity (2 Layers Dropout)

The neural networks demonstrated better performance when an additional dropout layer was added, as seen specifically in the volatility and share price sensitivity graphs. For the 1-layer dropout volatility graph (Figure 10.34), it is evident that the DNN values are closer to the LSMC values than the ANN values, indicating that the DNN performed better in capturing the model's behavior. While the neural networks effectively capture the general trend, they lack the capacity to fully align with the LSMC model's sensitivity to volatility, especially for the 1-layer dropout scenario. However, for the 2-layer dropout volatility graph (Figure 10.35), both the ANN and DNN values become closely aligned with the LSMC sensitivity values. This alignment becomes evident, even when analyzing out-of-range data, showcasing improved performance with additional dropout layers.

A similar pattern can be observed in the share price sensitivity graphs (Figure 10.38, 10.39), where the networks provide a closer approximation to the LSMC values with the inclusion of a second dropout layer. Again, the DNN values exhibit a stronger alignment with the LSMC values compared to the ANN values, suggesting enhanced performance.

In the case of time sensitivity graphs (Figure 10.36, 10.37), the neural networks were trained on options with shorter times to maturity. Consequently, the networks lacked the necessary information to accurately predict values for longer time periods, particularly for $T \in [3, 5]$.

For American option pricing, the time parameter introduces additional complexity. The value of an American option can increase significantly as time to maturity increases due to the added flexibility for early exercise. If the neural network has not adequately learned this non-linear behavior, potentially due to a lack of training data for longer maturities, its performance on untrained, out-of-range data will suffer. It is also possible that the models overfit to the shorter maturities in the training data, leading to poor generalization for longer maturities. As a result, the neural network models may produce results that deviate significantly from those of the CRR model, which serves as the baseline for comparison.

Additionally, it was noted that the choice of activation functions might limit the model's ability to predict extreme values if these were not present in the training set. For instance, functions like ReLU are particularly sensitive to outliers, which may impact the overall performance of the networks in these scenarios.

10.5 Ideal Parameters Results

We decided to implement a scenario with the best-case parameters:

- Activation function: LeakyReLU (to account for 'Dying ReLU problem')
- Scaler: MinMax Scaler
- Gradient Descent: Adam
- ANN structure: 3 layers of 32 neurons
- DNN structure, 4 layers of 64 neurons
- Dropout: 2 dropout layers (30%) were used for the DNN structure

We will only be considering the performance of the ANN and DNN for the vanilla American CRR option.

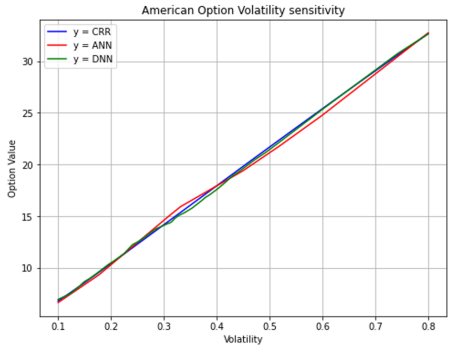


Figure 10.40: Ideal Parameters CRR: Volatility Sensitivity

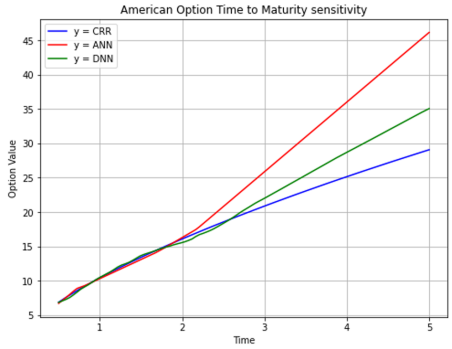


Figure 10.41: Ideal Parameters CRR: Time to Maturity Sensitivity

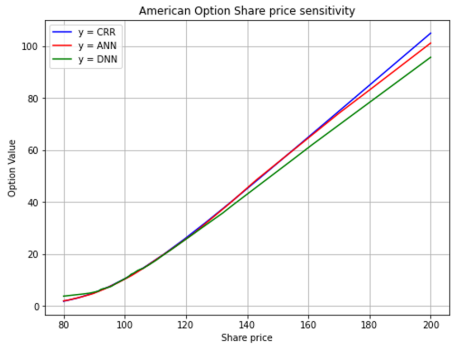


Figure 10.42: Ideal Parameters CRR: Option Share Price Sensitivity

Based on the results presented above, we can conclude that even under the most favorable parameter settings, as extensively researched in this dissertation, the artificial neural network (ANN) exhibits relatively weak performance concerning time on previously unseen data. Several potential reasons for this outcome were discussed earlier, including limitations in the model's adaptability to specific scenarios and the nature of the data itself. While the ANN may not represent an ideal fit under all conditions, it is worth noting that, overall, neural networks perform competitively compared to traditional models, particularly when evaluated against the 'true' values that the models aim to approximate.

The primary purpose of employing neural networks and machine learning methodologies in this dissertation is to explore avenues for improvement over traditional approaches. Traditional models, while often delivering the 'best' results on unseen data, are limited by computational inefficiencies and scalability issues, as discussed earlier in this dissertation. In contrast, machine learning (ML) methods, such as ANNs and DNNs, offer significant advantages in terms of reduced computational time and adaptability. These advantages depict the machine learning based methods as promising for further refinement and application.

10.6 Error

We analyze the error sensitivity graphs (Figures 10.43 - 10.45) for the American CRR model with ANN and DNN (using the updated ideal parameters)

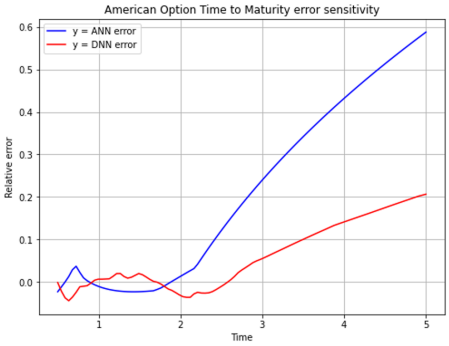


Figure 10.43: American Time to Maturity Error Sensitivity

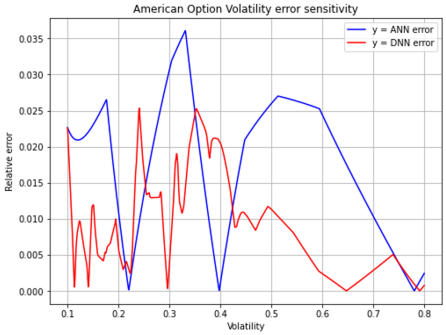


Figure 10.44: American Volatility Error Sensitivity

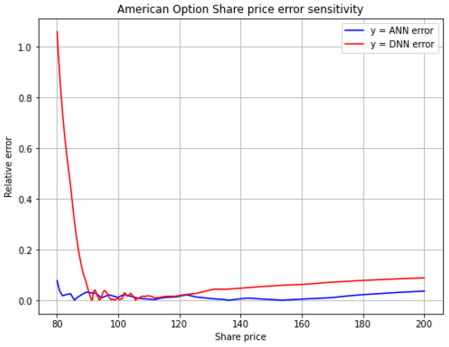


Figure 10.45: American Share Price Error Sensitivity

The error sensitivity graphs presented above depict how errors in American option pricing models vary based on three key parameters: time to maturity, volatility, and share price. These

comparisons are made between artificial neural networks (ANNs) and deep neural networks (DNNs) under varying conditions.

Time to Maturity Error Sensitivity: The first graph focuses on how the error changes as the time to maturity increases. At shorter maturities, both ANN and DNN models exhibit relatively low errors, with the DNN demonstrating slightly more stable error rates. However, as the time to maturity increases, the ANN's error grows significantly, indicating that the ANN struggles to maintain accurate predictions for options with longer maturities. In contrast, the DNN handles this variation more consistently.

Volatility Error Sensitivity: The second graph examines error sensitivity in relation to changes in volatility. At lower levels of volatility, with minimal differences in error rates and slight fluctuations seen in the DNN model. The relative error for the DNN model fluctuates significantly as volatility increases, with several spikes. Despite the fluctuations, DNN generally exhibits smoother error behavior compared to ANN in certain regions, especially at higher volatilities. ANN shows more variability and instability in its error behavior, especially at lower volatility levels, but improves significantly at higher volatilities. DNN demonstrates more consistent and reliable performance across the entire volatility range, with smaller errors at low and moderate volatilities. The instability suggests that ANN and DNN predictions might be more sensitive to volatility changes.

Share Price Error Sensitivity: The third graph evaluates how errors vary with share price changes. The relative error decreases as the share price increases for both the ANN and DNN models. The ANN model maintains a relatively low and stable error for most share prices, whilst the DNN model has a large spike in relative error at very low share prices. The ANN handles these edge cases better, showing greater robustness at low share prices. As the share price increases, the DNN model stabilizes and exhibits error levels comparable to the ANN but remains slightly higher overall. As the share price increases the relative errors of both models converge and remain consistently low, indicating better accuracy in predicting at-the-money or in-the-money option prices.

Overall, the DNN outperforms the ANN in terms of stability and accuracy across most scenarios. For time to maturity, the DNN maintains lower errors, particularly excelling at longer durations, while the ANN's error increases significantly with time, showing poor performance for long-term options. In terms of volatility, the ANN exhibits greater fluctuations, with unstable behavior in mid-volatility ranges, but performs comparably with DNN at very high volatilities, showcasing its ability to adapt in extreme cases. For share price sensitivity, the ANN is more accurate and stable at low share prices (deep out-of-the-money options), while the DNN struggles with large errors in this region; however, the DNN performs better as share prices increase, handling at-the-money and in-the-money scenarios more effectively.

Chapter 11

Results: Stochastic Volatility

11.1 Heston Stochastic Volatility American Option pricing

The following parameters were used for the Heston stochastic volatility model:

- Activation function: LeakyReLU (to account for 'Dying ReLU problem')
- Scaler: MinMax Scaler
- Gradient Descent: Mini-Batch Gradient Descent
- ANN structure: 3 layers of 32 neurons
- DNN structure, 4 layers of 64 neurons
- Dropout: 2 dropout layers (30%) were used for the DNN structure.

The Table 11.1 below depicts the comparison of American Option Prices using methods: LSM Heston, ANN and DNN for various parameters.

S	σ	T	LSM Heston (AM)	ANN (AM)	DNN (AM)
80	0.2	1	7.5150	7.3228	7.5246
80	0.2	2	10.7706	10.2875	10.2618
80	0.4	1	11.9978	11.6977	12.3876
80	0.4	2	15.8040	15.7146	15.6878
90	0.2	1	12.2507	11.6747	11.8413
90	0.2	2	15.7899	15.9447	15.9182
90	0.4	1	17.2459	17.1368	17.4886
90	0.4	2	22.3982	22.3200	22.2952
100	0.2	1	17.8899	17.5050	17.5932
100	0.2	2	22.2786	23.1523	22.7468
100	0.4	1	23.6381	24.1275	24.0208
100	0.4	2	29.8958	29.9501	29.8769
110	0.2	1	24.7370	24.6954	24.6922
110	0.2	2	29.8465	30.5266	30.5008
110	0.4	1	30.8424	31.6789	31.5231
110	0.4	2	37.0393	37.9324	37.7299
120	0.2	1	32.9628	33.0678	32.7777
120	0.2	2	38.8821	38.8819	39.159
120	0.4	1	39.3462	39.5179	39.3542
120	0.4	2	45.6771	46.2993	46.4205

Table 11.1: Comparison of American Option Prices using LSM Heston, ANN, and DNN Methods for Various Parameters

The graphs below (Figure 11.1 - 11.3) illustrate the sensitivity of American option prices to changes in volatility, time to maturity, and share price under the Heston stochastic volatility model compared to artificial neural network (ANN) and deep neural network (DNN) predictions. Our analysis focuses on how effectively the ANN and DNN models approximate the sensitivity trends observed in the Heston model.

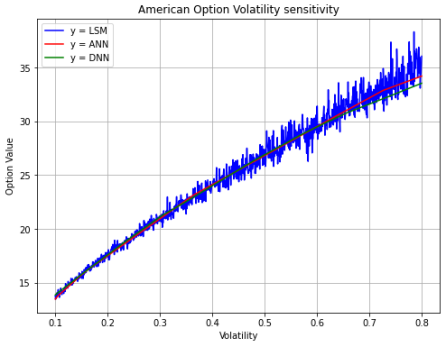


Figure 11.1: Heston American Option Volatility Sensitivity

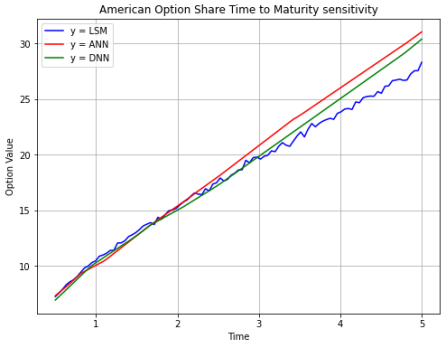


Figure 11.2: Heston American Option Time to Maturity Sensitivity

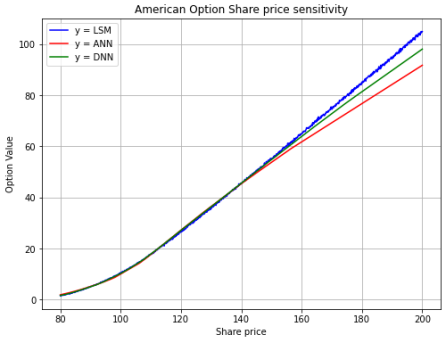


Figure 11.3: Heston American Option Share Price Sensitivity

Volatility sensitivity

The first graph (Figure 11.1) depicts how option prices respond to increasing volatility under the Heston model. As expected, option values increase with rising volatility, reflecting the greater uncertainty and potential for favorable price movements. The Heston model demonstrates noticeable fluctuations in this relationship, attributed to its reliance on stochastic volatility, which

introduces random sampling noise. Both ANN and DNN models depict a smooth relationship between volatility and the option value, indicating their reduced susceptibility to random noise. The lines for ANN and DNN closely overlap, suggesting that both NNs efectively capture the volatility sensitivityof option values, without the noise observed for the Heston model.

Time to Maturity Sensitivity
The second graph (Figure 11.2) shows the relationship between option prices and time to maturity. Under the Heston model, we observe a steady upward trend, as longer maturities provide greater opportunities for favorable price movements, increasing option values. Both ANN and DNN models also display this upward trend, effectively capturing the time sensitivity demonstrated by the Heston model. Whilst both NN models diverge slightly around $T = 3$, the DNN model provides a closer match to the Heston model's sensitivity for longer maturities than the ANN model. The discrepancies for the ANN model may stem from limitations in capturing the more complex interactions of the Heston dynamics. Overall the NNs accurately capture the Heston model's sensitivity to time, despite the slight divergence for out-of-sample data values

Share Price Sensitivity
The third graph (Figure 11.3) examines how option prices vary with share price changes. The Heston model shows a clear, upward-sloping relationship, with higher share prices corresponding to higher option values due to the intrinsic value of the option increasing. Both ANN and DNN models replicate this trend effectively, aligning closely with the Heston model's sensitivity for in-sample share prices, with slightl divergence for out-of-sample data. This indicates that both ANNs and DNN models are accurately capturing the share price sensitivity

The ANN and DNN models provide a reasonable approximation of the sensitivity trends observed in the Heston model across volatility, time to maturity, and share price. The DNN generally exhibits better performance in capturing the nonlinear dynamics of time and volatility sensitivity, which we anticipate is due to it benefiting from its deeper architecture and dropout layers. This is further supported by the table depicting the actual predictions of ANN, DNN and LSM Heston.

The results from applying dropout are provided in the Appendix for further reference.

Chapter 12

Results: Time Performance

Table 12.1 below presents a detailed breakdown of computing times for both training and runtime across various models and methods investigated in this dissertation. These include artificial neural networks (ANN), deep neural networks (DNN), and traditional option pricing models such as the Black-Scholes (BS) model, the Cox-Ross-Rubinstein (CRR) method, and Least Squares Monte Carlo (LSMC) with the Heston stochastic volatility framework. All times are reported in seconds, highlighting the computational demands of each method.

	Time to Train		**Run Time**		
	ANN	**DNN**	**BS**	**ANN**	**DNN**
European	19.96353	28.55916	0.06138	0.06619	0.08061
American	**ANN**	**DNN**	**CRR**	**ANN**	**DNN**
	21.48746	22.57591	0.10869	0.05365	0.08372
	ANN	**DNN**	**LSMC**	**ANN**	**DNN**
	20.38508	21.85418	0.08425	0.03735	0.04838
Heston (American)	**ANN**	**DNN**	**LSMC (Heston)**	**ANN**	**DNN**
	45.95497	57.94413	0.19455	0.12883	0.17038

Table 12.1: Time performance results for European, American, Heston (American)

General Observations

Training the neural network models (ANN and DNN) understandably takes the longest time due to the iterative optimization process required to adjust weights and biases. However, once the models are trained, the runtime for obtaining option prices is significantly reduced compared to traditional methods, particularly for more complex scenarios. This reduction demonstrates the efficiency of using neural networks as a viable alternative for option pricing once the initial computational investment in training is completed.

For American options, the increased training time relative to European options underscores the complexity associated with early exercise features. For example, ANN training time for American options (21.48746 seconds) is 7.6% higher than for European options (19.96353 seconds). Similarly, DNN training time for American options (22.57591 seconds) is 21.1% higher than for European options (28.55916 seconds). These differences highlight the increased computational complexity required to model the early exercise feature.

The considerable increase in training time for the stochastic volatility scenario modeled using the Heston framework further reflects the computational intensity required for scenarios with more realistic market dynamics. ANN training time for Heston American options is 45.95497 seconds, a 125% increase compared to regular American options (21.48746 seconds). Similarly, DNN training time for Heston American options is 57.94413 seconds, a 157% increase compared to the standard American DNN model (22.57591 seconds).

European Options

For European options[5], the BS model demonstrates a notably low runtime of 0.06138 seconds. This result is expected, as the BS model is an analytical solution and does not require iterative computations. Comparatively, the ANN and DNN models exhibit slightly higher runtimes of 0.06619 seconds and 0.08061 seconds, respectively. The ANN runtime is 7.8% higher than the BS model, while the DNN runtime is 31.3% higher than the BS model. Although these times are marginally longer than the BS model, they highlight the practicality of neural networks in efficiently pricing European options, even with their relatively simple structure.

American Options

For American options, the CRR method exhibits a runtime of 0.10869 seconds, which is significantly higher than the BS model. This increase is due to the iterative lattice structure of the CRR model, which accounts for the early exercise feature by evaluating potential decisions at each step in the binomial tree.

In contrast, the ANN and DNN models achieve runtimes of 0.05365 seconds and 0.08372 seconds, respectively, for pricing American options. The ANN runtime is 50.6% lower than the CRR method, and the DNN runtime is 22.9% lower than the CRR method. The ANN also outperforms the DNN model with a runtime that is 35.9% faster, demonstrating the computational advantage of shallow neural networks for early exercise complexities.

Stochastic Volatility (Heston Model)

The Heston stochastic volatility model introduces a further layer of complexity by modeling volatility as a stochastic process. For Heston American options, the ANN runtime is 0.12883 seconds, representing a 138% increase compared to the standard American ANN runtime of 0.05365 seconds. Similarly, the DNN runtime for Heston American options is 0.17038 seconds, a 103% increase compared to the standard American DNN runtime of 0.08372 seconds. Despite these increases, both ANN and DNN still perform faster than the LSMC method for Heston (0.19455 seconds), with ANN being 33.7% faster and DNN being 12.4% faster.

In summary, while traditional methods like BS and CRR provide reliable baselines, the neural network models exhibit significant advantages in runtime efficiency once trained. This efficiency combined with their ability to handle complex scenarios such as American options with early

[5]It is noted that the European options have a longer time than the American options for ANN and DNN of the Black-Scholes (BS) model compared to the CRR and LSMC model. This will be further investigated.

exercise and stochastic volatility models, emphasizes their possible potential as a powerful alternative to traditional option pricing methods.

Chapter 13

Conclusion

This dissertation successfully demonstrates the potential of neural networks as a viable alternative to traditional benchmark option pricing methods, namely the Black-Scholes model, the Cox-Ross-Rubinstein model and the Least-Squares Monte Carlo model. Through comprehensive investigation in our results, we were able to highlight the accuracy, robustness, and computational efficiency of artificial and deep neural networks in pricing both European and American options, under both stochastic and constant volatility. Overall, we can conclude that deep neural networks perform better than artificial neural networks as a viable alternative to option pricing using benchmark methods.

A key objective of the research was to evaluate whether the choice of input parameters significantly influences the performance of neural networks in option pricing. This was achieved by comparing the impact of scaler variations, such as MinMax and StandardScaler, on sensitivity graphs. Additionally, our dissertation examined the effects of incorporating varying dropout layers, demonstrating their influence on model performance. Insights gained from these analyses culminated in the development of an "ideal parameters" scenario, which integrated findings from prior research in this dissertation and sensitivity analyses to optimize neural network performance.

The disseration also explored the robustness of neural networks in dynamic market conditions by employing the Heston stochastic volatility model. This model provided a comprehensive framework for generating synthetic data, facilitating the evaluation of neural networks' ability to handle complex and realistic financial scenarios. By integrating the Heston stochastic volatility model with the Least-Squares Monte Carlo (LSMC) pricing method, we effectively assessed the networks' capacity to price options under stochastic volatility conditions. This integration allowed for a detailed examination of how neural networks perform when faced with highly dynamic market variables, highlighting their potential as reliable tools in financial modeling.

Moreover, the time efficiency of neural networks was rigorously compared to traditional benchmark methods, with the neural network models demonstrating superior running time metrics. These findings emphasize the computational advantages of neural networks, and this is particularly important in high-frequency trading and time-sensitive market environments.

While our model performs effectively within specific in-sample ranges, it is crucial to note that market conditions are rarely ideal, often necessitating a balance between time efficiency and predictive accuracy. In high-frequency trading environments where rapid decision-making is es-

sential, neural networks offer a distinct advantage. However, the extended training times can pose a significant limitation. As Bianco et al. (2018) highlight, there is no linear correlation between model complexity and accuracy. Additionally, not all deep neural network (DNN) parameters utilize their capacities with the same level of efficiency, emphasizing the need for careful optimization and parameter selection to achieve optimal performance.

By contrasting the performance of 'shallow' artificial neural networks (ANNs) with deep neural networks (DNNs), the dissertation sheds light on the architectural differences that influence model efficiency and accuracy. The results validate the hypothesis that deep learning architectures can serve as a viable alternative to traditional methods, such as the Black-Scholes (for European options) and CRR (for American options) models.

In the broader domain of deep learning, it is also important to note that with the integration of domain-specific parameters, including those derived from the Heston model, we were able to ensure that the neural network models were not treated as black-box systems. This interpretability and transparency aligns with the growing emphasis on explainable AI in financial modeling, bridging the gap between theoretical benchmarks and practical usability.

While the use of neural networks may perform very well in a controlled setting, we acknowledge that there is still work to be done in training these models to perform well in extreme and out-of-sample conditions. Areas for further research to improve the performance of neural networks when dealing with extreme market conditions is to sample the underlying using log-normal distribution instead of a uniform distribution. Financial asset prices often follow a log-normal distribution as noted by Pu (2021), and this distribution more accurately depicts real-world price movements, particularly at extreme values. The neural networks will learn to recognize and respond to extreme market conditions if it is trained with this distribution. This will provide us with more robust neural network models. We can also increase the range of data we train over, and with this possibly set up more advanced computational systems to handle the wider bounds. A higher computational power will allow us to test more complicated neural network structures or run it for more iterations.

In summary, our dissertation contributes to the field of quantitative finance by investigating the use of neural networks as a credible and efficient alternative to traditional option pricing methods. It achieves its objectives by presenting a detailed comparison of methodologies, optimizing neural network architectures, and addressing market complexities. These advancements pave the way for further research into enhancing neural network models for broader applications. All things considered, this dissertation advances us substantially toward attaining the pivotal objective of securing that critical additional minute.

Chapter 14

Challenges and Areas for Further Research

In this chapter we will look at challenges within the area of machine learning, drawing specific attention to neural networks, and our model. We will also propose possible areas for improvement and areas to be further researched.

In the area of further research in neural networks for option pricing, we focus on addressing challenges such as improving generalization to unseen market conditions, handling data scarcity, and incorporating domain knowledge to combine financial theory with machine learning. Enhancing interpretability through explainable AI (XAI) methods as highlighted by Rudin (2019), increasing robustness to noisy inputs, and optimizing architectures for real-time pricing are also important areas for consideration. Additional efforts are required for the handling exotic and path-dependent options using sequential models like LSTMs or transformers and to integrate features for more accurate pricing. These improvements aim to make neural network-based pricing models more accurate, efficient, and practical for real-world financial applications. A few of these areas will be explored in detail below.

To improve the architecture of neural networks for option pricing, it involves the optimizing their design with the intention of enhancing accuracy, efficiency, and robustness. This can be done by implementing the following alternatives:

- **Recurrent Neural Networks (RNNs):** Utilize RNNs, LSTMs, or GRUs for handling path-dependent options or sequential data, such as options with time-series features. This is discussed below, based on the work by Zouaoui & Naas (2023).
- **Transformer Models:** Leverage transformer-based architectures to model complex temporal relationships and dependencies, particularly useful for pricing multi-asset options. This is explored in detail by Guo & Tian (2022).
- **Convolutional Neural Networks (CNNs):** Employ CNNs for grid-based input data (e.g., volatility surfaces) to efficiently capture spatial patterns. This is explored in detail by Höfler (2024).

14.1 LSTM-GRU for Option Pricing

An interesting extension of the research in the deep learning domain is LSTM (long short-term memory) neural networks. In the paper by Zouaoui & Naas (2023) they presented a review of literature on option pricing using deep learning approach based on (long short-term memory-gated recurrent unit) LSTM-GRU models. This was done using data from the London Stock Exchange.

Recurrent neural networks (RNNs) differ from traditional neural networks by incorporating a transition weight that will allow the information flow over time. This transition weight makes each state dependent on the previous state, which introduces 'memory' to the modek. In RNNs, the hidden layers act as internal storage, retaining information from earlier stages. The term 'recurrent' is used based on the model's ability to apply the same operation to every element in a sequence, using historical information to predict future prices.

This process is depicted by the Figure 14.1 below:

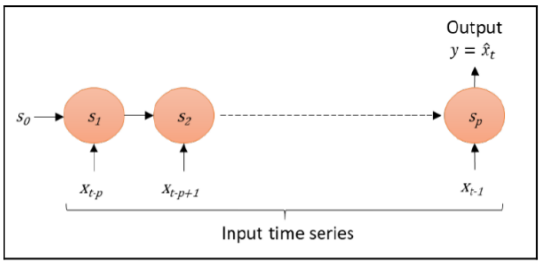


Figure 14.1: RNN with p time steps

The powerful RNN models LSTM and GRU are efficient for time dependent time-series forecasting, such as option pricing. This approach may be better suited to applications in finance, specifically option pricing, as LSTM is able to better capture the sequential dependencies in financial data, which standard ANNs and DNNs may struggle with.

Another advantage is that the LSTM model is more effective in dealing with the non-stationary and the volatile nature of financial markets, which is a limitation in standard ANN/DNN models. The advantage of memory with LSTM models is that when there's a significant market event the LSTM can retain this information for a longer time in its cell state. This allows the model to remember the effect of that event across many subsequent time steps, and thus better capturing long-term impacts on asset prices and their volatility. In contrast, conventional ANNs and DNNs lack this memory mechanism. Data is processed more statically, and the memory property is lacking.

14.2 Regularization Techniques

In this section we explore additional regularization techniques that extend beyond those previously discussed. These methods are specifically designed for the enhancement of model performance by addressing overfitting and improving generalization. By incorporating these advanced

approaches, we aim to equip the model with greater robustness and adaptability when applied to unseen data.

14.2.1 Early Stopping

Early stopping is a regularization technique discussed extensively by Brownlee (2019b) and Lindholm et al. (2019) in machine learning specifically designed to terminate the training process once the model's performance on a validation dataset begins to deteriorate, thus effectively mitigating overfitting. During the training phase, models are evaluated on both the training and validation datasets. While performance initially improves on both, a point may be reached where the model begins to overfit the training data, resulting in a decline in validation performance. By closely monitoring validation metrics and halting the training process at the earliest sighting of this decline, early stopping enhances the model's ability to generalize effectively to unseen data.

It essentially involves the tracking of a specific performance metric, such as validation loss or accuracy, during each training iteration or epoch. Training is halted when the chosen metric fails to improve over a predefined number of iterations, referred to as the "patience" parameter. This ensures that the model retains the parameters associated with its best validation performance. Early stopping proves particularly beneficial in cases where extended training risks overfitting, causing the model to capture noise and anomalies rather than the underlying patterns. By terminating training at the optimal moment, early stopping strikes a delicate balance between underfitting and overfitting, thereby enhancing the model's ability to generalize effectively to unseen data.

Prechelt (2002) highlights the susceptibility of neural networks, especially multilayer perceptrons (MLPs), to overfitting. He states:

> *While the network seems to get better and better, i.e., the error on the training set decreases, at some point during training it actually begins to get worse again, i.e., the error on unseen examples increases.*

This is particularly relevant, especially with respect to our experience with overfitting in this dissertation.

The following graph by illustrates the benefits of applying early stopping (Gençay & Qi, 2001).

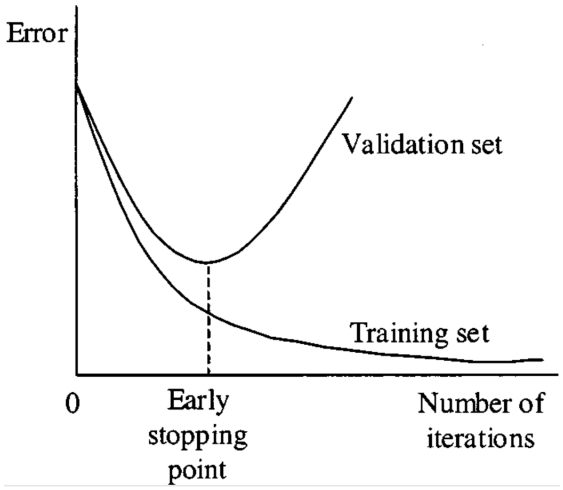


Figure 14.2: Early Stopping

This graph above illustrates the concept of early stopping in training machine learning models. The x-axis represents the number of training iterations (or epochs), while the y-axis shows the error (or loss). The training set curve consistently decreases as the model learns from the training data, which demonstrates the improved fitting to the training data over time. In comparison, the validation set curve initially decreases as the model's performance improves but eventually starts to increase, an indication of overfitting (the model begins memorizing training data and loses its generalization ability to unseen data). The early stopping point, indicated in the graph where the validation error reaches its minimum, is the optimal point to stop training to prevent overfitting. By stopping training at this point, the model strikes the balance between learning patterns in the data and maintaining its generalization capability for new, unseen data. Whilst we did not consider early stopping, it would be beneficial to neural network application to investigate this method further and test its efficiency.

14.2.2 Batch Normalization

Batch normalization is a technique introduced by Ioffe & Szegedy (2015) with the intention of addressing the issue of internal covariate shift in deep neural networks. Internal covariate shift refers to changes in the input distribution of a layer during training. This can hinder the learning process and complicate the training of deep networks. Batch normalization addresses this by ensuring the inputs to each layer maintain a consistent distribution, thereby improving the stability and accelerating the training process.

It normalizes the inputs of each layer within a mini-batch to have a mean of zero and a variance of one. This normalization is subsquently followed by scaling procedures that allow the network to maintain its capacity to represent complex functions. By the reduction of internal covariate shift, batch normalization enables the use of higher learning rates and alleviates issues related to poor initialization. We also note that it has a regularizing effect, often reducing the need for dropout.

This represents an additional technique that opens avenues for further exploration and research. It offers the potential to uncover new insights and refine existing methodologies thus contributing to the development of innovative solutions in the field of deep learning. With further investigation

regarding its applications as well as its implications, we can expand the understanding of its effectiveness and limitations.

14.2.3 L2 Parameter Regularization

L2 regularization, also known as ridge regularization or weight decay (Goodfellow et al., 2016), is a widely used technique in machine learning to prevent overfitting and improve model generalization. It works by adding a penalty term to the loss function that discourages large parameter values, thereby encouraging the model to keep its weights small and stable. The objective function in L2 regularization is modified as follows drawing upon the work by Karim (2018):

$$L(\theta) = L_0(\theta) + \lambda \sum_{i=1}^{n} \theta_i^2$$

- $L_0(\theta)$: Original loss function
- θ_i: Model parameters (weights).
- λ: Regularization coefficient (controls the strength of regularization).
- $\sum_{i=1}^{n} \theta_i^2$: Sum of the squares of the model parameters.

This penalty term, $\lambda \sum_{i=1}^{n} \theta_i^2$, is added to the original loss, ensuring the model prioritizes smaller weights during optimization.

The penalty term discourages the model from assigning high importance to any individual parameter. This can prevent overfitting, where the model memorizes noise in the training data by using overly large weights. During training, L2 regularization modifies the gradient updates for the parameters. The weights shrink slightly after each update, as smaller weights result in a smoother and more stable model that generalizes better to unseen data.

L2 regularization is a popular technique for reducing overfitting and enhancing model generalization by penalizing large weights. It ensures smoother decision boundaries and makes the model less sensitive to small input changes, improving its robustness to unseen data. Widely applicable to various models, including linear regression, logistic regression, and neural networks, L2 regularization is a cornerstone of machine learning. However, it is important to note that it introduces some limitations, such as the potential for underfitting due to bias toward smaller weights and the lack of sparsity in the resulting model, which can make interpretation more challenging. Despite these drawbacks, when properly tuned, L2 regularization significantly improves model stability and predictive performance.

In summary, while neural networks have shown great potential in option pricing, there remain numerous avenues for further research and improvement. Enhancing generalization, interpretability, and robustness, alongside leveraging advanced architectures and integrating domain knowledge, can significantly advance the field. By addressing these challenges, researchers can develop models that are not only accurate and efficient but also practical and adaptable to dynamic financial markets.

Reference List

[1] Acemoglu, D., & Restrepo, P. (2020). The wrong kind of AI? Artificial intelligence and the future of labour demand. *Cambridge Journal of Regions, Economy and Society*, 13(1), 25-35.

[2] Aggarwal, C. C. (2018). *Neural Networks and Deep Learning*. Springer.

[3] Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 22(2), 207-216.

[4] Ain, S. (2023). What is Association Rule Learning. Retrieved from `https://medium.com/@ainsupriyofficial/what-is-association-rule-learning-a6ef399fdc01`

[5] Alves de Oliveira, T., Michel, S., Habib, E., Silva, A., Taranto, A. (2023). Virtual Screening Algorithms in Drug Discovery: A Review Focused on Machine and Deep Learning Methods. *Drugs and Drug Candidates*, 2, 311-334.

[6] Anderson, D., & Ulrych, U. (2023). Accelerated American option pricing with deep neural networks. *Quantitative Finance and Economics*, 207-228.

[7] Anwar, A., (2021). What are Ensemble methods in Machine Learning? [Online] Available at: `https://towardsdatascience.com/what-are-ensemble-methods-in-machine-learning-cac1d17ed349` [Accessed August 2024].

[8] Bahoo, S., Cucculelli, M., Goga, X., & Mondolo, J. (2024). Artificial intelligence in finance: A comprehensive review through bibliometric and content analysis. *SN Business Economics*, 4(23).

[9] Balan, R. V., & Pulakkazhy, S. (2013). Data Mining in Banking and its Applications - A Review. *Journal of Computer Science*, 9(10), 1252-1259.

[10] Ballestra, L. V. (2018). Fast and Accurate Calculation of American Option Prices. *Decisions in Economics and Finance*, 399-426.

[11] Barles, G., & Souganidis, P. (1991). Convergence of approximation schemes for fully non-linear second order equations. *Asymptotic Analysis*, 4(3), 271-283.

[12] Barone-Adesi, G., & Whaley, E. R. (1987). Efficient Analytic Approximation of American Option Values. *The Journal of Finance*, 42(2), 301-320.

[13] Baxter, M., & Rennie, A. (1996). *Financial Calculus: An Introduction to Derivative Pricing*. Cambridge University Press.

[14] Bengio, Y. (2012). Practical Recommendations for Gradient-Based Training of Deep Architectures. Retrieved from https://arxiv.org/pdf/1206.5533

[15] Bhoi, A. K., de Albuquerque, V. H. C., Srinivasu, P. N., & Marques, G. (Eds.). (2022). Cognitive and Soft Computing Techniques for the Analysis of Healthcare Data. Academic Press.

[16] Bianco, S., Cadene, R., Celona, L., & Napoletano, P. (2018). Benchmark Analysis of Representative Deep Neural Network Architectures. *IEEE Access*, 6, 64270-64277.

[17] Bishop, C. M. (1996). *Neural Networks for Pattern Recognition*. Oxford University Press.

[18] Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *The Journal of Political Economy*, 81(3), 637-654.

[19] Boyle, P., Broadie, M., & Glasserman, P. (1996). Monte Carlo methods for security pricing. *Journal of Economic Dynamics and Control*, 21(8-9), 1267-1321.

[20] Boyle, P. P. (1986). Option Valuation using a tree-jump process. *International Options Journal*, 3, 7-12.

[21] Brennan, M. J., & Schwartz, E. S. (1977). The Valuation of American Put Options. *The Journal of Finance*, 32(2), 449-462.

[22] Broadie, M., & Detemple, J. (1996). American Option Evaluation: New Bounds, Approximations, and a Comparison of Existing Methods. *Review of Financial Studies*, 9(4), 1211-1250.

[23] Brownlee, J. (2019). Rectified Linear Activation Function for Deep Learning Neural Networks. Retrieved from https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[24] Brownlee, J., 2019. A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks. [Online] Available at: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/ [Accessed August 2024].

[25] Brownlee, J. (2020). How to Use Data Scaling to Improve Deep Learning Model Stability and Performance. Retrieved from https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/

[26] Bushaev, V. (2018). Understanding RMSprop — faster neural network learning. Retrieved from https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a

[27] Chakradhar, S. S. (2024). The McCulloch and Pitts Model: The Birth of Artificial Neurons. Retrieved from https://medium.com/@shivasaichakradhar/the-mcculloch-and-pitts-model-the-birth-of-artificial-neurons-0b1ef1ca2650

[28] Chandra, A. L. (2024). McCulloch-Pitts Neuron — Mankind's First Mathematical Model of a Biological Neuron. Retrieved from https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

[29] Chen, Y., & Wan, J. W. (2021). Deep neural network framework based on backward stochastic differential equations for pricing and hedging American options in high dimensions. *Quantitative Finance*, 21(1), 45-67.

[30] Clément, E., Lamberton, D., & Protter, P. (2002). An analysis of a least squares regression method for American option pricing. *Finance and Stochastics*, 6(4), 449-471.

[31] Contreras, M., Llanquihuén, A., & Villena, M. (2015). On the Solution of the Multi-asset Black-Scholes model: Correlations, Eigenvalues and Geometry. Retrieved from `https://arxiv.org/abs/1510.02768`

[32] Cotto, D. (2019). Neural Networks for Option Pricing. Retrieved from `https://towardsdatascience.com/neural-networks-for-option-pricing-danielcotto-c24569ad0bb`

[33] Cox, J. C., Ingersoll, J. E., & Ross, S. A. (1985). A theory of the term structure of interest rates. *Econometrica*, 53, 385-407.

[34] Cox, J. C., Ross, S. A., & Rubinstein, M. (1979). Option Pricing: A Simplified Approach. *Journal of Financial Economics*, 7, 229-263.

[35] Creamer, G. (2012). Model calibration and automated trading agent for euro futures. *Quantitative Finance*, 12(4), 531-545.

[36] Creamer, G., & Freund, Y. (2010). Automated trading with boosting and expert weighting. *Quantitative Finance*, 10(4), 401-420.

[37] Culkin, R., & Das, S. R. (2017). Machine Learning in Finance: The Case of Deep Learning for Option Pricing. *Journal of Investment Management*, 15(4), 92-100.

[38] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4), 303-314.

[39] Dertat, A. (2017). Applied Deep Learning: Part 1—Artificial Neural Networks. Retrieved from `https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6`

[40] DeSieno, D., & Trippi, R. R. (1992). Trading Equity Index Futures with a Neural Network. *The Journal of Portfolio Management*, 19(1), 27-33.

[41] Dowling, B. M. (2020). Neural Networks as an Option Pricing Method. *The Student Economic Review*, 34, 221-230.

[42] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning. *Journal of Machine Learning Research*, 12, 2121-2159.

[43] Durrett, R. (2010). *Probability Theory and Examples* (4th ed.). Cambridge University Press.

[44] Dutta, A. (2019). Geeks for Geeks: Stacking in Machine Learning. [Online] Available at: `https://www.geeksforgeeks.org/stacking-in-machine-learning/` [Accessed August 2024].

[45] Fan, J., & Zhong, Y. (2020). A Selective Overview of Deep Learning.

[46] Frino, A., et al. (2017). An Empirical Analysis of Algorithmic Trading Around Earnings Announcements. *Pacific-Basin Finance Journal*, 45, 34-51.

[47] Gaspar, R. M., Lopes, S. D., & Sequeira, B. (2020). Neural Network Pricing of American Put Options. *Risks*, 8(3), 73.

[48] Gençay, R. & Qi, M., 2001. Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging. IEEE Transactions on Neural Networks, Volume 12, pp. 726 - 734.

[49] Glasserman, P. (2003). *Monte Carlo Methods in Financial Engineering*. Springer.

[50] Goodfellow I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[51] Guo, T. & Tian, B., (2022). The Study of Option Pricing Problems based on Transformer Model. In: 2022 International Conference on Information Science and Communications Technologies (ICISCT), pp. 1-5. IEEEE

[52] Hammersley, J. M., & Handscomb, D. C. (1965). Monte Carlo Methods. *Physics Today*, 18(2), 55-56.

[53] Han, J., Jentzen, A., & Weinan, E. (2018). Solving High-Dimensional Partial Differential Equations Using Deep Learning. *Proceedings of the National Academy of Sciences*, 8505-8510.

[54] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.

[55] Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. Pearson.

[56] Hebb, D. O. (1949). The First Stage of Perception: Growth of the Assembly. *The Organization of Behaviour*, 4, 60-78.

[57] Hendershott, T., Jones, C. M., & Menkveld, A. J. (2011). Does Algorithmic Trading Improve Liquidity? *The Journal of Finance*, 66(1), 1-33.

[58] Heston, S. L. (1993). A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *The Review of Financial Studies*, 6(2), 327-343.

[59] Hinton, G. (2012). Coursera: Neural Networks for Machine Learning Online Course (Lecture 6e). Retrieved from `https://www.coursera.org/learn/neural-networks/home/welcome`

[60] Hirsa, A., Karatas, T., & Oskoui, A. (2019). Supervised Deep Neural Networks (DNNs) for Pricing/Calibration of Vanilla/Exotic Options under Various Processes. *arXiv preprint*, arXiv:1902.05810.

[61] Höfler, P., (2024). Volatility Surfaces and Expected Option Returns. Available at SSRN.

[62] Hornik, K. (1991). Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks*, 4(2), 251-257.

[63] Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2(5), 359-366.

[64] Horvath, B., Muguruza, A., & Mehdi, T. (2019). Deep Learning Volatility. Available at SSRN: `https://ssrn.com/abstract=3322085`.

[65] Huang, L., et al. (2022). Automatic Surgery and Anesthesia Emergence Duration Prediction Using Artificial Neural Networks. *Journal of Healthcare Engineering*, 2022(1), 2921775.

[66] Hull, J. C. (2015). *Options, Futures, and Other Derivatives* (9th ed.). Edinburgh: Pearson Education.

[67] Hunter, R., (2024). *UC David Lecture Notes: Chapter 2: The Supremum and Infimum.* [Online] Available at: `https://www.math.ucdavis.edu/~hunter/m125b/ch2.pdf` [Accessed November 2024].

[68] Hutchinson J. M., Lo, A. W., & Poggio, T. (1994). A Nonparametric Approach to Pricing and Hedging Derivative Securities. *The Journal of Finance*, 49(3), 851-889.

[69] Hutchinson J. M., Lo, A. W., & Poggio, T. (1994). A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, 49(3), 851-889.

[70] Ibáñez, A., & Zapatero, F. (2004). Monte Carlo Valuation of American Options through Computation of the Optimal Exercise Frontier. *Journal of Financial and Quantitative Analysis*, 39(2), 253-275.

[71] IBM Data (2023). AI versus machine learning versus deep learning versus neural networks: What's the difference? Retrieved from `https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks`.

[72] IBM Data, (2024). What is ensemble learning?. [Online] Available at: `https://www.ibm.com/topics/ensemble-learning?` [Accessed August 2024].

[73] Ioffe, S. & Szegedy, C., (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning (ICML), Volume 37, pp. 448-456.

[74] Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data Clustering: A Review. *ACM Computing Surveys*, 31(3), 264-323.

[75] Jain, S. (2024). Choosing the Right Activation Function for Your Neural Network. Retrieved from `https://www.geeksforgeeks.org/choosing-the-right-activation-function-for-your-neural-network/` [Accessed September 2024].

[76] Jain, S. (2024). Choosing the Right Activation Function for Your Neural Network. Retrieved from `https://www.geeksforgeeks.org/choosing-the-right-activation-function-for-your-neural-network/` [Accessed 22 October 2024].

[77] James, G., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R.* Springer.

[78] Jang, H., & Lee, J. (2019). Generative Bayesian Neural Network Model for Risk-Neutral Pricing of American Index Options. *Quantitative Finance*, 19(4), 587-603.

[79] Jaiswal, S., (2024). Datacamp: Multilayer Perceptrons in Machine Learning: A Comprehensive Guide. [Online] Available at: `https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning?` [Accessed August 2024].

[80] Jankova, Z. (2018). Drawbacks and Limitations of Black-Scholes Model for Option Pricing. *Journal of Financial Studies & Research*, 2018, 1-7.

[81] Jiang, L., & Dai, M. (1999). Convergence of Binomial Tree Method for American Options, in Partial Differential Equations and Their Applications. In I. H. Choi & L. R. R. Eds. (Eds.), *Partial Differential Equations and Their Applications.* New Jersey: World Scientific, 106-118.

[82] Jiang, L., & Dai, M. (2004). Convergence of Binomial Tree Methods for European/American Path-Dependent Options. *SIAM Journal on Numerical Analysis*, 42(3), 1094-1109

[83] John (2020). Pricing Options by Monte Carlo Simulation with Python. Retrieved from `https://www.codearmo.com/blog/pricing-options-monte-carlo-simulation-python` [Accessed November 2024].

[84] Johnson, P. (2020). Notes for MATH60082: Financial Mathematics. Retrieved from `https://personalpages.manchester.ac.uk/staff/paul.johnson-2/resources/math60082/notes-math60082-5.pdf` [Accessed November 2024].

[85] Jolliffe, I. T. (2002). *Principal Component Analysis* (2nd ed.). Springer Series in Statistics.

[86] Joshi, M. S., & Paterson, J. M. (2013). *Introduction to Mathematical Portfolio Theory*. Cambridge: Cambridge University Press.

[87] Kanas, A. (2001). Neural Network Linear Forecasts for Stock Returns. *International Journal of Finance and Economics*, 6(3), 245-254.

[88] Karim, R., (2018). Medium: Intuitions on L1 and L2 Regularisation. [Online] Available at: `https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261` [Accessed July 2024].

[89] Kelejian, H. H., & Mukerji, P. (2016). Does High Frequency Algorithmic Trading Matter for Non-AT Investors? *Research in International Business and Finance*, 37, 78-92.

[90] Khanna, A., & Amin, K. (1994). Convergence of American Option Values from Discrete to Continuous-Time Financial Models. *Mathematical Finance*, 4, 289-304.

[91] Khatun, A., (2018). Let's know Supervised and Unsupervised in an easy way. [Online] Available at: `https://chatbotsmagazine.com/lets-know-supervised-and-unsupervised-in-an-easy-way-9168363e06ab` [Accessed July 2024].

[92] Kohn, R., Smith, M. M., & Chan, D. (2001). Nonparametric Regression Using Linear Combinations of Basis Functions. *Statistics and Computing*, 11, 313-322.

[93] Korn, R., & Korn, E. (2001). *Option Pricing and Portfolio Optimization: Modern Methods of Financial Mathematics*. Providence: Irwin Professional Publishing.

[94] Korn, R., Korn, E., & Kroisandt, G. (2010). *Monte Carlo Methods and Models in Finance and Insurance*. CRC Press.

[95] Kushner, H. J. (1977). *Probability Methods for Approximations in Stochastic Control and for Elliptic Equations*. New York: Academic Press.

[96] Kwok, Y.-K. (2008). *Mathematical Models of Financial Derivatives* (2nd ed.). Springer.

[97] Lamberton, D. (1995). Error Estimates for the Binomial Approximation of American Put Options, Université de Marne-la-Vallée. [Online].

[98] Lamberton, D., & Pagès, G. (1990). Sur l'approximation des réduites. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 26(2), 331–355.

[99] Le Santo, G. (2020). Engage 2025 Forward: AI and Data-Driven World. Retrieved from `https://www.orange-business.com/en/blogs/engage-2025-forward-ai-and-data-driven-world` [Accessed November 2024].

[100] Leisen, D. P. (1996). Pricing the American Put Option: A Detailed Convergence Analysis for Binomial Models. *Journal of Economic Dynamics and Control*, 20(8-9), 1353–1383.

[101] Leisen, D. P., & Reimar, M. (1996). Binomial Models for Option Valuation: Examining and Improving Convergence. *Applied Mathematical Finance*.

[102] Lesuisse, M. (2022). What Are the Advantages of Pricing American Options Using Artificial Neural Networks? HEC-Ecole de gestion de l'Université de Liège.

[103] Lindholm, A., Wahlström, N., Lindsten, F., & Schön, T. B. (2019). *Supervised Machine Learning: Lecture Notes for the Statistical Machine Learning Course*. Retrieved from `https://mwns.co/blog/wp-content/uploads/2020/01/Supervised-Machine-Learning.pdf` [Accessed August 2024].

[104] Litzenberger, R., Castura, J., & Gorelick, R. (2012). The Impacts of Automation and High-Frequency Trading on Market Quality. *Annual Review of Financial Economics*, 4(1), 59-98.

[105] Liu, C. (2022). 5 Concepts You Should Know About Gradient Descent and Cost Function. Retrieved from `https://www.kdnuggets.com/2020/05/5-concepts-gradient-descent-cost-function.html` [Accessed August 2024].

[106] Liu, S., Leitao, A., Borovykh, A., & Oosterlee, C. W. (2020). On Calibration Neural Networks for Extracting Implied Information from American Options. *arXiv preprint arXiv:2001.11786*.

[107] Liu, S., Oosterlee, C. W., & Bohte, S. M. (2019). Pricing Options and Computing Implied Volatilities Using Neural Networks. *Risks*, 7(1), 16.

[108] Li, W., & Liao, J. (2011). An Empirical Study on Credit Scoring Model for Credit Card by Using Data Mining Technology. In *Proceedings of the 7th International Conference on Computational Intelligence and Security*, 1279-1282.

[109] Longstaff, F. A., & Schwartz, E. S. (2001). Valuing American Options by Simulation: A Simple Least-Squares Approach. *The Review of Financial Studies*, 14, 113-147.

[110] Luenberger, D. G. (1998). *Investment Science*. Oxford University Press.

[111] Lu, Y., & Lu, J. (2020). A Universal Approximation Theorem of Deep Neural Networks for Expressing Probability Distributions.

[112] Lu, Z., Pu, H., Wang, F., & Wang, L. (2017). The Expressive Power of Neural Networks: A View from the Width. *NIPS*.

[113] Madhavan, M., Varun, K. M., & Vishnu, C. M. (2012). Segmenting the Banking Market Strategy by Clustering. *International Journal of Computer Applications*, 45(17), 10-15.

[114] Maini, V., & Sabri, S. (2017). *Machine Learning for Humans*. Retrieved from `https://www.everythingcomputerscience.com/books/Machine%20Learning%20for%20Humans.pdf`.

[115] McCulloch, W. S., & Pitts, W. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5, 115-133.

[116] McFarland, J., & DeCarlo, E. (2020). A Monte Carlo Framework for Probabilistic Analysis and Variance Decomposition with Distribution Parameter Uncertainty. *Reliability Engineering & System Safety*, 197(0951-8320).

[117] Merton, R. C. (1973). Theory of Rational Option Pricing. *The Bell Journal of Economics and Management Science*, 4(1), 141-183.

[118] Microsoft (2017). Powering the Future of the Customer Experience. Retrieved from `https://news.microsoft.com/europe/features/ai-powering-customer-experience/` [Accessed November 2024].

[119] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Education.

[120] Moreno, M., & Navas, J. F. (2003). On the Robustness of Least-Squares Monte Carlo (LSM) for Pricing American Derivatives. *Review of Derivatives Research*, 6, 107-128.

[121] Mostafa, B. M., El-Attar, N., Abd-Elhafeez, S., & Awad, W. A. (2020). Machine and Deep Learning Approaches in Genome: Review Article. *Alfarama Journal of Basic & Applied Sciences*, 2(1), 105-113.

[122] Moutzouris, V. (2021). Valuing American-Style Derivatives by Simulation: Alternative Regression-Based Methods, Masters Dissertation, University of Pretoria.

[123] Nesterov, Y. (2013). *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Science & Business Media.

[124] Ngai, E. W., Hu, Y., Wong, Y. H., Chen, Y., & Sun, X. (2011). The Application of Data Mining Techniques in Financial Fraud Detection: A Classification Framework and an Academic Review of Literature. *Decision Support Systems*, 50(3), 559-569.

[125] Øksendal, B. (2003). *Stochastic Differential Equations: An Introduction with Applications* (6th ed.). Springer.

[126] Olamendy, J. C. (2023). Mini-Batch Gradient Descent: Optimizing Machine Learning. Retrieved from `https://juancolamendy.hashnode.dev/mini-batch-gradient-descent-optimizing-machine-learning/` [Accessed October 2024].

[127] Ono, S., & Goto, T. (2022). Introduction to Supervised Machine Learning in Clinical Epidemiology. *Ann Clin Epidemiol*, 4(3), 63-71.

[128] Oppermann, A. (2022). Optimization in Deep Learning: AdaGrad, RMSProp, ADAM. [Online]. Retrieved from `https://artemoppermann.com/optimization-in-deep-learning-adagrad-rmsprop-adam/` [Accessed November 2024].

[129] Oussama, B. (2018). Pricing of American Options Using Neural Networks.

[130] Pakkanen, M. S., Wood, B., Murray, P., & Buehler, H. (2021). Deep Hedging: Learning Risk-Neutral Implied Volatility Dynamics. arXiv preprint arXiv:2103.11948.

[131] Partridge, R. (2023). Learning By Doing: A Detailed Overview of the Reinforcement Learning Process. [Online]. Retrieved from `https://medium.com/@achronus/learning-by-doing-a-detailed-overview-of-the-reinforcement-learning-process-05011883ad9c` [Accessed October 2024].

[132] Pelletier, H. (2024). Data Scaling 101: Standardization and Min-Max Scaling Explained. [Online]. Retrieved from `https://towardsdatascience.com/data-scaling-101-standardization-and-min-max-scaling-explained-60789833e160` [Accessed October 2024].

[133] Piepenbreier, N. (2023). Datagy: Relu Activation Function. [Online]. Retrieved from https://datagy.io/relu-activation-function/ [Accessed September 2024].

[134] Pramoditha, R. (2022). The Relationship between AI, ML, NN, and DL. [Online]. Retrieved from https://medium.com/data-science-365/the-relationship-between-ai-ml-nns-and-dl-60bd40069908.

[135] Prechelt, L., (2002). Early stopping—but when?. In Neural Networks: Tricks of the trade (pp. 55-69). Berlin, Heidelberg: Springer Berlin Heidelberg.

[136] Pu, V. R. H. (2021). Pricing Options Using Deep Neural Networks from a Practical Perspective: A Comparative Study of Supervised and Unsupervised Learning, Masters Dissertation, London: Imperial College London.

[137] Pykes, K. (2024). Introduction to Unsupervised Learning. [Online]. Retrieved from https://www.datacamp.com/blog/introduction-to-unsupervised-learning\#SnippetTab.

[138] Quinn, B. (2023). Cornell University. [Online]. Retrieved from https://arxiv.org/abs/2306.02773v1.

[139] Raschka, S., (2014). Predictive modeling, supervised machine learning, and pattern classification. [Online] Available at: https://sebastianraschka.com/Articles/2014_intro_supervised_learning.html [Accessed September 2024].

[140] Raschka, S. (2018). STAT 479: Machine Learning Lecture Notes. [Online]. Retrieved from https://sebastianraschka.com/pdf/lecture-notes/stat479fs18/01_ml-overview_notes.pdf.

[141] Reddy, K. M. (2018). Data Preprocessing Using Python Sklearn. [Online]. Retrieved from https://medium.com/@kesarimohan87/data-preprocessing-6c87d27156 [Accessed October 2024].

[142] Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6), 386-408.

[143] Ross, S. M. (1995). *Stochastic Processes*. Wiley & Sons.

[144] Rouah, F. D. (2016). Four Derivations of the Black-Scholes Formula. [Online]. Retrieved from https://mmquant.net/wp-content/uploads/2016/08/BlackScholesFormula.pdf [Accessed August 2024].

[145] Ruder, S. (2016). An Overview of Gradient Descent Optimization Algorithms. [Online]. Retrieved from https://arxiv.org/abs/1609.04747 [Accessed September 2024].

[146] Rudin, C. (2019). Stop Explaining Black Box Machine Learning Models for High-Stakes Decisions and Use Interpretable Models Instead. *Nature Machine Intelligence*, 1(5), 206-215.

[147] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Representations by Back-Propagating Errors. *Nature*, 323(6088), 533-536.

[148] Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson.

[149] Salvador, B., Oosterlee, C. W., & van der Meer, R. (2020). European and American Options Valuation by Unsupervised Learning with Artificial Neural Networks. *Multidisciplinary Digital Publishing Institute Proceedings*, 54(1), 14.

[150] Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3), 210-229.

[151] Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks*, 85-117.

[152] Schmidt, M., (2016). *Argmax and Max Calculus*. [Online] Available at: `https://www.cs.ubc.ca/~schmidtm/Documents/2016_540_Argmax.pdf` [Accessed November 2024].

[153] Shih, C. (2023). The Future of AI at Dreamforce 2023: Interview with Clara Shih. [Online]. Retrieved from `https://www.salesforce.com/plus/experience/Dreamforce\_2023/series/the\_future\_of\_ai\\\_at\_dreamforce\_2023/episode/episode-s1e6` [Accessed 2024].

[154] Shih, J.-Y. (2011). Using Self-Organizing Maps for Analyzing Credit Rating and Financial Data. In *Proceedings of the IEEE International Summer Conference of Asia Pacific Business Innovation and Technology Management*, 109-112.

[155] Shreve, S. E. (2004). *Stochastic Calculus for Finance 1*. Springer-Verlag.

[156] Soleymani, F., & Vasighi, M. (2020). Efficient Portfolio Construction by Means of CVaR and k-means++ Clustering Analysis: Evidence from the NYSE. *International Journal of Finance & Economics*, 27(3), 3679-3693.

[157] Soni, B., (2023). Understanding Boosting in Machine Learning: A Comprehensive Guide. [Online] Available at: `https://medium.com/@brijesh_soni/understanding-boosting-in-machine-learning-a-comprehensive-guide-bdeaa1167a6` [Accessed July 2024].

[158] Soraemon (2022). Optimizers: SGD with Momentum, NAG, Adagrad, RMSProp, AdaDelta, and ADAM. [Online]. Retrieved from `https://velog.io/@soraemon/Optimizers-SGD-with-Momentum-NAG-Adagrad-RMSProp-AdaDelta-and-ADAM`. [Accessed October 2024].

[159] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15, 1929–1958.

[160] Stinchcombe, M., Hornik, K. M., & White, H. (1989). Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In *Proceedings of the International Joint Conference on Neural Networks*, 613–618.

[161] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.

[162] Szőke, C. (2017). AI to drive GDP gains of $15.7 trillion with productivity, personalisation improvements. [Online]. Retrieved from `https://www.pwc.com/hu/en/pressroom/2017/ai.html`.

[163] Tang, S. (2015). *American-style Option Pricing and Improvement of Regression-based Monte Carlo Methods by Machine Learning Techniques*. Ph.D. Thesis, Technische Universit.

[164] Terven, J. R., Garcia-Garcia, J., Moya-Sanchez, E. U., & Morales-Hernandez, L. A. (2023). Loss Functions and Metrics in Deep Learning. 10.48550/arXiv.2307.02694.

[165] Tewari, U., (2021). Regularization — Understanding L1 and L2 regularization for Deep Learning. [Online] Available at: https://medium.com/analytics-vidhya/regularization-understanding-l1-and-l2-regularization-for-deep-learning-a7b9e4a409bf [Accessed May 2024].

[166] Thevenot, A. (2020). 12 Main Dropout Methods: Mathematical and Visual Explanation for DNNs, CNNs, and RNNs. [Online]. Retrieved from https://towardsdatascience.com/12-main-dropout-methods-mathematical-and-visual-explanation-58cdc2112293.

[167] Tian, X., & Benkrid, K. (2012). Implementation of the Longstaff and Schwartz American Option Pricing Model on FPGA. *Journal of Signal Processing Systems*, 67, 79–91.

[168] Tilley, J. A. (1993). Valuing American Options in a Path Simulation Model. *Transactions of Society of Actuaries*, 45, 499–549.

[169] van der Meer, R., Oosterlee, C., & Borovykh, A. (2020). Optimally weighted loss functions for solving PDEs with neural networks. *arXiv preprint*, arXiv:2002.06269.

[170] Vyacheslav, E. (2023). Understanding Deep Learning Optimizers: Momentum, AdaGrad, RMSProp & Adam. [Online]. Retrieved from https://towardsdatascience.com/understanding-deep-learning-optimizers-momentum-adagrad-rmsprop-adam-e311e377e9c2.

[171] Wang, R. J. (2022). NTU (National Taiwan University) Financial Computation Notes: Ch4 Binomial Tree Model. [Online]. Retrieved from https://homepage.ntu.edu.tw/~jryanwang/courses/Financial%20Computation%20or%20Financial%20Engineering%20(graduate%20level)/FE_Ch04_Binomial_Tree_Model.pdf.

[172] Wang, S., & Cao, J. (2021). AI and Deep Learning for Urban Computing. *Urban Informatics*, 815–844.

[173] Wang, X., et al. (2023). Dynamic User Resource Allocation for Downlink Multicarrier NOMA with an Actor–Critic Method. *Energies*, 16(7), 2984.

[174] Wei, W., & Zhu, D. (2022). Generic improvements to least squares Monte Carlo methods with applications to optimal stopping problems. *European Journal of Operational Research*, 298(3), 1132–1144.

[175] Weron, Rafa, and Uwe Wystup. "7 Heston's Model and the Smile." Statistical tools for finance and insurance (2005): 161.

[176] Wrigglesworth, A. (2021). Consumer Credit Default Prediction Using Machine Learning: An Application of Deep Learning, Masters Dissertation, University of Pretoria.

[177] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. [Online]. Retrieved from https://arxiv.org/pdf/1212.5701.

[178] Zhang, Y., Chu, G., & Shen, D. (2021). The role of investor attention in predicting stock prices: The long short-term memory networks perspective. *Finance Research Letters*, 38, 101484.

[179] Zouaoui, H., & Naas, M.-N. (2023). Option Pricing Using Deep Learning Approach Based on LSTM-GRU Neural Networks: Case of London Stock Exchange. *Data Science in Finance and Economics*, 3(3), 267–284.

[180] Zvan, R., Forsyth, P. A., & Vetzal, K. R. (1998). Penalty Methods for American Options with Stochastic Volatility. *Journal of Computational and Applied Mathematics*, 91(2), 199-218.

Chapter 15

Appendix

15.1 Appendix 1: General Important Definitions

Theorem 1 - (Durrett, 2010)
Let $X_1, X_2, ...$ be independent and identically distributed with $\mathbb{E}[X_i]$=μ, $VAR(X_i) = \sigma^2$ an element in interval (0,∞).

If $\bar{X}_n$=$\frac{1}{n}\sum_{i=1}^{n} X_i$ then $\bar{X}_n \to \mu$ in L^2.

Theorem 2 - (Durrett, 2010)
Let $X_1, X_2, ...$ be independent and identically distributed with $\mathbb{E}[X_i] = \mu$, $VAR(X_i) = \sigma^2$ an element in interval (0,∞).
If $\bar{X}_n$=$\frac{1}{n}\sum_{i=1}^{n} X_i$ then

$$\sqrt{n}\frac{\bar{X}_n - \mu}{\sigma}$$

converges in distribution to a Gaussian stochastic variable with (0,1)

Theorem 3: Ito's Lemma - (Yoo, 2017)
Suppose f is a C^2 function and B_t is a standard Brownian motion.
Then for every t,

$$f(B_t) = f(B_0) + \int_0^t f'(B_s)dB_s + \frac{1}{2}\int_0^t f''(B_s)ds$$

The formula can also be written in differential form as

$$df(B_t) = f'(B_t)dB_t + \frac{1}{2}f''(B_t)dt.$$

Proposition 1 - (Bergstrom, 2014)
Assume that V is enough differentiable and

$$V(t,x) = \sup_{t\leq\tau\leq T} \mathbb{E}_{t,x}[e^{-r(\tau-t)}\Phi(\tau, X_\tau)]$$

Then for the region $C := \{(t,x); V(t,x) > \Phi(t,x)\}$ the following holds:

$$V(T,x) = \Phi(T,x)$$
$$\mathcal{L}V(t,x) = rV(t,x) \text{ for all } (t,x) \in C$$
$$\mathcal{L}V(t,x) < rV(t,x) \text{ for all } (t,x) \notin C$$

where

$$\mathcal{L}= \frac{\partial}{\partial t} + rx\frac{\partial}{\partial x} + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2}{\partial x^2}.$$

The optimal stopping time standing at (t, x) is

$$\hat{\tau} = inf\{s \geq t; V(s, X_s) = \Phi(s, X_s)\}$$

15.2 Appendix 2: Complete Binomial Tree Framework and Derivation

In the following section we draw upon the work by (Wang, 2022).
The method is based on the assumption of risk-neutral valuation, as illustrated in the figure below where

S_0 is the value of the underlying stock.

K is the strike price

$C_u = max[0, S_0u - K]$ is the value of the call option in the upward state

$C_d = max[0, S_0d - K]$ is the value of the call option in the downward state

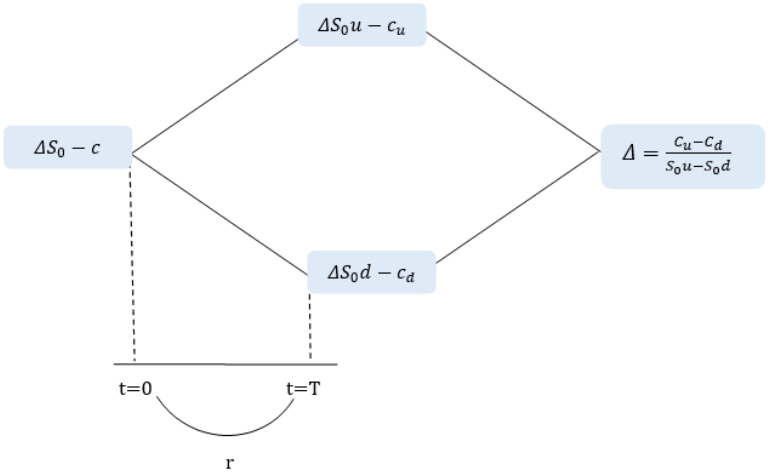


Figure 15.1: One-Period binomial tree model

We consider the Δ-hedge portfolio that writes a single call, buys Δ units of the underlying. The value of Δ is chosen such that the portfolio is risk-free:

$$\Delta S_0 d - c_d = \Delta S_0 u - c_u$$

$$\Delta = \frac{c_u - c_d}{S_0 u - S_0 d}.$$

With our risk-free interest rate r, we solve for c (Hull, 2015):

$$\Delta = \frac{c_u - c_d}{S_0 u - S_0 d}$$

$$\frac{c_u - c_d}{S_0 u - S_0 d} S_0 - c = \left(S_0 u \frac{c_u - c_d}{S_0 u - S_0 d} - c_u \right) e^{-rT}$$

$$c = \frac{c_u - c_d}{u - d} - \left(\frac{c_u - c_d}{u - d} u - c_u \right) e^{-rT}$$

$$c = e^{-rT} \left(\frac{(c_u - c_d)e^{rT}}{u - d} - \frac{(c_u - c_d)u}{u - d} + \frac{(u - d)c_u}{u - d} \right)$$

$$c = e^{-rT} \left(\frac{(c_u - c_d)e^{rT} - (c_u - c_d)u + uc_u - dc_u + c_d u - c_d u}{u - d} \right)$$

$$= e^{-rT} \left(\frac{e^{rT} - d}{u - d} c_u + \left(u - \frac{e^{rT} - d}{u - d} \right) c_d \right)$$

$$= e^{-rT} \left(\frac{e^{rT} - d}{u - d} c_u + \left(1 - \frac{e^{rT} - d}{u - d} \right) c_d \right)$$

We denote p as $\left(\frac{e^{rT} - d}{u - d} \right)$ and $1 - p$ as $1 - \left(\frac{e^{rT} - d}{u - d} \right)$.

$$= e^{-rT} \left(p c_u + (1 - p) c_d \right)$$

with p and $1 - p$ our risk-neutral probabilities in the binomial tree framework.

CRR Binomial Tree We will be using work from Hull (2015), Cox, et al. (1979) and Wang (2022) for the derivations in this section.

Due to the assumption of no-arbitrage, the following holds:

$$u > e^{r\Delta t} > d. \tag{15.1}$$

We now calculate the expected value and variance of the underlying at time $t + \Delta t$ with subscript B referring to 'binomial' as:

$$\mathbb{E}_{\mathrm{B}}(S_{t+\Delta t}) = \sum x f(x) = p(uS_t) + (1 - p)(dS_t) \tag{15.2}$$

$$\begin{aligned} \mathrm{Var}_{\mathrm{B}}(S_{t+\Delta t}) &= \mathbb{E}_{\mathrm{B}}^2(S_{t+\Delta t}) - \left(\mathbb{E}_{\mathrm{B}}(S_{t+\Delta t}) \right)^2 \\ &= \sum x^2 f(x) - \left(\sum x f(x) \right)^2 \\ &= p(S_t u)^2 + q(S_t d)^2 - \left(p(S_t u) + q(S_t d) \right)^2 . \end{aligned}$$

Risk-Neutral Probabilities in the binomial Tree Framework

It is important to note that p is not the actual probability for c_u (Or of S moving 'upward'). However, in the **risk-neutral world** p can be considered as the probability for c_u. This is because in the real world, if the expected stock return is μ,

$$S_0 e^{\mu T} = S_0 u \cdot q + S_0 d(1-q)$$

$$q = \frac{e^{rT} - d}{u - d}$$

With risk-neutrality, our expected returns for all securities are identical and equal to r, we have:

$$S_0 e^{rT} = S_0 u \cdot p + S_0 d(1-p)$$

$$p = \frac{e^{rT} - d}{u - d}$$

Therefore, p and $1-p$ are considered the risk-neutral probabilities in the binomial tree framework.

The pricing formula

$$c = e^{-rT}\left(pc_u + (1-p)c_d\right)$$

in the binomial tree model aligns with Risk-Neutral Valuation Relationship (RNVR) as both the expected growth rate of the underlying and the discount rate applied to the option payoff are the risk-free rate.

While we use the probabilities (q and $1-q$ in the real world, practically it is quite challenging to identify a proper discount rate to apply to the options expected payoff function.

Noting that the discount rates for expected option payoffs depend on multiple factors: the expected returns (μ), volatilities (σ) of underlying and the different K and T of options. Consequently, its quite challenging to compute theoretical option prices in practice.

Lognormal Property

If X is lognormally distributed then $\ln X$ is normally distributed:

Mean$= E[\ln S_T] \implies E[S_T] = e^{E[\ln S_T] + \frac{1}{2}\text{var}(\ln S_T)}$

Variance $= \text{var}(\ln S_T) \implies \text{var}(S_T) = e^{2E[\ln S_T] + \text{var}(\ln S_T)} \cdot (\text{var}(\ln S_T) - 1)$.

$$\ln S_T \sim N(\ln S_0 + (\mu - \frac{\sigma^2}{2})T, \sigma^2 T)$$

$$E(S_T) = S_0 e^{\mu T}$$

and

$$\text{var}(S_T) = S_0 e^{2\mu T}(e^{\sigma^2 T} - 1).$$

Therefore, under the risk-neutral measure $\mathbb{Q}$, we have:

$$\begin{aligned}
\mathbb{E}^{\mathbb{Q}}(S_{t+\Delta t}) &= S_t e^{\mu \Delta t}, \\
\text{var}^{\mathbb{Q}}(S_{t+\Delta t}) &= S_t^2 e^{2\mu\Delta t}\left(e^{\sigma^2 \Delta t} - 1\right) \\
&\approx S_t^2(1 + 2\mu\Delta t)\left(1 + \sigma^2\Delta t - 1\right) \\
&= S_t^2\sigma^2\Delta t + S_t^2(2\mu\Delta t)(\sigma^2\Delta t) \\
&\approx S_t^2\sigma^2\Delta t.
\end{aligned}$$

We use this to derive the parameters u, d, and p.

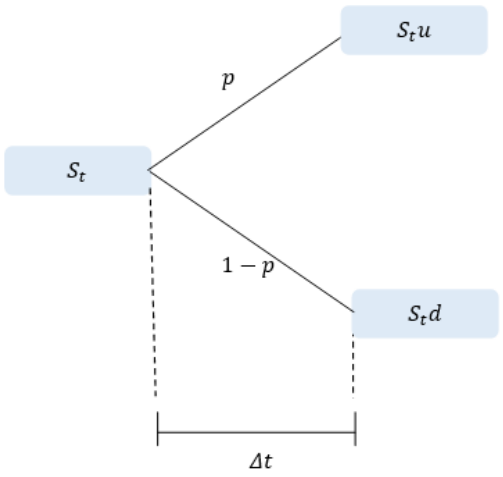


Figure 15.2: One-step binomial model

By setting the moments from the binomial tree equal to the population moments we obtain (Moutzouris, 2021):

$$\begin{aligned}
\mathbb{E}^{\mathbb{Q}}(S_{t+\Delta t}) &= \mathbb{E}_B(S_{t+\Delta t}) \\
S_t e^{r\Delta t} &= p \cdot S_u + (1-p) \cdot S_d \\
\therefore p &= \frac{e^{r\Delta t} - d}{u - d}.
\end{aligned}$$

Similarly, for the variance (Wang, 2022):

$$\begin{aligned}
\text{var}^{\mathbb{Q}}(S_{t+\Delta t}) &= \text{var}_B(S_{t+\Delta t}) \\
\text{var}^{\mathbb{Q}}(S_{t+\Delta t}) &= \mathbb{E}_B[(S_{t+\Delta t})^2] - \mathbb{E}_B[S_{t+\Delta t}]^2.
\end{aligned}$$

Note that

$$\begin{aligned}
\sigma^2\Delta t &= e^{r\Delta t}(u+d) - u\cdot d - e^{2r\Delta t} \\
\sigma^2\Delta t &= e^{r\Delta t}\left(u+\frac{1}{u}\right) - u\cdot\frac{1}{u} - e^{2r\Delta t} \quad \text{(by defining } d=\frac{1}{u}\text{)} \\
u+\frac{1}{u} &= \frac{\sigma^2\Delta t + 1 + e^{2r\Delta t}}{e^{r\Delta t}} \\
&= e^{-r\Delta t}\sigma^2\Delta t + e^{-r\Delta t} + e^{r\Delta t} \\
&\approx \sigma^2\Delta t + 2
\end{aligned}$$

$$\begin{aligned}
u^2 - (\sigma^2\Delta t + 2)u + 1 &= 0 \\
u &= \frac{\sigma^2\Delta t + 2 \pm \sqrt{(\sigma^2\Delta t+2)^2 - 4}}{2} \\
&= \frac{\sigma^2\Delta t + 2 \pm \sqrt{\sigma^4\Delta t^2 + 4\sigma^2\Delta t + 4 - 4}}{2} \quad \text{(since } \sigma^4\Delta t^2 \to 0\text{)} \\
&\approx \frac{\sigma^2\Delta t}{2} + 1 \pm \sigma\sqrt{\Delta t} \\
&\approx 1 \pm \sigma\sqrt{\Delta t} \\
&\approx e^{\pm\sigma\sqrt{\Delta t}} \quad .
\end{aligned}$$

The approximations for u and d are obtained by ignoring terms in Δt^2 and higher powers of Δt to get:

$$\begin{aligned}
u &= e^{\sigma\sqrt{\Delta t}} \\
d &= e^{-\sigma\sqrt{\Delta t}}
\end{aligned}$$

15.3 Appendix 3: Othonormal Basis Functions

Chebyshev 1^{st} kind - (Mason & Handscomb, 2003)
The recursive formula depicting the analytical solution of the sequence:

$$CH_1(n,x) = xCH_1(n-1,x) - CH_1(n-2,x)$$

where

$$CH_1(0,x) = 1 \text{ and } CH_1(1,x) = \text{x}.$$

With the assumption that $CH_1(n,x)$ is representative of the n^{th} degree Chebyshev 1st kind polynomial of x.

Chebyshev 2^{nd} kind - (Mason & Handscomb, 2003)
The recursive formula depicting the analytical solution of the sequence:

$$CH_2(n,x) = 2xCH_2(n-1,x) - CH_2(n-2,x)$$

where

$$CH_2(0,x) = 1 \text{ and } CH_2(1,x) = 2\text{x}.$$

With the assumption that $CH_2(n,x)$ is representative of the n^{th} degree Chebyshev 2nd Kind A polynomial of x.

Hermite Polynomial - (Moreno & Navas, 2003)
The recursive formula depicting the analytical solution of the sequence:

$$H(n,x) = 2xH(n-1,x) - 2(n-1)H(n-2,x)$$

where

$$H(0,x) = 1 \text{ and } H(1,x) = 2x.$$

With the assumption that $H(n,x)$ is representative of the n^{th} degree Hermite polynomial of x.

Laguerre Polynomial
The recursive formula depicting the analytical solution of the sequence:

$$La(n,x) = \frac{(2n-1-x)La(n-1,x)-(n-1)La(n-2,x)}{n}$$

where

$$La(0,x) = 1 \text{ and } La(1,x) = 1\text{-}x.$$

With the assumption that $La(n,x)$ is representative of the n^{th} degree Laguerre polynomial of x.

Legendre Polynomial - (Moreno & Navas, 2003)
The recursive formula depicting the analytical solution of the sequence:

$$Le(n,x) = \frac{2n-1}{n}xLe(n-1,x) - \frac{n-1}{n}Le(n-2,x)$$

where

$$Le(0,x) = 1 \text{ and } Le(1,x) = x.$$

With the assumption that $Le(n,x)$ is representative of the n^{th} degree Legendre polynomial of x.

Powers Polynomial - (Moreno & Navas, 2003)
The recursive formula depicting the analytical solution of the sequence:

$$P(n,x) = xP(n-1,x)$$

where

$$P(0,x) = 1 \text{ and } P(1,x) = x.$$

With the assumption that $P(n,x)$ is representative of the n^{th} degree Powers polynomial of x.

Trigonometric Polynomial - (Rudin, 1987)
The recursive formula depicting the analytical solution of the sequence:

$$T(r,x) = a_0 + \sum_{r=1}^{R} a_r \cos(rx) + i\sum_{r=1}^{R} b_r \sin(rx)$$
$$\text{for } (x, a_r, b_r \in \mathbb{R})$$

where

$$0 \leq r \leq R \text{ and } a_r, b_r \in \mathbb{C}$$

With the assumption that $T(n,x)$ is representative of the n^{th} degree Trigonometric polynomial of x.

15.4 Appendix 3: Tables

A table of different activation functions obtained from (Pu, 2021) and (Pakkanen, Wood, Murray, & Buehler, 2021) is shown below. These activation functions include:

Activation	Plot	Definition	Derivative	Range
Identity (Id)		$g(x) = x$	$g'(x) = 1$	$\mathbb{R}$
Heaviside (H)		$g(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$	$g'(x) = 0, x \neq 0$	{0, 1}
Sigmoid (σ)		$g(x) = \frac{1}{1+e^{-x}}$	$g'(x) = g(x)(1-g(x))$	(0, 1)
Hyperbolic Tangent (tanh)		$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$g'(x) = 1 - g(x)^2$	(-1, 1)
ArcTan		$g(x) = \tan^{-1}(x)$	$g'(x) = \frac{1}{1+x^2}$	(-1.5, 1.5)
Rectified linear unit (ReLU)		$g(x) = \max(x, 0)$	$g'(x) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$	$[0, \infty)$
Softplus		$g(x) = \log(1 + e^x)$	$g'(x) = \frac{1}{1+e^{-x}}$	$(0, \infty)$

Figure 15.3: Activation functions

The following table shows different loss functions obtained from (Pu, 2021) and (Pakkanen, Wood, Murray, & Buehler, 2021).

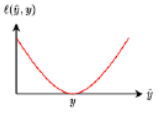
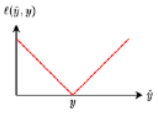
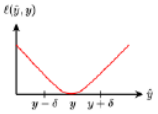
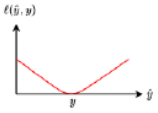
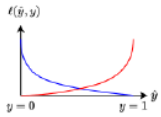
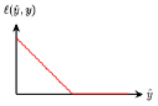
Loss	Plot	Definition	Use
Squared loss		$\ell(\hat{y}, y) = (\hat{y} - y)^2, \quad \hat{y}, y \in \mathbb{R}$	Regression
Absolute loss		$\ell(\hat{y}, y) = \lvert\hat{y} - y\rvert, \quad \hat{y}, y \in \mathbb{R}$	Regression
Huber loss		$\ell(\hat{y}, y) = \begin{cases} \frac{1}{2}(\hat{y} - y)^2, & \lvert\hat{y} - y\rvert \le \delta \\ \delta(\lvert\hat{y} - y\rvert - \frac{1}{2}\delta), & \lvert\hat{y} - y\rvert > \delta \end{cases}$	Regression
Log-cosh loss		$\ell(\hat{y}, y) = \log(\cosh(\hat{y} - y)) \quad \hat{y}, y \in \mathbb{R}$	Regression
Binary cross-entropy		$\ell(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log 1 - \hat{y}, \quad \hat{y} \in (0, 1), y \in \{0, 1\}$	Binary classification
Hinge loss		$\ell(\hat{y}, y) = \max(0, 1 - \hat{y}y), \quad \hat{y} \in (0, 1), y \in \{0, 1\}$	Binary classification

Figure 15.4: Loss functions

15.5 Appendix 4: Results for ANN and DNN fit without a scaler

Set 1: CRR vs ANN DNN (American)

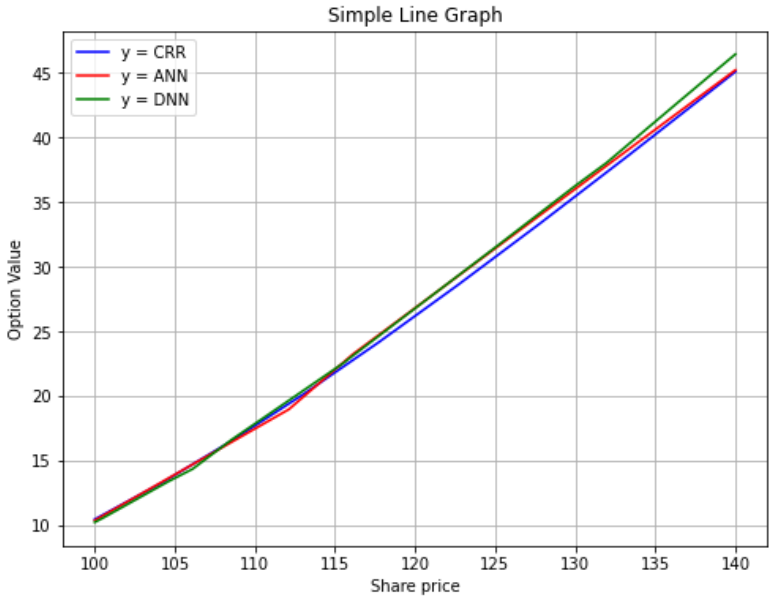


Figure 15.5: Share price sensitivity

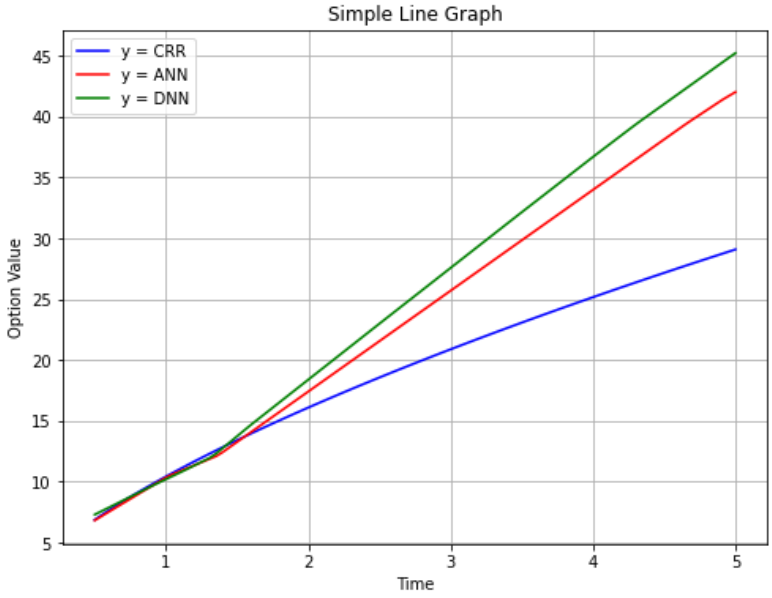


Figure 15.6: Time to maturity

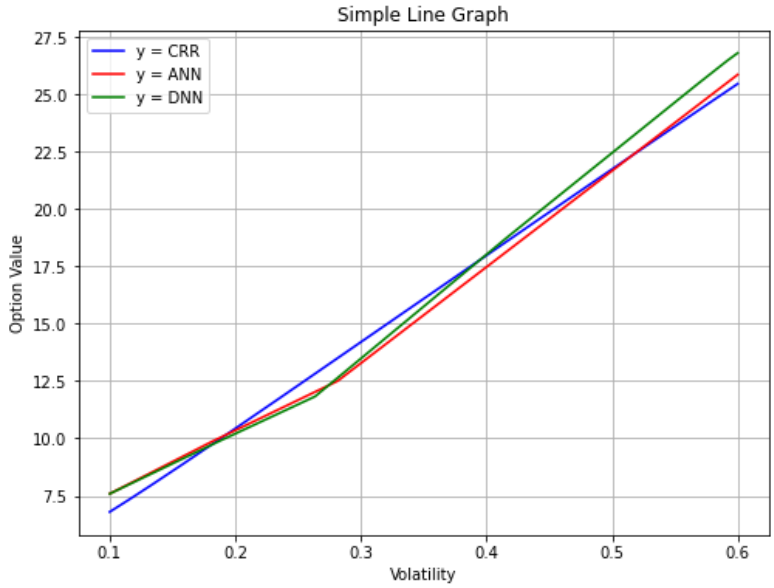


Figure 15.7: Volatility sensitivity

Set 2:LSM vs ANN DNN (American)

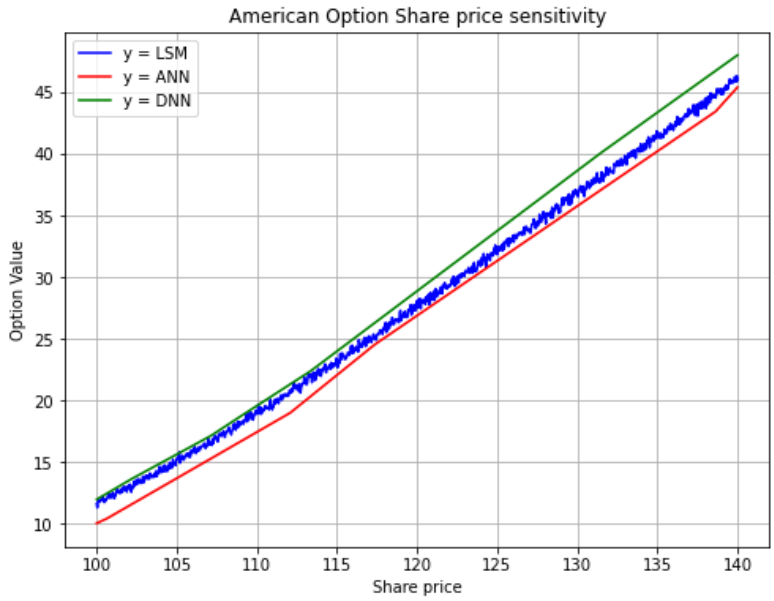


Figure 15.8: Share price sensitivity

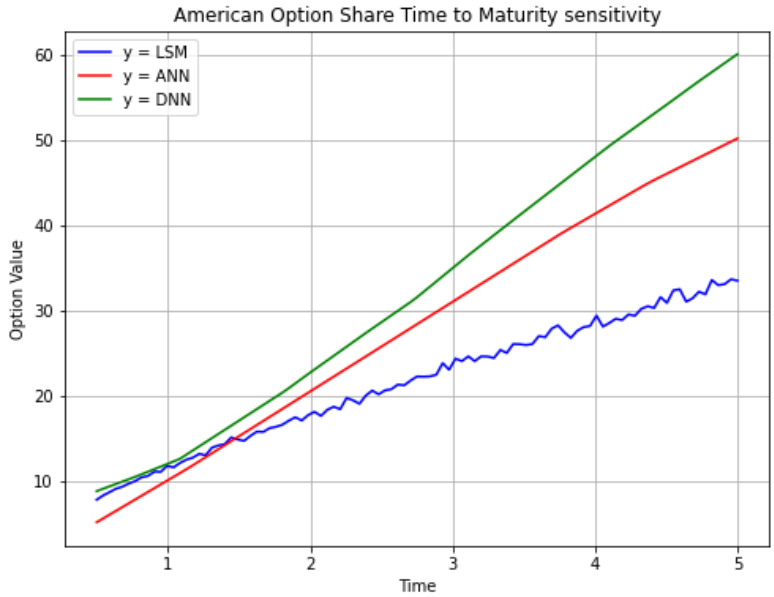


Figure 15.9: Time to maturity

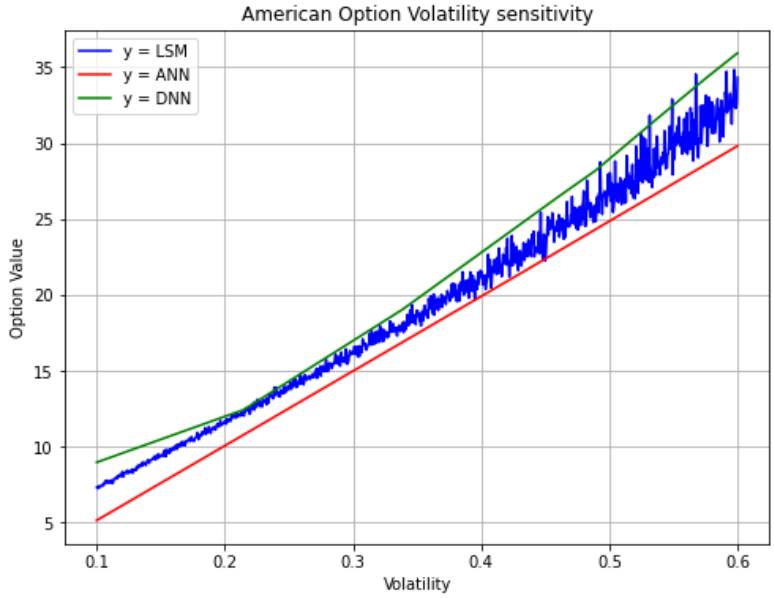


Figure 15.10: Volatility sensitivity

15.6 Appendix 5: Heston Dropout Performance

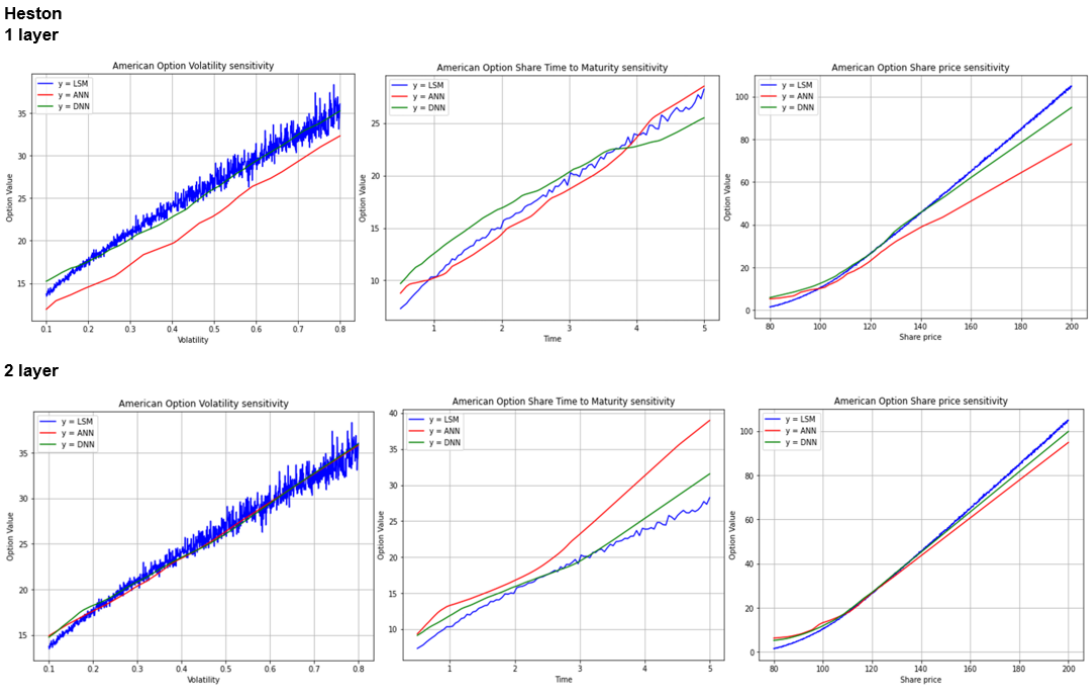


Figure 15.11: Heston LSMC option dropout 1&2 layers

15.7 Appendix 6: Code snippet for Ideal Parameter Scenario

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.layers import LeakyReLU, Dropout
import time
start_time = time.time()


# Define the CRR binomial tree model
def CRR_american_option_price(S, K, r, sigma, T, N):
dt = T / N
u = np.exp(sigma * np.sqrt(dt))
d = 1 / u
p = (np.exp(r * dt) - d) / (u - d)
```

```
# Initialize stock price tree
stock_tree = np.zeros((N + 1, N + 1))
for i in range(N + 1):
for j in range(i + 1):
stock_tree[j, i] = S * (u ** (i - j)) * (d ** j)

# Initialize option value tree
option_tree = np.zeros((N + 1, N + 1))
# Calculate option values at maturity
option_tree[:, N] = np.maximum(stock_tree[:, N] - K, 0)

# Calculate option values at earlier nodes
for i in range(N - 1, -1, -1):
for j in range(i + 1):
exercise_value = stock_tree[j, i] - K
hold_value = np.exp(-r * dt) * (p * option_tree[j, i + 1] + (1 - p) *
option_tree[j + 1, i + 1])
option_tree[j, i] = np.maximum(exercise_value, hold_value)

return option_tree[0, 0]

# Generate training data
def generate_training_data_CRR(num_samples, N):
np.random.seed(42)
S = np.random.uniform(80, 120, num_samples)  # Initial stock price
K = np.random.uniform(80, 120, num_samples)  # Strike price
r = np.random.uniform(0.01, 0.1, num_samples)  # Risk-free rate
sigma = np.random.uniform(0.1, 0.5, num_samples)  # Volatility
T = np.random.uniform(0.25, 2, num_samples)  # Time to maturity
call_prices = np.zeros(num_samples)

for i in range(num_samples):
call_prices[i] = CRR_american_option_price(S[i], K[i], r[i], sigma[i],
 T[i], N)

X = np.column_stack((S, K, r, sigma, T))
return X, call_prices

# Define and train the neural network DNN
def train_neural_network_DNN(X_train, y_train):
model = tf.keras.Sequential([
tf.keras.layers.Dense(64, input_shape=(5,)),
LeakyReLU(alpha = 0.01),
tf.keras.layers.Dense(64),
LeakyReLU(alpha = 0.01),
tf.keras.layers.Dropout(0.3),
tf.keras.layers.Dense(64),
LeakyReLU(alpha = 0.01),
tf.keras.layers.Dropout(0.3),
```

```
tf.keras.layers.Dense(1, activation='linear')
])
model.compile(optimizer='adam', loss='mse')
history = model.fit(X_train, y_train, validation_split=0.2,
 epochs=100, verbose=0)
return model, history

# Define and train the neural network ANN
def train_neural_network_ANN(X_train, y_train):
model = tf.keras.Sequential([
tf.keras.layers.Dense(32, input_shape=(5,)),
LeakyReLU(alpha = 0.01),
tf.keras.layers.Dense(32),
LeakyReLU(alpha = 0.01),
tf.keras.layers.Dense(1, activation='linear')
])
model.compile(optimizer='adam', loss='mse')
history = model.fit(X_train, y_train, validation_split=0.2,
 epochs=100, verbose=0)
return model, history

# Parameters
S = 100     # Initial stock price
K = 100     # Strike price
r = 0.05    # Risk-free rate
sigma = 0.4 # Volatility
T = 2       # Time to maturity in years
N = 50     # Number of time steps in CRR
num_samples = 10000  # Number of samples for training

# Generate training data for CRR
X_train_CRR, y_train_CRR = generate_training_data_CRR(num_samples, N)

# Scale features
scaler = StandardScaler()#StandardScaler()
X_train_CRR_scaled = scaler.fit_transform(X_train_CRR)

pre_train = time.time()
# Train the neural network DNN for CRR
model_DNN_CRR, history_DNN_CRR = train_neural_network_DNN
(X_train_CRR_scaled, y_train_CRR)
model_train_time_DNN = time.time()


# Train the neural network ANN for CRR
model_ANN_CRR, history_ANN_CRR = train_neural_network_ANN
(X_train_CRR_scaled, y_train_CRR)
model_train_time_ANN = time.time()
```

```
#Time to run
time_pre_results = time.time()
CRR_american_option_price(S, K, r, sigma, T, N)
LSM_time_result = time.time()
input_data_ = scaler.transform(np.array([[100, 100, 0.05, 0.2, 1]]))
model_ANN_CRR.predict(input_data_)
ANN_time_result = time.time()
model_DNN_CRR.predict(input_data_)
DNN_time_result = time.time()


train_time_DNN = model_train_time_DNN - pre_train
train_time_ANN = model_train_time_ANN - model_train_time_DNN
LSM_runtime = LSM_time_result - time_pre_results
ANN_runtime = ANN_time_result - LSM_time_result
DNN_runtime = DNN_time_result - ANN_time_result



# Make predictions for ITM, ATM, and OTM scenarios
scenarios = {
        "ITM": (110, K, r, sigma, T),   # In-the-money
        "ATM": (S, K, r, sigma, T),     # At-the-money
        "OTM": (80, K, r, sigma, T)     # Out-of-the-money
}

for scenario, (S, K, r, sigma, T) in scenarios.items():
input_data = scaler.transform(np.array([[S, K, r, sigma, T]]))
prediction_DNN_CRR = model_DNN_CRR.predict(input_data)
prediction_ANN_CRR = model_ANN_CRR.predict(input_data)
AOP_CRR = CRR_american_option_price(S, K, r, sigma, T, N)

print(f"\n{scenario} Scenario:")
print("CRR American Call Option Price:", AOP_CRR)
print("DNN Predicted Call Option Price (CRR):", prediction_DNN_CRR[0][0])
print("ANN Predicted Call Option Price (CRR):", prediction_ANN_CRR[0][0])


#=============================================================================
# Graphing the above
#=============================================================================

lower_share = 80
upper_share = 200
increments = 1001

# First check sensitivity to underlying price
share_price = np.zeros(increments)
interval = (upper_share - lower_share)/(increments-1)
```

```
for i in range(0, increments):
share_price[i] = lower_share + i*interval

# Create predictions
CRR = np.zeros(increments)
ANN = np.zeros(increments)
DNN = np.zeros(increments)

for i in range(0, increments):
input_data_share = scaler.transform
(np.array([[share_price[i], 100, 0.05, 0.2, 1]]))
CRR[i] = CRR_american_option_price(share_price[i], 100, 0.05, 0.2, 1, N)
ANN[i] = model_ANN_CRR.predict(input_data_share)
DNN[i] = model_DNN_CRR.predict(input_data_share)

# plot graph
plt.figure(figsize=(8, 6))
plt.plot(share_price, CRR,  linestyle='-', color='b', label='y=-CRR')
plt.plot(share_price, ANN,  linestyle='-', color='r', label='y=-ANN')
plt.plot(share_price, DNN,  linestyle='-', color='g', label='y=-DNN')
plt.title('American-Option-Share-price-sensitivity')
plt.xlabel('Share-price')
plt.ylabel('Option-Value')
plt.legend()
plt.grid(True)
plt.show()
```