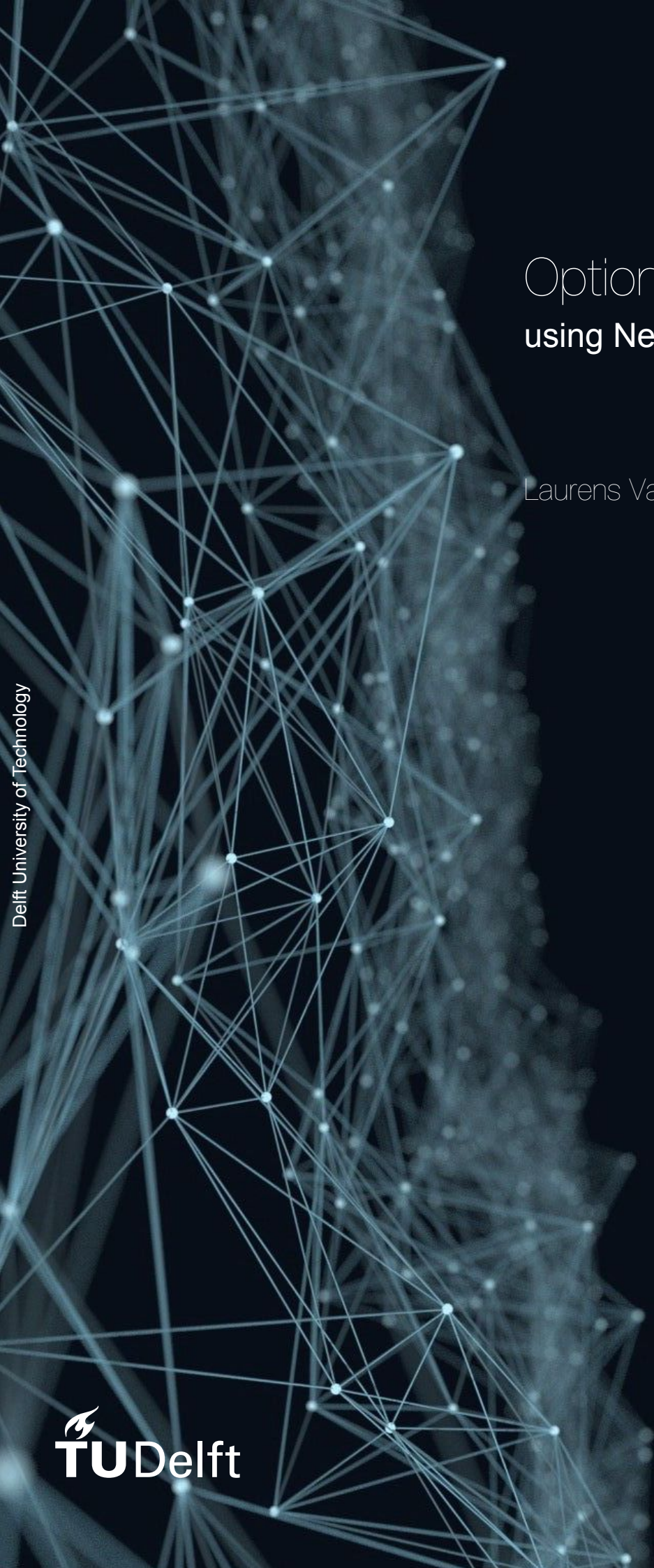

Option Pricing Techniques

using Neural Networks

Laurens Van Mieghem

Delft University of Technology

TU Delft

Option Pricing Techniques

using Neural Networks

by

Laurens Van Mieghem

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday July 27, 2022 at 12:30 AM.

Student number: 4680669
Project duration: November 1, 2021 – July 27, 2022
Thesis committee: Prof. Dr. A.P. Papapantoleon, TU Delft, supervisor
Dr. F. Fang TU Delft

TU Delft

Abstract

With the emergence of more complex option pricing models, the demand for fast and accurate numerical pricing techniques is increasing. Due to a growing amount of accessible computational power, neural networks have become a feasible numerical method for approximating solutions to these pricing models. This work concentrates on analysing various neural network architectures on option pricing optimisation problems in a supervised and semi-supervised learning setting. We compare the mean-squared error (MSE) and computational training time of a multilayer perceptron (MLP), highway architecture and a recently developed DGM network [20] along with slight variations on the Black-Scholes and Heston European call option pricing problem as well as the implied volatility problem. We find that on nearly all the supervised learning problems, the generalised highway architecture outperforms its counterparts in terms of MSE relative to computation time. On the Black-Scholes problem, we noticed a reduction of 9.8% in MSE for the generalised highway network while containing 96.2% fewer parameters compared to the MLP considered in [25].

On the semi-supervised learning problem, where we directly optimise the neural network to fit the partial differential equation (PDE) and boundary/initial conditions, we concluded that the network architecture of the DGM allows for optimisation of both the interior condition as well as the non-smooth terminal condition. As this was not the case for the MLP and highway networks, the DGM network turned out to be the best performing network architecture on the semi-supervised learning problems. Additionally, we found indications that on the semi-supervised learning problem the performance of the DGM network remained consistent when increasing the dimensionality of the problem.

Acknowledgements

First and foremost I would like to express my gratitude for the supervision and guidance of Antonis Papapantoleon, who was always available to express his thoughts and provide valuable feedback whenever I deemed it necessary.

I would also like to thank Dennis Palagin for providing access to DelftBlue's supercomputer, as the simulations would have not been possible without this tremendous amount of computing power.

I would also like to thank Fang Fang for being a part of my thesis committee, and Heiko Schäfer for unknowingly causing a great spark in interest in the discipline of option pricing.

Finally, I would like to show appreciation to my family and friends, who provided me with the endless support and perseverance throughout all the years of study that ultimately lead to the research and writing of this thesis.

Thank you.

Laurens Van Mieghem
Delft, July 2022

Contents

1 Introduction

A substantial discipline in mathematical finance focuses on modelling the derivatives market. Options in particular have gained widespread interest ever since the pioneering Black-Scholes partial differential equation [15] was invented in 1973. Present day option pricing models are generally more complex than the Black-Scholes model, where solutions are often not known in closed-form. To this end, accurate numerical solvers are required, such as Fourier based techniques [10, Chapter 6] or Monte-Carlo simulation [10, Chapter 9]. The emergence of the Internet over past decades caused trading to shift from physical trading floors to online exchanges, resulting in an increase in automated trading systems. As a result of these systems, speed is becoming a prominent requirement when pricing these derivatives, especially in the field of high frequency trading (HFT) [18]. There is a considerable demand in finding accurate and fast pricers, with the ability to approximate solutions to complex financial models. A recently emerging area of interest is the application of neural networks as a numerical approximation technique.

Neural networks originally date back to 1943, when McCulloch and Pitts created a computational model resembling a neural network [37], paving the way for later research. A major problem with these networks was the lack of computational power required for back propagation through the network, causing the development to stagnate. The rise of computational power from the 1980's allowed for practical development of neural networks, and in 2010 Schmidhuber et al. [11] demonstrated the feasibility of back propagation on GPUs, solving the computational bottleneck with neural networks in general. In the following years, research in this discipline vastly accelerated, and variants of neural networks have since then proven to be very effective in various fields, including image recognition [2] and natural language processing (NLP) [40].

The successes of neural networks in these fields naturally raise questions of their effectiveness in finance, with in particular the approximation of option pricing models. After a computationally intensive calibration stage of the neural network which we refer to as training, the evaluation speed of the neural network is comparable to analytical functions, as primarily matrix-vector multiplications must be computed. This property makes neural networks extremely attractive as an option pricing technique, provided that they can be trained to output consistent and accurate approximations during inference.

In 2019, Oosterlee et al. conducted a study on the effectiveness of neural networks as a numerical method on various problems [25]. In this work, a regular neural network configuration is chosen and applied to a Black-Scholes and Heston option pricing problem as well as the implied volatility problem. The problems are organised in a supervised way, in which a dataset is constructed with input variables and the numerical solution as a target variable. The aim of the neural network is to approximate the target variable using the input variables. Chapter 4 of this thesis is concerned with extending this analysis by developing and training various network architectures on the problems in an identical setting. One of these network architectures originate from Sirignano et al. [20], in which a neural network is developed to approximate the solution to partial differential equations (PDEs). Albeit in a different setting than proposed by the authors, we evaluate this network architecture alongside two variations to quantify the effectiveness of each part of the network.

In addition to the study of network architectures on the supervised learning problem, we adopt a concept originating from Sirignano et al. [20] and continued on by Wunderlich and Glau in [22], where

a PDE is transformed into an optimisation problem which we can approximate using neural networks. This procedure, referred to as the *DGM method*, allows us to directly optimise the PDE, instead of compiling a dataset with an approximated solution from a reference pricing technique. Additionally, the mesh-free algorithm of the DGM method allows for generalisation to higher dimensions, where other techniques suffer from the curse of dimensionality. Chapter 5 concentrates on the so-called DGM method, and analyses the performance of various network architectures on multidimensional pricing models. The aim is to quantify the effectiveness of this optimisation method, and provide an analysis on the capabilities of various network architectures on such problems. As an extension to the DGM method, we conduct a study on the parametrisation of the PDE. By treating the parameters of the PDE as variables, a single neural network can approximate the solution to an entire family of PDEs with only one training phase. The concept of parametrisation originates from [22], and will be extended in this thesis with the analysis of multiple network architectures.

2

Stochastic Calculus Fundamentals

In this chapter, we briefly review some of the basics in stochastic calculus. This chapter closely follows [33, Chapter 3, 4]. We touch upon the relationship between probability densities and characteristic functions. Subsequently, we define Brownian motion, along with a few useful properties. From this definition, we construct the Itô integral, which will serve as a fundamental tool to define Itô's formula. Using this formula, we can take derivatives of functions of Brownian motions, which we will use to derive the Black-Scholes PDE in section 3.2 and the Heston PDE in section 3.4.

2.1. Density & Characteristic Function

When considering a random variable X, we often describe it in terms of its distribution. We define the cumulative distribution function (CDF) as

$$F_X(x) := \mathbb{P}(X \leq x), \tag{2.1}$$

along with its probability density function (PDF)

$$f_X(x) = \frac{dF_X(x)}{dx}. \tag{2.2}$$

Using the PDF, we can calculate moments such as the expectation

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x) dx \tag{2.3}$$

and the variance

$$Var[X] = \int_{-\infty}^{\infty} (x - \mathbb{E}[x])^2 f_X(x) dx, \tag{2.4}$$

provided that both integrals exist and are finite. In option pricing, we are generally concerned with calculating the expectation of a stochastic variables, such as a function of a stock price. Therefore, it is particularly useful to have an analytical solution for the probability density function. Unfortunately, not all relevant stochastic processes have a density that is known. For a collection of such processes, we can derive an alternative expression to a probability density function called a characteristic function. Effectively, the characteristic function is the Fourier transform of the probability density function

$$\phi_X(u) = \mathbb{E}\left[e^{iuX}\right] = \int_{-\infty}^{\infty} e^{iux} f_X(x) dx. \tag{2.5}$$

We can use the characteristic function to determine the moments of the random variable X, completely determining the distribution of X, as

$$\mathbb{E}[X_k] = \frac{1}{i^k} \frac{d^k}{du^k} \phi_X(u)|_{u=0}. \tag{2.6}$$

In this thesis, we are interested in the characteristic function of random variables for another reason, namely expansion of the option price formulas as Fourier series. In section 3.4.3, we construct a Fourier based pricing function for the 2-dimensional Heston model, for which we require the characteristic function.

2.2. Martingales

A martingale is a sequence of random variables with certain properties. Perhaps the most elementary example of a martingale is that of a random walk

$$X_n = \sum_{i=0}^{n} Y_i, \tag{2.7}$$

where Y_i are i.i.d. random variables with $\mathbb{E}[Y_i] = 0$. The random process X_n exhibits no future trend, i.e.

$$\mathbb{E}(X_{n+1} - X_n | X_1, \dots, X_n) = 0. \tag{2.8}$$

The defining property of a martingale is (2.8). In order to define martingales rigorously, we introduce the concept of a filtration.

Definition 1. *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. We call a sequence of σ-algebra's $\{\mathcal{F}_n, n \in \mathbb{N}\}$ a filtration if*

- *It is an increasing sequence, $\mathcal{F}_0 \subset \mathcal{F}_1 \subset \cdots \subset \mathcal{F}_n$ ...*
- *$\mathcal{F}_n \subset \mathcal{F}, \forall n \in \mathbb{N}$*

1. *We call a process $\{X_n, n \in \mathbb{N}\}$ adapted to the filtration $\{\mathcal{F}_n, n \in \mathbb{N}\}$ if for all $n \in \mathbb{N}$, X_n is a random variable which is $\mathcal{F}_n$ measurable.*
2. *We call a process $\{X_n, n \in \mathbb{N}\}$ predictable with respect to the filtration $\{\mathcal{F}_n, n \in \mathbb{N}\}$ if for all $n \in \mathbb{N}$, X_n is a random variable which is $\mathcal{F}_{n-1}$ measurable.*

Definition 2. *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space equipped with filtration $\{\mathcal{F}_n, n \in \mathbb{N}\}$. Let $\{X_n, n \in \mathbb{N}\}$ denote an adapted sequence of real valued random variables. The sequence is called a martingale if*

1. *For all $n \in \mathbb{N}$, X_n is integrable.*
2. *$\{X_n, n \in \mathbb{N}\}$ satisfies the martingale property, being*

$$\mathbb{E}(X_{n+1} | \mathcal{F}_n) = X_n, \forall n \in \mathbb{N}. \tag{2.9}$$

In addition to the definition of a martingale, we list a few computational properties of martingales, but omit the proof for brevity.

Theorem 1. *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space equipped with a filtration $\{\mathcal{F}_n, n \in \mathbb{N}\}$ be given.*

(i) If $\{X_n, n \in \mathbb{N}\}$ is a martingale, then its expectation is constant, i.e. $\mathbb{E}[X_n] = \mathbb{E}[X_0]$ for all $n \in \mathbb{N}$.

(ii) Let $\{X_n, n \in \mathbb{N}\}$ and $\{\tilde{X_n}, n \in \mathbb{N}\}$ both be martingales. For $a, b \in \mathbb{R}$, we have that

$$aX_n + b\tilde{X}_n, n \in \mathbb{N} \tag{2.10}$$

is also a martingale.

We have defined a stochastic process $\{X_n, n \in \mathbb{N}\}$ in discrete time. Many models, however, require a continuous time stochastic process. Therefore, we consider a scaled symmetric random walk, by fixing a positive integer n, and defining

$$W^{(n)}(t) = \frac{1}{\sqrt{n}} X_{nt}, \tag{2.11}$$

given that nt is an integer. If not, we interpolate between the values of $\lfloor nt \rfloor$ and $\lceil nt \rceil$. If we take the limit $n \to \infty$, we obtain a Brownian motion, which is a continuous time stochastic process with interesting properties (section 2.3).

Theorem 2. *Fix $t \geq 0$. As $n \to \infty$, the distribution of the scaled random walk $W^{(n)}(t)$ evaluated at time t converges to the normal distribution with mean 0 and variance t.*

In the next section, we formally define Brownian motion, and list some of its properties.

2.3. Brownian Motion

Brownian motion is a real valued stochastic process that is a fundamental building block for the Itô integral, defined in section 2.4, and therefore also for stochastic differential equations.

Definition 3. *A real valued process $\{W(t) : t \geq 0\}$ is called a Brownian motion if*

1. *$W(0) = 0$.*
2. *For all $0 \leq s < t, W(t) - W(s) \sim \mathcal{N}(0, t-s)$.*
3. *For $0 \leq t_0 < t_1 < \cdots < t_n$, the random variables $Y_i := W(t_i) - W(t_{i-1}), i = 1, \ldots, n$ are independent.*
4. *The map $t \mapsto W(t)$ is continuous.*

In order to derive further properties of Brownian motion, we introduce the concept of a filtration

Definition 4. *Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space on which is defined a Brownian motion $\{W(t), t \geq 0\}$. A filtration for the Brownian motion is a collection of σ-algebras $\{\mathcal{F}(t), t \geq 0\}$, such that*

(i) For $0 \leq s < t$, every set in $\mathcal{F}(s)$ is also in $\mathcal{F}(t)$.

(ii) For each $t \geq 0$, the Brownian motion $W(t)$ at time t is $\mathcal{F}(t)$-measurable.

(iii) For $0 \leq t < u$, the increment $W(u) - W(t)$ is independent of $\mathcal{F}(t)$. That is, any increment of the Brownian motion after time t is independent of the information contained in $\mathcal{F}(t)$.

A filtration attempts to denote the amount of information that is available at time t. Properties *(i)* and *(ii)* guarantee that all the information available at each time t is at least as much as one would learn from observing the Brownian motion up to time t. Property *(iii)* tells us that the information contained in $\mathcal{F}(t)$ does not give us any information on future movements of the Brownian motion $\{W(u), u > t\}$.

Definition 5. *We say that a process $\{M(t), t \geq 0\}$ is a $\{\mathcal{F}(t), t \geq 0\}$ martingale if*

1. *Adapted: $M(t)$ is $\mathcal{F}(t)$ measurable for all $t \geq 0$.*
2. *Integrable: $M(t)$ is integrable for all $t \geq 0$.*
3. *Martingale property: For all $0 \leq s < t$:*

$$\mathbb{E}[M(t)|\mathcal{F}(s)] = M(s). \tag{2.12}$$

We can verify that the Brownian motion $W(t)$ is a martingale, by checking that the process is integrable and the martingale property holds. Indeed, for $t < \infty$

$$\mathbb{E}[|W(t)|] = \frac{1}{\sqrt{2\pi t}} \int_{-\infty}^{\infty} |x| e^{-\frac{x^2}{2t}} dx \tag{2.13}$$

$$= \frac{2}{\sqrt{2\pi t}} \int_{0}^{\infty} x e^{-\frac{x^2}{2t}} dx \tag{2.14}$$

$$= \frac{2t}{\sqrt{2\pi t}} \int_{0}^{\infty} e^{-z} dz \tag{2.15}$$

$$= \frac{2t}{\sqrt{2\pi t}} < \infty, \tag{2.16}$$

where we used the substitution $z = \frac{x^2}{2t}$. For the martingale property we have

$$\mathbb{E}[W(t)|\mathcal{F}(s)] = \mathbb{E}[W(s) + (W(t) - W(s))|\mathcal{F}(s)] \tag{2.17}$$
$$= \mathbb{E}[W(s)|\mathcal{F}(s)] + \mathbb{E}[W(t) - W(s)|\mathcal{F}(s)] \tag{2.18}$$
$$= W(s), \tag{2.19}$$

for all $s, t > 0$.

2.3.1. Quadratic Variation

We define the quadratic variation for a process as the sum of squared differences between the process over consecutive time steps. More formally,

Definition 6. *Let $f(t)$ be a function defined on $[0,T]$. Let $\Pi = \{t_0, t_1, \dots, t_n\}$ be a partition of $[0,T]$, and $\|\Pi\|$ be the mesh of the partition. The quadratic variation of f up to time T is*

$$[f,f](T) = \lim_{\|\Pi\|\to 0} \sum_{j=0}^{n-1} \Big[f(t_{j+1}) - f(t_j)\Big]^2. \tag{2.20}$$

We need the notion of quadratic variation in stochastic calculus to account for the non-differentiability in stochastic processes. In ordinary calculus, functions with continuous derivatives have 0 quadratic variation. When defining Itô's formula used for taking stochastic derivatives (section 2.4.2), we must add an additional term to the derivative, accounting for the quadratic variation of the process.

2.4. Itô's Integral

Suppose we wish to calculate the profit we realise by taking a position in some asset of size $\Delta(t)$ at time $t \in [0,T]$. We use a Brownian motion to determine the price of the asset at time t, along with a filtration $\{\mathcal{F}(t), t \geq 0\}$, for the Brownian motion. We let $\Delta(t)$ be adapted to the filtration $\mathcal{F}(t)$. The total profit we make from this strategy over the time interval should be

$$\int_0^T \Delta(t) dW(t). \tag{2.21}$$

However, Brownian motion paths cannot be differentiated with respect to time and therefore (2.21) is not well defined. Fortunately, mathematician Itô presented a procedure to define this integral.

2.4.1. Construction of the Integral

In this section, we focus on defining (2.21). To this end, we first define the integral for *simple processes*, a process $\Delta(t)$ which is constant in t on each subinterval $[t_j, t_{j+1})$. We can then extend this definition to more general processes. Let $\Pi = \{t_0, t_1, \dots, t_n\}$ be a partition of $[0,T]$. Let $\Delta(t)$ be a simple process on $[0,T]$, which is constant on each $[t_i, t_{i+1})$. Notice that take we the interval to be right open. The value of each subinterval with respect to the change in Brownian motion is

$$I(t_j) = \Delta(t_j)[W(t_{j+1}) - W(t_j)]. \tag{2.22}$$

Summing over all $I(t_j)$ in the interval yields

$$I(t) = \sum_{j=0}^{n} I(t_j) = \sum_{j=0}^{n} \Delta(t_j)[W(t_{j+1}) - W(t_j)]. \tag{2.23}$$

We write the Itô integral for simple processes as

$$I(t) = \int_0^t \Delta(u) dW(u). \tag{2.24}$$

For the more general case, we no longer assume that $\Delta(t)$ is a simple process. Instead, we approximate $\Delta(t)$ by a sequence of simple processes. To do this, we choose a partition $\pi = \{t_0, t_1, \dots, t_n\}$, where we set the approximating simple process equal to the value of $\Delta(t_j)$ for each t_j, over the interval $[t_j, t_{j+1})$. Let $\|\pi\| = \max_{t_1 \le t_i \le t_n}(t_i - t_{i-1})$, the length of the longest subinterval in the partition. As we let $\|\pi\| \to 0$, the simple process becomes a good approximation of the general one. We can choose a sequence $\Delta_n(t)$ of simple processes in the partitions such that

$$\lim_{n\to\infty} \mathbb{E} \int_0^T |\Delta_n(t) - \Delta(t)|^2 dt = 0. \tag{2.25}$$

Since we defined the Itô integral for simple processes in (2.24), we can define the integral for general processes as

$$\int_0^t \Delta(u) dW(u) = \lim_{n\to\infty} \int_0^t \Delta_n(u) dW(u). \tag{2.26}$$

In the remainder of this section, we list a few properties of the Itô integral, without proofs. The proofs can be found in [33, Chapter 4].

Theorem 3. *Let $\{\mathcal{F}(t), t \ge 0\}$ be a filtration, and let $\Delta(t), 0 \le t \le T$ be a stochastic process adapted to $\mathcal{F}(t)$. Then $I(t) = \int_0^t \Delta(u) dW(u)$ has the following properties*

(i) ***(Continuity)*** *The paths of $I(t)$ are continuous.*

(ii) ***(Adaptivity)*** *For each t, $I(t)$ is $\mathcal{F}(t)$ measurable.*

(iii) ***(Linearity)*** *If $I(t) = \int_0^t \Delta(u) dW(u)$ and $J(t) = \int_0^t \Gamma(u) dW(u)$, then $I(t) \pm J(t) = \int_0^t (\Delta(u) \pm \Gamma(u)) dW(u)$. Furthermore, for every constant c, $cI(t) = \int_0^t c\Delta(u) dW(u)$.*

(iv) ***(Martingale)*** *$I(t)$ is a martingale.*

(v) ***(Itô isometry)*** $\mathbb{E}[I^2(t)] = \mathbb{E}\Big[\int_0^t \Delta^2(u) du\Big]$.

2.4.2. Itô's Formula

Let $f(x)$ be a differentiable function and $W(t)$ a Brownian motion. We would like to find an expression for the derivative of $f(W(t))$. Since $W(t)$ is not differentiable, we cannot use the chain rule we know from calculus. Itô's formula solves this, by adding an additional term to account for the quadratic variation of the Brownian motion.

Theorem 4. *Let $f(t,x)$ be a function for which the partial derivatives $f_t(t,x)$, $f_x(t,x)$ and $f_{xx}(t,x)$ are defined and continuous. Let $W(t)$ be a Brownian motion. Then, for $T \ge 0$,*

$$f(T, W(T)) = f(0, W(0)) + \int_0^T \frac{\partial f}{\partial t}(t, W(t)) dt + \int_0^T \frac{\partial f}{\partial x}(t, W(t)) dW(t) + \frac{1}{2}\int_0^T \frac{\partial^2 f}{\partial x^2}(t, W(t)) dt. \tag{2.27}$$

In differential form, which is more convenient but mathematically less precise, (2.27) becomes

$$df(t, W(t)) = \left(\frac{\partial f}{\partial t}(t, W(t)) + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(t, W(t))\right) dt + \frac{\partial f}{\partial x}(t, W(t)) dW(t). \tag{2.28}$$

In the following example we use Itô's formula to derive the dynamics of an asset price $S(t)$.

Example 1. *Let $\{W(t), t \ge 0\}$ be a Brownian motion with associated filtration $\{\mathcal{F}(t), t \ge 0\}$. Furthermore, let $\mu(t), \sigma(t)$ be two processes adapted to the filtration. We define the process*

$$X(t) = \int_0^t \sigma(u) dW(u) + \int_0^t \left(\mu(u) - \frac{1}{2}\sigma^2(u)\right) du, \tag{2.29}$$

which in differential notation reads

$$dX(t) = \sigma(t) dW(t) + \left(\mu(t) - \frac{1}{2}\sigma^2(t)\right) dt. \tag{2.30}$$

We define an asset price process as

$$S(t) = S(0)e^{X(t)} = S(0)\exp\left\{\int_0^t \sigma(u)dW(u) + \int_0^t \left(\mu(u) - \frac{1}{2}\sigma^2(u)\right)du\right\} \tag{2.31}$$

Using Itô's formula, we can derive the dynamics of the price process $dS(t)$

$$dS(t) = S(0)e^{X(t)}dX(t) + \frac{1}{2}S(0)e^{X(t)}dX(t)dX(t) \tag{2.32}$$

$$= S(t)dX(t) + \frac{1}{2}S(t)dX(t)dX(t) \tag{2.33}$$

$$= S(t)\left[\sigma(t)dW(t) + \left(\mu(t) - \frac{1}{2}\sigma^2(t)\right)dt\right] + \frac{1}{2}S(t)\sigma^2(t)dt \tag{2.34}$$

$$= \mu(t)S(t)dt + \sigma(t)S(t)dW(t), \tag{2.35}$$

where we used the quadratic variation $[W,W](t) = t$*, or in differential form* $dW(t)dW(t) = dt$*. We find that the asset price is driven by a drift* $\mu(t)$ *and a volatility coefficient* $\sigma(t)$*, both allowed to be random and vary through time.*

3

Option Pricing

3.1. Option Theory

In finance, an option is a financial product that is based on the value of an underlying asset, oftentimes a stock. An option gives the holder the right to buy or sell the underlying asset at some point in the future, at a price predetermined by the *option writer*. It is important to emphasise that the option holder has the *option* to buy or sell the underlying. Unlike an instrument such as a future, there is no obligation. We make a distinction between two types of option contracts; A call and a put option. A call option gives the holder the right to buy the underlying asset at some fixed *strike price* K, whereas a put option gives the holder the right to sell the underlying asset. In *European* type options, we allow the holder to only exercise its right to buy or sell the underlying asset at the expiry time $t = T$. The option cannot be exercised early (when $t < T$). *American* style options, on the other hand, allow the holder to exercise the option at any time $t_0 \leq t \leq T$, effectively ending the contract. In the remainder of this thesis, we consider an option to be of European type, unless explicitly mentioned otherwise.

When trading options, we assume that the option holder will exercise the contract when a positive profit can be made. In the case of a call option, if the underlying asset price at time $t = T$ is larger than the strike price K, we can exercise the option and sell the asset back to the market, effectively realising a profit of $S(T) - K$. If, however, the asset price at expiry is lower than the strike price K, i.e. when $S(T) \leq K$, there is no profit to be made when exercising the option, since the option holder can buy the asset for a lower price in the market. Thus, the option expires worthless.

Based on the previous example, we can define a *payoff function* $V_c(T, S)$ for a call option on an underlying asset S with expiry T and strike price K as

$$V_c(T, S) = \max\{S(T) - K, 0\}. \tag{3.1}$$

Equivalently, for a put option, we have

$$V_p(T, S) = \max\{K - S(T), 0\}. \tag{3.2}$$

Figure 3.1 shows the payoff diagram of a call and put option. We call an option in-the-money (ITM) if the payoff is positive, at-the-money (ATM) if the payoff is close to 0, and out-of-the-money if the payoff is 0, resulting in a worthless option.

3.2. Black-Scholes Partial Differential Equation

A large topic in quantitative finance revolves around finding the fair value of an option contract before its expiry ($t < T$). In this section, we make several assumptions allowing us to derive an option pricing PDE, notoriously known as the *Black-Scholes* PDE.

We start the derivation by assuming that the underlying asset assumes a geometric Brownian Motion process. We denote with $S(t)$ the value of the underlying asset at time t. With this assumption we obtain for the dynamics of S under the real world measure $\mathbb{P}$

$$dS(t) = \mu S(t)dt + \sigma S(t)dW^{\mathbb{P}}(t), \tag{3.3}$$

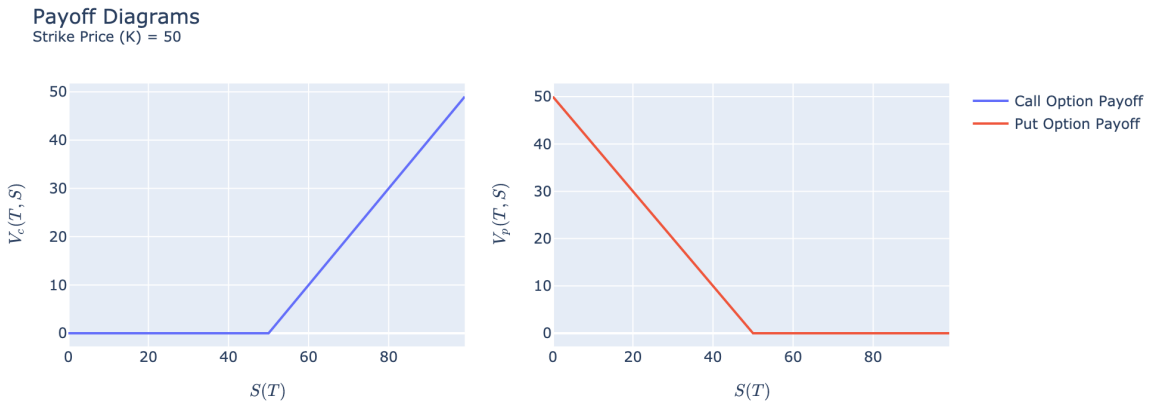

Figure 3.1: The payoff diagram of a call option (left) and a put option (right) with strike price $K = 50$.

where μ denotes the drift parameter, and σ the volatility. Both of these parameters are assumed to be known and constant. Next, we denote with $V(t,S)$ the option contract value, dependent on time t and the stochastic process S. Using Itô's lemma, we derive its dynamics

$$dV(t,S) = \frac{\partial V}{\partial t}dt + \frac{\partial V}{\partial S}dS + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}(dS)^2 \tag{3.4}$$

$$= \left(\frac{\partial V}{\partial t} + \mu S\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}\right)dt + \sigma S\frac{\partial V}{\partial S}dW^{\mathbb{P}}. \tag{3.5}$$

We construct a portfolio $\Pi(t,S)$ containing one option $V(t,S)$ as well as some amount $-\Delta$ of the underlying asset $S(t)$, yielding

$$\Pi(t,S) = V(t,S) - \Delta S(t). \tag{3.6}$$

We use the result of Itô's lemma to derive the dynamics of portfolio Π

$$d\Pi = dV - \Delta dS \tag{3.7}$$

$$= \left(\frac{\partial V}{\partial t} + \mu S\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}\right)dt + \sigma S\frac{\partial V}{\partial S}dW^{\mathbb{P}} - \Delta\left(\mu S dt + \sigma S dW^{\mathbb{P}}\right) \tag{3.8}$$

$$= \left[\frac{\partial V}{\partial t} + \mu S\left(\frac{\partial V}{\partial S} - \Delta\right) + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}\right]dt + \sigma S\left(\frac{\partial V}{\partial S} - \Delta\right)dW^{\mathbb{P}}. \tag{3.9}$$

Due to the $\sigma S\left(\frac{\partial V}{\partial S} - \Delta\right)dW^{\mathbb{P}}$ term, this portfolio contains random fluctuations. We can remove the portfolio's additional random fluctuations by choosing

$$\Delta = \frac{\partial V}{\partial S}. \tag{3.10}$$

Then, equation 3.7 becomes

$$d\Pi = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}\right)dt. \tag{3.11}$$

In order to avoid arbitrage, the value of this portfolio should generate exactly the same return as money invested in a risk-free savings account. For an amount equal to $\Pi(t,S)$, we can model this as

$$d\Pi = r\Pi dt \tag{3.12}$$

$$= r\left(V - S\frac{\partial V}{\partial S}\right)dt \tag{3.13}$$

where r denotes the risk-free rate. In the second equality, we used equations 3.6 and 3.10. Finally, by equating equations 3.11 and 3.12 and dividing by dt, we obtain the Black-Scholes PDE:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV = 0. \tag{3.14}$$

This parabolic PDE requires an additional condition to be well-posed. Indeed, we know that the final condition of this PDE should be the payoff function determined in equation 3.1 or 3.2 depending on the option type.

3.2.1. One-Dimensional Black-Scholes

There exists an analytical solution for the one dimensional Black-Scholes PDE (equation 3.14). To derive this solution, we introduce the Feynman-Kac theorem.

Theorem 5. *Consider the partial differential equation*

$$\frac{\partial V}{\partial t} + \mu(t,S)\frac{\partial V}{\partial S} + \frac{1}{2}\sigma(t,S)^2 \frac{\partial^2 V}{\partial S^2} - rV = 0 \tag{3.15}$$

$$V(T,S) = \psi(T,S), \tag{3.16}$$

defined for $S(t) \in \mathbb{R}$ and $t \in [0,T]$, where μ, σ are known functions, and r constant. The solution $V(t,S)$ is then given by

$$V(t,S) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\Big[\psi(T,S)|\mathcal{F}(t)\Big], \tag{3.17}$$

under the risk-neutral probability measure $\mathbb{Q}$, with respect to a process S, defined by

$$dS(t) = \mu(t,S)dt + \sigma(t,S)dW^{\mathbb{Q}}(t). \tag{3.18}$$

A proof of this theorem can be found in [17]. Using theorem 5 we can express the Black-Scholes PDE as

$$V(t,S) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\Big[\psi(T,S)|\mathcal{F}(t)\Big]. \tag{3.19}$$

In equation 3.3 we see that S is a dependent variable. In order to change this, we apply the transformation $X(t) := \log S(t)$. Using Itô's lemma we obtain

$$dX(t) = d\log S(t) = \left(\mu - \frac{1}{2}\sigma^2\right)dt + \sigma dW(t) \tag{3.20}$$

$$\log S(t) - \log(S_0) = \left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma W(t) \tag{3.21}$$

$$S(t) = S_0 \exp\left\{\left(r - \frac{1}{2}\sigma^2\right)t + \sigma W(t)\right\}. \tag{3.22}$$

Notice that

$$Y := \frac{W(T) - W(t)}{\sqrt{T-t}} \sim \mathcal{N}(0,1). \tag{3.23}$$

Also, $S(T) > K$ implies

$$S_t \exp\left\{\left(r - \frac{1}{2}\sigma^2\right)(T-t) + \sigma\sqrt{T-t}y)\right\} > K \tag{3.24}$$

$$y < d_2 := \frac{\log\frac{S_t}{K} + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}. \tag{3.25}$$

$$\tag{3.26}$$

We can now use theorem 5 to obtain for a call option

$$V_c(t,S) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\Big[S(T)\mathbb{1}_{\{S(T)>K\}}\Big|\mathcal{F}(t)\Big] - e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\Big[K\mathbb{1}_{\{S(T)>K\}}\Big|\mathcal{F}(t)\Big] \tag{3.27}$$

$$= e^{-r(T-t)}\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{d_2} S_t \exp\left\{\left(r-\frac{1}{2}\sigma^2\right)(T-t)+\sigma\sqrt{T-t}y-\frac{1}{2}y^2\right\}dy - e^{-r(T-t)}K\Phi(d_2) \tag{3.28}$$

$$= S(t)\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{d_2}\exp\Big\{-\frac{1}{2}\left(y+\sigma\sqrt{T-t}\right)^2\Big\}dy - e^{-r(T-t)}K\Phi(d_2) \tag{3.29}$$

$$= S(t)\Phi(d_1) - e^{-r(T-t)}K\Phi(d_2), \tag{3.30}$$

where

$$d_1 := d_2 + \sigma\sqrt{T-t} = \frac{\log\frac{S_t}{K} + (r+\frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \tag{3.31}$$

and $\Phi(x)$ the CDF of the standard normal distribution. The computation for a put option is analogous, and yields

$$V_p(t,S) = e^{-r(T-t)}K\Phi(-d_2) - S(t)\Phi(-d_1). \tag{3.32}$$

3.2.2. Multidimensional Black-Scholes

In this section, we consider the multidimensional Black-Scholes PDE [36]. We have $S_t = \left(S_t^1, \dots, S_t^d\right)$ the vector of underlying assets in d dimensions. We assume that each of the assets S^i are modelled by a geometric Brownian motion. We express the multidimensional Black-Scholes equation in logarithmic asset variables, $S_t^i = e^{x_t^i}$. Then

$$\frac{\partial V}{\partial t}(t,x) - \sum_{i=1}^{d}\left(r-\frac{\sigma_i^2}{2}\right)\frac{\partial V}{\partial x_i}(t,x) - \sum_{i,j=1}^{d}\frac{\rho_{ij}\sigma_i\sigma_j}{2}\frac{\partial^2 V}{\partial x_i \partial x_j}V(t,x) + rV(t,x) = 0, \tag{3.33}$$

with r the risk-free rate, σ_i the volatility of underlying S_i, and ρ_{ij} the correlation between underlyings i and j. For the payoff function, we choose

$$g(x) = G(e^x) = \max\left\{\frac{1}{d}\sum_{i=1}^{d} e^{x_i} - K, 0\right\}. \tag{3.34}$$

The payoff (3.34) denotes a European basket option where each of the constituents have equal weight. We fix the strike price K to some positive constant.

3.2.3. Gauß-Hermite Pricing Technique

In order to validate the output from the trained neural network, we must construct an alternative pricing technique. The default option for this would be a Monte-Carlo simulation. However, Monte-Carlo simulation is less accurate than alternative pricers, and requires a considerable amount of computational power to achieve acceptable accuracy. Therefore, in addition to Monte-Carlo simulation, we explore an alternative method using Gauß-Hermite polynomials. It turns out that this method is working well as a solution for (3.33). This technique is outlined in [7, 8]. In this section, we follow results and derivations from both to construct the pricing technique.

We would like to solve a d-dimensional Black-Scholes option pricing problem. The payoff function defined in (3.34) is not differentiable. To circumvent this problem, we split the problem into a one dimensional and $d-1$ dimensional Black-Scholes problem. We can use the analytical solution to solve the first. The second is a smooth problem, on which we apply Gauß-Hermite quadrature. We consider the PDE from equation 3.33. The solution for the individual geometric Brownian motion assets is

$$S_t^i = S_0^i e^{(r-\frac{1}{2}\sigma_i^2)t+\sigma_i W_t^i}, i = 1, \dots, d. \tag{3.35}$$

We can write the price of the call option using (3.34) as

$$V_c(t,S) = e^{-rT}\mathbb{E}\left[\left(\sum_{i=1}^{d} w_i S_T^i - K\right)^+\right] \tag{3.36}$$

$$= e^{-rT}\mathbb{E}\left[\left(\sum_{i=1}^{d} w_i S_0^i e^{(r-\frac{1}{2}\sigma_i^2)t+\sigma_i W_t^i} - K\right)^+\right] \tag{3.37}$$

$$= \mathbb{E}\left[\left(\sum_{i=1}^{d} \beta_i e^{X_i} - e^{-rT}K\right)^+\right]. \tag{3.38}$$

with $\vec{X} \sim \mathcal{N}_d(0,\Sigma)$, where we have

$$\beta_i = w_i S_0^i e^{-\frac{1}{2}\sigma_i^2 T}, \tag{3.39}$$

$$\Sigma_{ij} = \sigma_i \sigma_j \rho_{ij} T. \tag{3.40}$$

We can now write the problem as a d-dimensional integral over the density of $\vec{X}$

$$\mathbb{E}\left[\left(\sum_{i=1}^{d} \beta_i e^{X_i} - e^{-rT}K\right)^+\right] = \int_{\mathbb{R}^d}\left(\sum_{i=1}^{d} \beta_i e^{X_i} - e^{-rT}K\right)^+ \phi_{\vec{X}}(\vec{x})d\vec{x}. \tag{3.41}$$

As mentioned, (3.41) is not smooth. In [7], a transformation matrix V is found such that $\vec{X} = V\vec{Y}$. Note that the components of Y are independent. This allows us to write equation 3.38 as

$$\mathbb{E}\left[\left(\sum_{i=1}^{d} \beta_i e^{X_i} - e^{-rT}K\right)^+\right] = \mathbb{E}\left[\left(\sum_{i=1}^{d} \beta_i e^{(VY)_i} - e^{-rT}K\right)^+\right] \tag{3.42}$$

$$= \mathbb{E}\left[\left(\sum_{i=1}^{d} \beta_i e^{Y_1+\sum_{j=2}^{d} V_{i,j}Y_j} - e^{-rT}K\right)^+\right] \tag{3.43}$$

$$= \mathbb{E}\left[\left(e^{Y_1}h(\tilde{Y}) - e^{-rT}K\right)^+\right] \tag{3.44}$$

$$= \mathbb{E}\left[\mathbb{E}\left[\left(e^{Y_1}h(\tilde{Y})e^{-rT} - K\right)^+ | \tilde{Y}\right]\right]. \tag{3.45}$$

with $h(\tilde{Y}) = \sum_{i=1}^{d}\beta_i \exp\{\sum_{j=2}^{d} V_{i,j}\tilde{Y}_j\}$ a differentiable function, $\tilde{Y} = (Y_2, \ldots, Y_d) \sim \mathcal{N}_{d-1}(0,D)$ independent of Y_1, and D diagonal matrix with values λ_i^2. For the final equality, we used [7, Lemma 3.2]. Lemma 3.1 in [7] proves that such a V exists, and Proposition 13 in [8] elaborates on how to find the transformation. Notice that we reduced the dimension of the outer expectation by one. The inner expectation can be calculated analytically as a one-dimensional Black-Scholes problem using [7, Lemma 3.2]. We can use Gauß-Hermite polynomials to solve the $(d-1)$-dimensional problem [30],

$$\mathbb{E}\left[e^{Y_1}h(\tilde{Y})\right] \approx e^{\lambda_1^2/2}\frac{1}{\sqrt{\pi}}\sum_{i=1}^{n} w_i h(\sqrt{2}\sigma y_i). \tag{3.46}$$

3.3. Implied Volatility

Suppose we see a call option in the market on some stock S with given interest rate r, maturity T and strike price K, which we will denote as $V_c^M(K,T)$. The volatility entered into the Black-Scholes equation which reproduces the price of $V_c^M(K,T)$ quoted in the market is defined as the implied volatility. Specifically, in the case of a call option:

Definition 7. *The Black-Scholes implied volatility σ_{imp}, which we refer to as implied volatility, is the value of σ for which*

$$V_c(t_0, S; K, T, \sigma_{imp}, r) = V_c^M(K, T), \tag{3.47}$$

where $t_0 = 0$, the current time.

We provide two numerical techniques to approximate the implied volatility, as there is no closed form solution available. These techniques are the Newton-Raphson iterative method and Brent's method.

3.3.1. Newton-Raphson Method

We can transform the equation stated in definition 7 to a root-finding problem

$$g(\sigma_{imp}) := V_c^M(K, T) - V_c(t_0, S_0; K, T, \sigma_{imp}, r) = 0. \tag{3.48}$$

Denote with $\sigma_{imp}^{(0)}$ the initial guess of the sequence. We can find the consecutive approximations of σ_{imp} by computing the Newton-Raphson iterative formula

$$\sigma_{imp}^{(k+1)} = \sigma_{imp}^{(k)} - \frac{g(\sigma_{imp}^{(k)})}{g'(\sigma_{imp}^{(k)})}, \tag{3.49}$$

where

$$g'(\sigma_{imp}) = -\frac{\partial V(t_0, S_0; K, T, \sigma, r)}{\partial \sigma} \tag{3.50}$$

$$= -Ke^{-r(T-t_0)}\Phi(d_2)\sqrt{T - t_0}, \tag{3.51}$$

where d_2 is as in equation 3.24, and the second equation is the result of taking the derivative of equation 3.27 with respect to σ. Since there exists an analytical solution for $g'(\sigma_{imp})$, the iterative expression in equation 3.48 can be found analytically as well.

3.3.2. Brent's Method

The Newton-Raphson method is arguably the fastest root-finding algorithm to compute the implied volatility. The method converges quadratically given that the initial guess is close to a root. However, we encounter convergence issues and numerical instability when $g'(\sigma_{imp}) \approx 0$. This is the case when the derivative of the option price with respect to the volatility is low, happening in far ITM or OTM options, i.e. options with strike prices far from the underlying spot price. To circumvent this problem, we introduce three techniques alongside the Newton-Raphson root-finding method. The combination of these techniques, known as Brent's method, results in a robust algorithm with computation times close to those of Newton-Raphson's.

The first technique is called the bisection method. This is a root-finding method which repeatedly bisects the interval by selecting the subinterval where the function changes sign. This interval must therefore contain the root. Algorithm 1 shows the pseudocode for the bisection method. Since this method is relatively slow, we should only use it to get to rough approximations.

Albeit slow, the bisection method is a robust method. The algorithm is used in Brent's method as a failsafe, when all other methods fail due to convergence issues.

The second method we incorporate in the Brent's method is the secant method. This method is identical to Newton-Raphson, except a finite difference approximation is done instead of calculating the derivative. The reason for this is that Brent's method is a derivative-free algorithm. We can therefore not rely upon derivatives of the input function. The secant method is defined as

$$\sigma^{(k+1)} = \sigma^{(k)} - g(\sigma^{(k)})\frac{\sigma^{(k)} - \sigma^{(k-1)}}{g(\sigma^{(k)}) - g(\sigma^{(k-1)})}. \tag{3.52}$$

The third and final technique is the inverse quadratic interpolation technique. It is a more complex method than the secant method, but improves convergence as we use higher order polynomials as

Algorithm 1 The bisection algorithm

Require: $max\_iterations, tol > 0, \sigma_0^- < \sigma_0^+$ and $g(\sigma_0^-) \leq 0 \leq g(\sigma_0^+)$
  $k \leftarrow 1$
  while $k \leq max\_iterations$ **do**
    $\sigma_{imp} = (\sigma_0^+ + \sigma_0^-)/2$
    if $g(\sigma_{imp}) = 0$ or $(\sigma_0^+ - \sigma_0^-)/2 < tol$ **then exit**
    end if
    $k \leftarrow k + 1$
    if $sign(g(\sigma_{imp})) = sign(g(\sigma_0^-))$ **then**
      $\sigma_0^- \leftarrow \sigma_{imp}$
    else
      $\sigma_0^+ \leftarrow \sigma_{imp}$
    end if
  end while

approximation. This technique is based on three previously computed values used to compute the second degree Lagrange interpolating polynomial. Using this, the interpolation is given by

$$\sigma^{(k+1)} = \frac{g(\sigma^{(k-1)})g(\sigma^{(k-2)})\sigma^{(k)}}{(g(\sigma^{(k)}) - g(\sigma^{(k-1)}))(g(\sigma^{(k)}) - g(\sigma^{(k-2)}))} \tag{3.53}$$

$$+ \frac{g(\sigma^{(k-2)})g(\sigma^{(k)})\sigma^{(k-1)}}{(g(\sigma^{(k-1)}) - g(\sigma^{(k-2)}))(g(\sigma^{(k-1)}) - g(\sigma^{(k)}))} \tag{3.54}$$

$$+ \frac{g(\sigma^{(k-1)})g(\sigma^{(k)})\sigma^{(k-2)}}{(g(\sigma^{(k-2)}) - g(\sigma^{(k-1)}))(g(\sigma^{(k-2)}) - g(\sigma^{(k)}))}. \tag{3.55}$$

Brent's method is a combination of the inverse quadratic interpolation, secant and bisection algorithms. A high level pseudocode algorithm is given in algorithm 2.

Algorithm 2 Brent's method

Require: $tol > 0, \sigma^- < \sigma^+$ and $g(\sigma^-) \leq 0 \leq g(\sigma^+)$
  $\sigma \leftarrow \frac{\sigma^- + \sigma^+}{2}$
  while $|g(\sigma)| \leq tol$ **do**
    if $\sigma_- < \sigma < \sigma+$ **then** use the inverse quadratic interpolation method to obtain σ_{next}.
  Depending on σ_{next} in $[\sigma_-, \sigma]$ or $[\sigma, \sigma^+]$, set σ_-, σ_+ as the new borders in which the interval is contained.
    else if $\sigma_- < \sigma_+$ **then** use the secant method to obtain σ_{next}. If $\sigma_{next} \in (\sigma_-, \sigma_+)$, set $\sigma \leftarrow \sigma_{next}$.
    else use the bisection method.
    end if
  end while
  return σ_{next}

Further speed comparisons of the discussed algorithms can be found in [25, Table 8].

3.3.3. Skew & Smile

The Black-Scholes model makes the assumption that the volatility is a known constant. However, when looking at real market option prices on the same underlying for different strike prices and fixed maturity, instead of a constant, we see a so-called implied volatility skew, visualised in figure 3.2.

One of the major shortcomings of the Black-Scholes model is that it is unable to account for the skew or smile we see in the market, because of the assumption of constant volatility. Alternative models have been developed based without the assumption of constant volatility. Such models include local volatility models, stochastic volatility models and jump diffusion models. In the next section, we derive a stochastic volatility model named the Heston model, which does not assume constant volatility and instead can be modelled as a stochastic process.

Figure 3.2: The market implied volatility. The implied volatility curve looks like a smile. Data is obtained from Cboe's SPX weekly expiring option sample, which is an option on the S&P Future.

3.4. Heston Partial Differential Equation

In section 3.2 we discussed the Black-Scholes PDE, and observed that the volatility was considered constant in this model. In section 3.3 we argued that this assumption does not align with observations from market data, in which we often observe a skew (figure 3.2). In this section, we present the Heston PDE and discuss its improvements over the Black-Scholes PDE with the addition of its stochastic volatility. In chapter 5 we construct a model to solve this PDE with a neural network in higher dimensions. For this, we must derive the multidimensional Heston PDE.

3.4.1. One-Dimensional Heston

We follow the derivation from [10] to construct the Heston PDE in one dimension. In the following section, we extend this derivation to multiple dimensions. We start with the dynamics of the underlying asset $S(t)$, which we model as a geometric Brownian motion as in equation 3.3. We add a stochastic differential equation to describe to variance process $v(t)$, which we replace with the diffusion coefficient in $S(t)$. Under the risk-neutral measure $\mathbb{Q}$ we obtain the following two SDE's

$$\begin{cases} dS(t) = rS(t)dt + \sqrt{v(t)}S(t)dW_x(t), & S(t_0) = S_0 > 0 \\ dv(t) = \kappa(\bar{v} - v(t))dt + \gamma\sqrt{v(t)}dW_v(t), & v(t_0) = v_0 > 0, \end{cases} \tag{3.56}$$

with $dW_v(t)dW_x(t) = \rho dt$ the correlation between the Brownian motions.

Using Itô's lemma for multidimensional processes [10, eq 7.10], we can determine the dynamics of dV

$$dV = \left(\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \kappa(\bar{v} - v)\frac{\partial V}{\partial v} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\gamma Sv\frac{\partial^2 V}{\partial S \partial v} + \frac{1}{2}\gamma^2 v\frac{\partial^2 V}{\partial v^2}\right)dt \tag{3.57}$$

$$+ S\sqrt{v}\frac{\partial V}{\partial S}dW_x + \gamma\sqrt{v}\frac{\partial V}{\partial v}dW_v. \tag{3.58}$$

To construct the Heston PDE, we use Feynman-Kac (theorem 5) and represent the option price as a risk-neutral discounted expectation

$$V(t,S,v) = e^{rt}\mathbb{E}^{\mathbb{Q}}\left[e^{-rT}V(T,S,v)|\mathcal{F}(t)\right]. \tag{3.59}$$

Denote with $M(t) := e^{rt}$, then $dM(t) = rM(t)dt$. Dividing (3.59) by $M(t)$ yields

$$\frac{V(t,S,v)}{M(t)} = \mathbb{E}^{\mathbb{Q}}\left[\frac{V(T,S,v)}{M(T)}|\mathcal{F}(t)\right]. \tag{3.60}$$

Because of the risk-neutral measure, the righthand side expectation should be a martingale. Using Itô's lemma, we can determine the dynamics of the lefthand side

$$d\left(\frac{V}{M}\right) = \frac{1}{M}dV - r\frac{V}{M}dt, \tag{3.61}$$

which should also be a martingale. Using (3.57) and the martingale property we have that all the drift terms should be 0. Thus, after multiplication with M, we have

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \kappa(\bar{v} - v)\frac{\partial V}{\partial v} + \frac{1}{2}vS^2\frac{\partial^2 V}{\partial S^2} + \rho\gamma Sv\frac{\partial^2 V}{\partial S \partial v} + \frac{1}{2}\gamma^2 v\frac{\partial^2 V}{\partial v^2} - rV = 0, \tag{3.62}$$

which is the Heston PDE. We can solve this PDE using its characteristic function and a Fourier based method very efficiently, as further outlined in [10, Chapter 6, 8.4]. In the remainder of this section, we derive one such Fourier based method, called the COS method.

COS Method

In this section, we examine a Fourier based method to price a European call option using the Heston model. This method is originally developed in [16], and provides a fast and reliable approximation of the option price using Fourier cosine series. We will be using this pricing method to compute the option price under the Heston model in 4.4.

We follow the derivation in [16], in which we start with writing the expectation in equation 3.59 as a truncated integral $[a, b] \in \mathbb{R}$ over the probability density function

$$V(t,S) = e^{rt}\mathbb{E}^{\mathbb{Q}}\left[e^{-rT}V(T,S,v)|\mathcal{F}(t)\right] \approx e^{-r(T-t)}\int_a^b v(y,T)f(y|x)dy, \tag{3.63}$$

where x and y state variables at t and T, respectively. Since the density function is not known in closed form, we replace it by its Fourier cosine expansion in y, yielding

$$f(y|x) = \sum_{k=0}^{\infty}{}' A_k(x)\cos\left(k\pi\frac{y-a}{b-a}\right) \tag{3.64}$$

where

$$A_k(x) := \frac{2}{b-a}\int_a^b f(y|x)\cos\left(k\pi\frac{y-a}{b-a}\right)dy, \tag{3.65}$$

after applying a change of variables from the interval $[0, \pi]$ to $[a, b]$. The primed sum denotes that the first term must be multiplied by $\frac{1}{2}$. Substitution into (3.63) then results in

$$V(t,S) \approx e^{-r(T-t)}\int_a^b v(y,T)\sum_{k=0}^{\infty}{}' A_k(x)\cos\left(k\pi\frac{y-a}{b-a}\right)dy. \tag{3.66}$$

We interchange summation and integration to obtain

$$V(t,S) \approx \frac{1}{2}(b-a)e^{-r(T-t)}\sum_{k=0}^{\infty}{}' A_k(x)V_k, \tag{3.67}$$

with

$$V_k := \frac{2}{b-a}\int_a^b v(y,T)\cos\left(k\pi\frac{y-a}{b-a}\right)dy. \tag{3.68}$$

Finally, we can approximate the integrals in $A_k(x)$ by $F_k(x)$, using the characteristic function ϕ, the Fourier transform of the density function

$$F_k(x) = \frac{2}{b-a}\Re\left\{\phi\left(\frac{k\pi}{b-a}\right)\exp\left(-i\frac{ka\pi}{b-a}\right)\right\}, \tag{3.69}$$

where $\Re\{\cdot\}$ denotes taking the real part of the expression inside the brackets. After truncating the infinite sum, we obtain

$$V(t,S) \approx e^{-r(T-t)} \sum_{k=0}^{N-1}{}' \Re\left\{\phi\left(\frac{k\pi}{b-a}; x\right) \exp\left(-i\frac{ka\pi}{b-a}\right)\right\} V_k. \tag{3.70}$$

For the European call option, we can calculate the coefficients V_k analytically, which is done in [16, Result 2.3.1]. We list the result below.

$$V_k^{call} = \frac{2}{b-a}\int_0^b K\left(e^y - 1\right)\cos\left(k\pi\frac{y-a}{b-a}\right)dy = \frac{2}{b-a}K\left(\chi_k(0,b) - \psi_k(0,b)\right), \tag{3.71}$$

with

$$\chi_k(c,d) := \frac{1}{1+\left(\frac{k\pi}{b-a}\right)^2}\left[\cos\left(k\pi\frac{d-a}{b-a}\right)e^d - \cos\left(k\pi\frac{c-a}{b-a}\right)e^c\right. \tag{3.72}$$

$$\left. + \frac{k\pi}{b-a}\sin\left(k\pi\frac{d-a}{b-a}\right)e^d - \frac{k\pi}{b-a}\sin\left(k\pi\frac{c-a}{b-a}\right)e^c\right], \tag{3.73}$$

and

$$\psi_k(c,d) := \begin{cases}\left[\sin\left(k\pi\frac{d-a}{b-a}\right) - \sin\left(k\pi\frac{c-a}{b-a}\right)\right]\frac{b-a}{k\pi}, & k \neq 0,\\ (d-c), & k = 0.\end{cases} \tag{3.74}$$

3.4.2. Multidimensional Heston

In section 3.4.1 we derived the Heston PDE for one underlying asset. In this section, we derive a Heston PDE for multiple assets driven by one variance process $v(t)$. Consequently, we will use this multidimensional PDE in chapter 5, in which we approximate the solution of higher dimensional option pricing problems using neural networks. In the remainder of this section, we also construct two alternative pricing methods, being

- Monte-Carlo Simulation: This method is robust, but computationally expensive in higher dimensions. We will use this method sparingly to evaluate the PDE models.
- Fourier Transforms: Using the characteristic function for a 2D Heston model from [14], we build a Fourier based approximation for a Heston PDE with 2 underlying assets.

The derivation of the multidimensional Heston PDE is very similar to the one in section 3.4.1. Consider the following SDE's, in which we have d assets

$$\begin{cases} dS_i(t) = rS_i(t)dt + \sqrt{v(t)}S_i(t)dW_i(t), & i = 1,\dots,d\\ dv(t) = \kappa(\bar{v} - v(t))dt + \gamma\sqrt{v(t)}dW_v(t), & v(t_0) = v_0 > 0, \end{cases} \tag{3.75}$$

and $\vec{S}(0) = \vec{S}_0 = (S_0^1, \dots, S_0^d)^T$. Following section 3.4.1, we obtain as in (3.60)

$$\frac{V(t,\vec{S},v)}{M(t)} = \mathbb{E}^{\mathbb{Q}}\left[\frac{V(T,\vec{S},v)}{M(T)}|\mathcal{F}(t)\right]. \tag{3.76}$$

For the dynamics dV we find

$$dV = \left(\frac{\partial V}{\partial t} + r\sum_{i=1}^{d} S_i\frac{\partial V}{\partial S_i} + \kappa(\bar{v}-v)\frac{\partial V}{\partial v} + \frac{1}{2}v\sum_{i=1}^{d}\sum_{j=1}^{d}\rho_{ij}S_iS_j\frac{\partial^2 V}{\partial S_i\partial S_j}\right. \tag{3.77}$$

$$\left. + \gamma v\sum_{i=1}^{d}\rho_{v,i}S_i\frac{\partial^2 V}{\partial S_i\partial v} + \frac{1}{2}\gamma^2 v\frac{\partial^2 V}{\partial v^2}\right)dt + \sum_{i=1}^{d} S_i\sqrt{v}\frac{\partial V}{\partial S_i}dW_i + \gamma\sqrt{v}\frac{\partial V}{\partial v}dW_v, \tag{3.78}$$

where ρ_{ij} is the correlation between asset S_i, S_j, and $\rho_{v,i}$ is the correlation between the variance process and asset S_i. Because of the martingale property, we must have

$$\frac{\partial V}{\partial t} + r\sum_{i=1}^{d} S_i \frac{\partial V}{\partial S_i} + \kappa(\bar{v} - v)\frac{\partial V}{\partial v} + \frac{1}{2}v\sum_{i=1}^{d}\sum_{j=1}^{d}\rho_{ij}S_iS_j\frac{\partial^2 V}{\partial S_i \partial S_j} \tag{3.79}$$

$$+ \gamma v\sum_{i=1}^{d}\rho_{v,i}S_i\frac{\partial^2 V}{\partial S_i \partial v} + \frac{1}{2}\gamma^2 v\frac{\partial^2 V}{\partial v^2} - rV = 0, \tag{3.80}$$

yielding the d-dimensional Heston PDE with a single variance process v.

3.4.3. Numerical Methods

In this section we develop two numerical approximation methods for the multidimensional Heston PDE. The first one is Monte-Carlo simulation, and the second one is a Fourier based approximation technique.

Monte-Carlo Simulation

In [38] a scheme is developed for multidimensional Heston Monte-Carlo sampling. Instead of considering d variance processes, for each asset, we consider only one variance process driving the assets using correlations. To this end, we define a correlation matrix of dimension $(D+1)\times(D+1)$ as

$$\Sigma = \begin{pmatrix} 1 & \rho_{12} & \cdots & \rho_{1d} & \rho_{1,v} \\ \rho_{12} & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & & \\ \rho_{1d} & & \cdots & 1 & \rho_{d,v} \\ \rho_{1,v} & \rho_{2,v} & \cdots & \rho_{d,v} & 1 \end{pmatrix}, \tag{3.81}$$

where $\{\rho_{ij}\}$ are the asset correlations and $\{\rho_{i,v}\}$ the variance process correlations with the asset.

Using a Cholesky decomposition, we can find matrix L such that

$$\Sigma = LL^T. \tag{3.82}$$

We must correlate independent Brownian motions W_i by performing the matrix-vector multiplication $L\cdot\vec{W}$. For the simulation of the asset price processes, we use the $\log$ transformation of the asset prices. System (3.75) then becomes

$$\begin{cases} dX_i(t) = \left(r - \frac{1}{2}v(t)\right)dt + \sqrt{v(t)}dW_i(t), & i = 1,\dots,d \\ dv(t) = \kappa(\bar{v} - v(t))dt + \gamma\sqrt{v(t)}dW_v(t), & v(t_0) = v_0 > 0, \end{cases} \tag{3.83}$$

and $\vec{X}(0) = \vec{X}_0 = \log\vec{S}_0$. Let $\tilde{W} = (\tilde{W}_1,\dots,\tilde{W}_{d+1})^T$ be a vector of $d+1$ independent Brownian motions. The vectorised process using the Cholesky decomposition to correlate the independent Brownian motions

$$d\begin{bmatrix}\vec{X}(t)\\ v(t)\end{bmatrix} = \begin{bmatrix} r - \frac{1}{2}v(t)\\ \kappa(\bar{v} - v)\end{bmatrix}dt + \begin{bmatrix}\sqrt{v(t)}\\ \gamma\sqrt{v(t)}\end{bmatrix}dL\tilde{W}. \tag{3.84}$$

As mentioned in [38], we must perform the sampling step for $v_i(t)$ before $X_i(t)$.

Fourier Series

In this section, we present an Fourier based direct integration method. The COS method, presented in 3.4.1, is challenging to generalise to a multidimensional case, since the analytic formula for the coefficients of the payoff function (equation 3.68) should be recovered numerically [6]. Instead, we use results from [14]. In particular, [14, Theorem 3.2] provides a pricing function in d spatial dimensions. In the remainder of this section, we briefly state the assumptions that must be attained.

Denote with M_{X_T} the moment generating function of a $\mathbb{R}^d$-valued random variable X_T, such that

$$M_{X_T}(u) = \mathbb{E}\left[e^{uX_T}\right] = \phi_{X_T}(-iu), \tag{3.85}$$

where ϕ_{X_T} is the characteristic function of X_T. Further, $\vec{s} = (s^1, \dots, s^d) \in \mathbb{R}^d$ with $s_i = -\log S_0^i$, the log transform of the underlying, and $\hat{f}$ the Fourier transform of a measurable function f. Let $R \in \mathbb{R}^d$ be a damping factor, and define

$$g(x) := e^{-\sum_k R_k x_k} f(x), \qquad \text{for } x \in \mathbb{R}^d. \tag{3.86}$$

We assume that

- g is a bounded continuous function in $L^1(\mathbb{R}^d)$;
- $M_{X_T}(R)$ exists;
- $\hat{g} \in L^1(\mathbb{R}^d)$.

In this case, the option price at time $t = 0$ is given by

$$V_f(X; \vec{s}) = \frac{e^{-\sum_k R_k s_k}}{(2\pi)^d} \int_{\mathbb{R}^d} e^{-\sum_k u_k s_k} M_{X_T}(R + iu) \hat{f}(iR - u) du. \tag{3.87}$$

A proof, along with other useful results can be found in [14]. In section 5.4 we will use this method as a reference pricer for the minimum of 2 options using the Heston model. The moment generating function of the two dimensional Heston model for the vector $\vec{s} = (s^1, s^2)$ can be found in [26].

4

Neural Networks

In this chapter we examine neural networks. First, in section 4.1, we discuss a possible contribution of neural networks to option pricing. We argue that a neural network can be used as an approximation tool in combination with existing techniques. Consequently, we discuss the fundamentals of a neural network (section 4.2), which we require to create network variations suited for our specific approximation problems in section 4.3. Finally, in 4.4 we implement and analyse the discussed networks on both the Black-Scholes and Heston vanilla European call option problems, as well as the implied volatility problem.

4.1. Networks in Option Pricing

Before we look at the individual components embodying a neural network, we briefly motivate the reasons behind using this technology, and explain two applications in finance. In section 3.2 we constructed the Black-Scholes pricing equation, attempting to price an option given a few parameters from the market, such as the interest rate r and the realised volatility σ. We argued in section 3.3.3 that the Black-Scholes model did not account for the implied volatility skew that we often find in the market. In addition to this shortcoming, the assumption of a geometric Brownian motion for modelling the underlying asset does not conform with the market, where we often see more heavy-tailed distributions. However, more complex models that do account for these shortcomings often do not have an analytical solution. In some cases, Monte-Carlo simulation is the main approach to evaluate such models, leading to computationally expensive and slower evaluation. Instead of trying to create a model based on assumptions of distributions, another approach would be to assemble a dataset from (historical) market data with *features* along with the quoted market price as the *labels*. Using the features $\vec{x}$ and the labels y, we try to find the function $\hat{f}$ best approximating the labels. That is,

$$\hat{f} = arg\min_{f} \|f(\vec{x}) - y\|. \tag{4.1}$$

We effectively reformulated the option pricing problem into an optimisation problem. The neural network architecture, as we will see in the following sections, is designed to approximate any nonlinear function using many small interconnected building blocks, much like how neurons in a brain function. One might consider neural networks to be the 'holy grail': a straightforward optimisation problem providing a solution to any problem, as long as we have sufficient data and computation power. However, a few pitfalls can be identified, which make a network as a standalone pricing function, i.e. without the combination of other techniques such as a stochastic model, inconvenient. Namely

- If we train the network on historical data, it will likely not generalise well for unseen data. A stochastic model, on the other hand, generally provides a coherent solution on the entire domain. We cannot expect the model to perform well in edge cases.
- In finance, we are often concerned with managing risk. Using a stochastic model, we can quantify the response of varying input parameters, and can therefore quantify the risk corresponding to the stochastic model output. A neural network, on the other hand, ingests data and produces an

output, making it considerably harder to quantify the effect of changed input variables given its output.

- Training a model on historical data implies that history is a valid representation for the future. This is a very strong assumption to make, and can often not be justified. The concept of a filtration used in stochastic processes (section 2.3) ensures that the future is independent of the past and known asset process up to the present time.

These reasons make it difficult to justify using only a neural network as pricing function in finance. However, neural networks can also be used as a numerical method. Neural networks can become useful in addition to a stochastic model with no analytical solution. Indeed, learning a map from the parameters of the stochastic model to the output saves costly time during inference, as we do not require expensive computations such as Monte-Carlo simulation. Additionally, we can use a neural network as a numerical method to approximate the implied volatility surface. In section 3.3 we discussed two numerical methods to approximate the implied volatility. Using a neural network for these type of approximations can yield the following benefits

- We are not dependent on market data. Since we are learning a mapping from a stochastic model, we can sample infinitely many datapoints and compute the output using the stochastic model. We will therefore never have a lack of data.
- We can do training of the model off-line. The training process will be computationally intensive. Once the model is trained, the speed of inference is comparable to that of an analytical function, since a network consists solely of matrix multiplication operations and non-linear simple functions. This property makes a neural network as numerical method extremely attractive.
- Because we know the input to the neural network, these are in fact the parameters for the stochastic model, risk models related to the stochastic models remain untouched [5].

In this thesis, we primarily focus on using neural networks as numerical methods, which we use to approximate the solution to a stochastic model. In chapter 5 we construct a model in which we employ a neural network to approximate the function satisfying a stochastic differential equation.

4.2. Fundamentals

In this section we discuss the components that embody a neural network. One will see that elementary concepts from mathematics are used to create a network, allowing for great flexibility. Beginning with a perceptron, the smallest building block in a network, we work our way up to a layer of perceptrons.

4.2.1. Perceptron

The perceptron is the fundamental building block of any neural network. Over the years, many different configurations of neural networks have been developed. Most, if not all, of these network are using perceptrons in some way. A perceptron can be seen as a neuron in the brain. A neuron is a specialised cell designed to transmit information to other cells. Similarly, the perceptron takes an input, performs an arbitrary calculation and returns an output. In mathematics, a perceptron can be seen as a function taking an input $\vec{x}$ and performing an operation g on it, outputting $g(\vec{x})$.

A perceptron can have multiple inputs. Each input is multiplied by its corresponding weight. All inputs are summed and passed into the *activation function*, a function which determines what the output of the perceptron should be. Figure 4.1 gives a visual representation of this process. Each of the weights give a certain relevance to their corresponding input variable. For instance, let x_i be a very relevant input variable. Its weight should be higher than a more irrelevant variable x_j. The operations in a single perceptron can be denoted as

$$\hat{y} = f\left(\sum_{i=0}^{n} w_i x_i\right). \tag{4.2}$$

where $\hat{y}$ is the output, $\vec{x}$ the input, $\vec{w}$ the weights and f the activation function.

In addition to the weights $\vec{w}$, we introduce the bias b, another set of parameters which can be updated and trained. The bias is added to the input product, serving as an intercept. This allows the

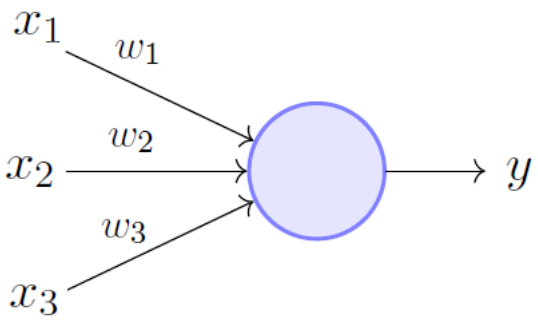

Figure 4.1: A visual representation of a perceptron. Adopted from [35].

perceptron to represent any affine transformation, resulting in a better ability to fit the target function. Including the bias, the complete operation for a perceptron becomes

$$\hat{y} = f\left(\sum_{i=0}^{n} w_i x_i + b\right) = f\left(\vec{w}^T \vec{x} + b\right). \tag{4.3}$$

In the remainder this thesis we use the terms perceptrons and nodes interchangibly, as this naming aligns best with existing work.

4.2.2. Activation Function

An activation function in a perceptron can be seen as the rate of action potential firing in a perceptron. In theory, any function can serve as an activation function. For instance, we could take f to be the Heaviside step function

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}. \tag{4.4}$$

In this case, the perceptron will output 0 when $\sum_{i=0}^{n} w_i x_i + b < 0$ and 1 else. However, there are a two requirements that a function should satisfy in order to be useful as an activation in perceptrons.

Most importantly, the activation function should be *non-linear*. This means that the output cannot be reproduced from a linear combination of inputs, which in turn allows the output to depend non-linearly on its inputs. Many problems that we attempt to solve with neural networks do not have a linear solution. The introduction of non-linear activation functions theoretically enables a neural network to estimate any function. Secondly, the activation function should be differentiable, preferably on its entire domain. As we will see in section 4.2.3, we train the network by defining a loss function which we must minimise. We minimise the loss function by updating the weights and bias. Many present-day training utilities for neural networks use *stochastic gradient descent* for this process. This technique requires us to compute the derivative of the loss with respect to the network parameters. If an activation function is not differentiable, such a derivative will not be defined, preventing the updating of the network and therefore the convergence.

We present a few popular activation functions which we use throughout the development of various neural networks. Each of the activation functions has its benefits in certain scenario's and drawbacks in others.

- **Sigmoid function**. The sigmoid function is a differentiable function bounded on $(0, 1)$.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.5}$$

  A sigmoid essentially 'squashes' its input to a value between 0 and 1, making it a useful function to output a Bernoulli probability. However, when back propagating through previous layers of the neural network the sigmoid can cause the gradient to converge to zero way faster than desirable,

especially when the network has many layers. This is due to the fact that the derivative of the sigmoid, defined as $\sigma(x)(1-\sigma(x))$, is always smaller than 1. Repeated multiplication of this derivative when performing gradient descent causes the gradient to get very small. When the gradient is very small, the layers will not be updated properly, causing the network to be unable to learn. This issue is called the *vanishing gradient problem*.

- **Hyperbolic Tangent (tanh)**. The hyperbolic tangent is very similar to the sigmoid, except it's range is $(-1, 1)$. It is therefore able to output negative values, indicating inverse relationships. Similarly to the sigmoid, it is prone to the vanishing gradient issue.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.6}$$

- **Rectified Linear Unit (ReLU)**. The rectified linear unit is a simple non-linear unbounded function that is very easy to compute. Its derivative for $x > 0$ is always 1, which makes it immune to the vanishing or *exploding gradient problem*, in which the latter is where the gradient blows up through repetitive multiplication with a value larger than 1.

$$ReLU(x) = \max\{0, x\} \tag{4.7}$$

  Theoretically, the downside about this activation function is that it does not have a derivative at 0. In practise, however, this does not seem to be an issue.

- **Gaussian Error Linear Units (GELU)**. The Gaussian error linear unit is a relatively new activation function, which weights inputs by their value, rather than gates inputs by their sign as in ReLUs [12]. The GELU activation function contains the CDF of the standard normal distribution, weighted by the input value, i.e.

$$GELU(x) = x\Phi(x), \tag{4.8}$$

  with Φ the CDF of the standard normal distribution. The authors show with empirical tests that this activation function performs particularly well on natural language processing and computer vision tasks.

- **Softmax**. The softmax function is a function that takes a vector as input and normalises it into a probability distribution. The softmax function is nearly identical to the *Boltzmann distribution* which is used in statistical thermodynamics to give the probability that a system will be in a certain state given the state energy and the temperature of the system. The Boltzmann distribution is expressed in the form:

$$p_i \propto e^{-\frac{\epsilon_i}{k_B T}}, \tag{4.9}$$

  where p_i is the probability that the system is in state i, ϵ_i the energy of state i, k_B the Boltzmann constant and T the thermodynamic temperature. The thermodynamic temperature is omitted in the softmax function, but the temperature still has an intuitive meaning. For instance, high temperature in thermodynamics causes particles to have more accessible energy states, which reduces the probability of the atom being in some state i. During training, we essentially attempt to lower the temperature, comparable to a condensation process, in which we reduce the energy in the system and therefore raise the probability of a particle to be in a certain state. Hence, in the course of a successful training process, the probability distribution associated with any given sample becomes more and more articulate.
  The softmax function is given below.

$$\sigma(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{4.10}$$

  The additional benefit of using the softmax function over other normalisation functions is that the softmax function can calculate a probability distribution over input vectors with negative components. We typically use this activation function in the final layer of the network, in order to output a probability distribution of classes. We will generally not be using this function, as all the problems considered in this thesis are regression problems. In the case of classification, this activation function should be used on the final node.

4.2.3. Stochastic Gradient Descent & Back Propagation

In the previous sections we introduced a perceptron, which included parameters. These parameters must be updated in order to provide reasonable estimates of the target variable. In order to update the parameters of the perceptron, we need a measure that quantifies the distance from the output of the perceptron compared to the desired output. Using the distance between the network output and the actual output as error measure, we can update the weights in order to minimise this error. We refer to the distance as the loss, and call the function we minimise the loss function.

$$J(\theta) = \|f(\vec{x};\theta) - \hat{y}\|, \tag{4.11}$$

where y is the target or desired output, and $f(\vec{x};\theta)$ the actual output of the perceptron given input $\vec{x}$ and parameters θ. In theory, we calculate the loss for each sample that we pass into the network. We then alter parameters in the neural network in order to decrease the loss for the consecutive samples. In practise, we calculate the loss per batch. A batch is a collection of multiple samples. Calculating the loss and the gradient on batches improves training speed and reduces variance.

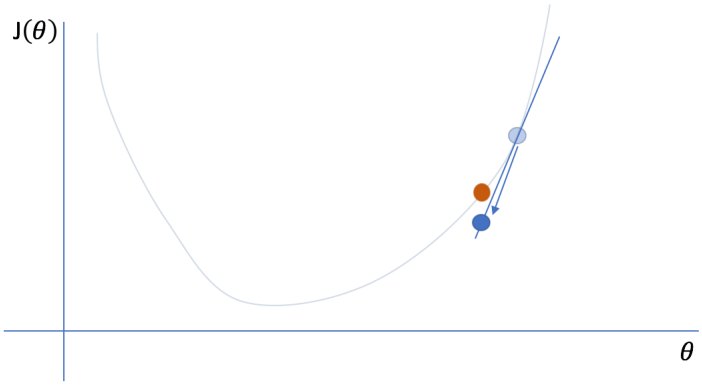

Figure 4.2: A visual representation of one step in the gradient descent process. Adopted from [21].

Consider the following convex function $J(\theta)$ illustrated in figure 4.2, where at the start of a step in the gradient descent process, our location is the light blue dot. The objective is to find the the minimum of this function, since J is the loss function which we want to minimise. At each step, we calculate the gradient of the loss function (equation 4.11). We then move a factor α in the opposite direction of the gradient, ending up at the dark blue dot. The factor α is called the *learning rate*. Depending on this factor, we can descent faster or slower, depending on the landscape of the loss function. In terms of the parameters θ, we have the following iterative updating formula

$$\theta := \theta - \alpha \nabla_\theta J(\theta). \tag{4.12}$$

Suppose that we chain two or more perceptrons after each other, where the output of perceptron i is the input of perceptron $i+1$. When computing the derivative of the loss with respect to the parameters of the first perceptron, we must back propagate through the consecutive perceptrons. We do this using the chain rule. Because all the perceptrons make use of elementary activation functions with a closed-form derivative, back propagation across all the perceptrons in a network is feasible.

4.2.4. Layers

Until now, we have only discussed operations within a single perceptron. The single perceptron itself is unable to approximate complex functions due to its simple configuration. By increasing the amount of perceptrons and chaining them together in various ways, we obtain a network of perceptrons, allowing for great complexity and abilities to approximate many functions. To slightly standardise network configuration, we introduce a *layer*, simply referring to a stack of perceptrons which generally all depend on the same previous input. This input could be the input data or output from previous layers. Note

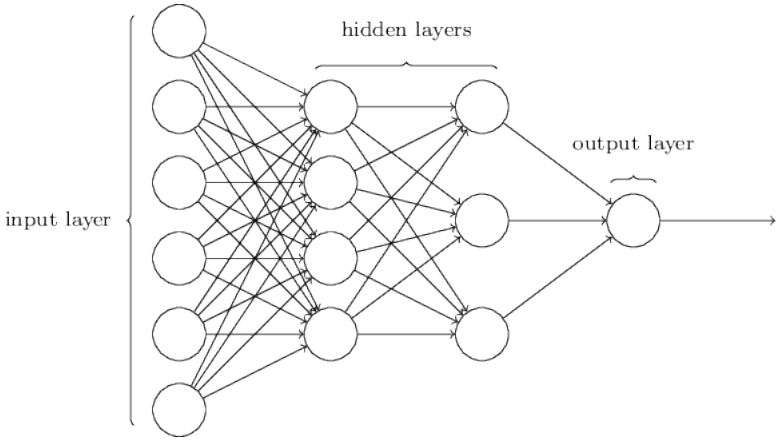

Figure 4.3: A visual representation of a multilayer perceptron. Adopted from [9].

that the flow of information is one way, meaning that we treat the network as a *directed acyclic graph (DAG)*.

When using the notion of layers, we can have many perceptrons in one layer being connected to many perceptrons in the consecutive layer. An example is illustrated in figure 4.3, where the first two layers are densely connected. These types of layers are called *dense layers*. Say that the first layer has m perceptrons, and the second layer has n. We then denote the connection between perceptron i and j as W_{ij}, where W is an $m \times n$ matrix. If such a connection exist, the value W_{ij} is the weight that perceptron i contributes to the input of perceptron j. If there is no connection between i and j, $W_{ij} = 0$. Analogously to the case in which we have a single perceptron, the task of the neural network is to learn the matrix W and bias $\vec{b}$ using gradient descent. Using the matrix W, we can write (4.3) as

$$y_i = f_i\left(\sum_j W_{ij}x_j + b_j\right), \tag{4.13}$$

where f_i is the activation function of perceptron i, and y_i the output of perceptron i. We generally use the same activation function for each perceptron in a layer, in which case $f_j = f_i$ for all $i, j \in \{1, \dots, n\}$.

Possibly the most vanilla type of network one can encounter is a network in which each layer is densely connected to only the previous and consecutive layer, yielding a visual representation as in figure 4.3. This network is called a *multilayer perceptron (MLP)*. By varying the size of the layers, i.e. the amount of nodes inside it, and the amount of consecutively placed layers, we can increase or decrease the size of the network. In section 4.3 we create more complex networks which could be better suited for certain problems. To compare performance of these networks to some benchmark, we use a MLP, as this is the most basic type of network.

4.2.5. Initialisation

When designing networks with many layers, special care should be taken to avoid vanishing gradients. The gradient is calculated using the chain rule when back propagating through the network layers. If the network contains n layers, we must multiply the derivative of the activation functions per layer n times, which can cause the gradient to converge to 0 when n is large. However, a larger gradient is required to update the weights appropriately, improving the performance of the network in the consecutive iteration. This can be seen directly from equation 4.12, in which the parameters are not updated into the gradient direction when $\nabla_\theta J(\theta) = 0$.

One thing we can do to avoid vanishing gradients is initialising the parameters in the network in a suitable way. The current standard for initialisation is Glorot initialisation [39]. In this section, we examine another initialisation technique, empirically shown to perform better on deep networks that are using $ReLU(x) = \max\{0, x\}$ activation functions [23].

Forward Propagation
When selecting an appropriate distribution to sample the initial values for the parameters from, the aim is to keep the variance of the outputs in each layer constant. Consider a network with L layers. For $l \in L$ we have

$$\vec{y}_l = W_l \vec{x}_l + \vec{b}_l, \tag{4.14}$$

And $\vec{x}_l = f(\vec{y}_{l-1})$. In this example, W_l is the weight matrix for layer l, $\vec{x}_l$ the input produced by a non-linear function of the output from the previous layer, and $\vec{b}_l$ the bias vector. We consider the initialised parameters in W_l to be $i.i.d.$ and assume that $\vec{x}_l$ also has $i.i.d.$ elements (we note that this assumption may not hold in general). Then we have

$$Var[y_l] = n_l Var[w_l x_l], \tag{4.15}$$

where y_l, x_l, w_l are the elements in the vectors and n_l the number of connections of output from layer l. We let $E[w_l] = 0$, then

$$Var[y_l] = n_l Var[w_l] E[x_l^2]. \tag{4.16}$$

Note that when we take $ReLU$ activation, $x_l = \max\{0, y_{l-1}\}$ does not have $E[x_l] = 0$, and therefore $E[x_l^2] \neq Var[x_l]$. In [39] the assumption of equality is made, leading to a different distribution for the initial parameters. Let w_{l-1} have a symmetric distribution with $E[w_{l-1}] = 0$, and set $b_{l-1} = 0$, we have that y_l has the same distribution as w_{l-1}, and

$$E[x_l^2] = \frac{1}{2} Var[y_{l-1}] \tag{4.17}$$

when f is $ReLU$. Substitution of (4.17) into (4.16) yields

$$Var[y_l] = \frac{1}{2} n_l Var[w_l] Var[y_{l-1}]. \tag{4.18}$$

For L layers we have

$$Var[y_L] = Var[y_1] \left(\prod_{l=2}^{L} \frac{1}{2} n_l Var[w_l]. \right) \tag{4.19}$$

Since we want the variance of the output to remain constant with respect to the input, a sensible condition would be

$$\frac{1}{2} n_l Var[w_l] = 1, \qquad \forall l \in L. \tag{4.20}$$

Backward Propagation
Similar to the case of forward propagation, the aim is to keep the variance of the output constant over the layers. Let J be the loss function, we have for layer l

$$\frac{\partial J}{\partial \vec{x}} = \hat{W}_l \frac{\partial J}{\partial \vec{y}}. \tag{4.21}$$

Again, we assume that w_l, $\frac{\partial J}{\partial y_l}$ are independent and w_l are sampled from a symmetric distribution around 0, leading to $E[\frac{\partial J}{\partial \vec{x}}] = 0$. In the case of $ReLU$, we have that

$$f'(y_l) = \begin{cases} 0 & y_l < 0 \\ 1 & y_l > 0. \end{cases} \tag{4.22}$$

Assuming that $f'(y_l)$ and $\frac{\partial J}{\partial x_{l+1}}$ are independent, we have

$$E\left[\frac{\partial J}{\partial \vec{y}_l}\right] = \frac{E\left[\frac{\partial J}{\partial \vec{x}_{l+1}}\right]}{2} = 0 \tag{4.23}$$

$$E\left[(\frac{\partial J}{\partial \vec{y}_l})^2\right] = Var\left[\frac{\partial J}{\partial \vec{y}_l}\right] = \frac{1}{2} Var\left[\frac{\partial J}{\partial \vec{y}_l}\right]. \tag{4.24}$$

Together the variance of the gradient in (4.21) becomes

$$Var\Big[\frac{\partial J}{\partial \vec{x}_l}\Big] = \hat{n}_l Var\Big[w_l\Big] Var\Big[\frac{\partial J}{\partial \vec{y}_l}\Big] \tag{4.25}$$

$$= \frac{1}{2}\hat{n}_l Var\Big[w_l\Big] Var\Big[\frac{\partial J}{\partial \vec{x}_{l+1}}\Big], \tag{4.26}$$

where $\hat{n}_l$ denotes here the number of connections in the input of layer l. For L layers

$$Var\Big[\frac{\partial J}{\partial \vec{x}_2}\Big] = Var\Big[\frac{\partial J}{\partial \vec{x}_{L+1}}\Big]\left(\prod_{l=2}^{L}\frac{1}{2}\hat{n}_l Var\Big[w_l\Big]\right). \tag{4.27}$$

We find that a sensible condition would be

$$\frac{1}{2}\hat{n}_l Var[w_l] = 1 \qquad \forall l \in L. \tag{4.28}$$

When assuming that the number of connections throughout layers remain close to each other, we can see that sampling the initial parameters should be done from a zero mean symmetric distribution with variance $2/\hat{n}_l$. In theory, any distribution that satisfies this two criteria is an eligible candidate to sample from. In practise, however, [23] primarily use a Gaussian distribution. A uniform distribution is mentioned as well, but not further used. Hence, when using $ReLU$ activations in deep networks, we sample

$$w_l \sim \mathcal{N}\left(0, \frac{2}{\hat{n}_l}\right) \qquad \forall l \in L. \tag{4.29}$$

In the remainder of this thesis, we will refer to this initialisation technique as the He initialiser [23].

4.2.6. Batch Normalisation

In addition to sampling the initial parameters of the network appropriately (section 4.2.5), we examine another tool which helps keeping the distribution of the input consistent over the consecutive layers while training. This technique is called *batch normalisation* [32].

It has been shown that a network converges faster if its inputs are transformed to have zero mean and unit variance [34]. Let $X = (\vec{x}_1, \dots, \vec{x}_d)$ be a batch of input data to a layer with d dimensions. We normalise each dimension separately

$$\hat{x}_k = \frac{x_k - E[x_k]}{\sqrt{Var\Big[x_k\Big]}}. \tag{4.30}$$

Note that the expectation and the variance of x_k are computed over the batch. Also, since we do not normalise the joint distribution of X, we do not decorrelate the inputs. However, doing so is costly and could lead to back propagation errors due to non differentiable functions. When normalising the input of a layer, we change the range of the activation function. To avoid this, we introduce two additional parameters which can scale and shift the normalised input, and set the output of the batch normalisation process to be

$$y_k = \gamma_k \hat{x}_k + \beta_k. \tag{4.31}$$

In this case, y_k can again assume values in the entire domain. The parameters γ, β are to be learned by the network. The operations in the batch normalisation layer with an input batch of $x_1, \dots, x_n$ then becomes

$$\mu \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.32}$$

$$\sigma^2 \leftarrow \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{4.33}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu}{\sqrt{\sigma^2}} \tag{4.34}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta. \tag{4.35}$$

4.3. Network Variations

In this section we consider variations on the classical MLP network discussed in 4.2.4. The goal is to find a network architecture well suited for approximating partial differential equations and implied volatilities. To test the performance of each network architecture, we train various networks in a supervised learning way on Black-Scholes European call option data, Heston European call option data and the implied volatility of call options. The results of these comparisons are presented in 4.4.

A typical MLP generally consists of L layers, where in each layer $l \in L$ a weight matrix is multiplied with the input, a bias vector is added and a non-linear function H is applied to the result. In order to remain consistent with the notation in [31], we write in the remainder of this section

$$\vec{y} = H(\vec{x}, W_H) = H(W_H\vec{x} + \vec{b}_H), \tag{4.36}$$

omitting the bias vector for readability.

4.3.1. Residual Network

The first network architecture we consider, originating from [24], is a slight variation of the classical MLP. This network architecture consists of layers with the transformation

$$\vec{y} = H(\vec{x}, W_H) + \vec{x}, \tag{4.37}$$

where we add the input to the layer to the non-linearity. A visual representation of this layer is given in figure 4.4.

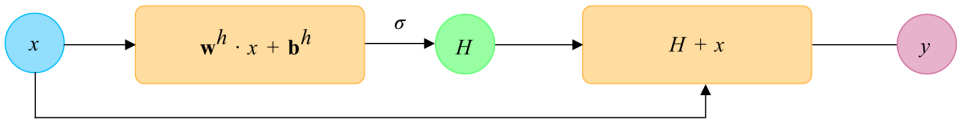

Figure 4.4: Schematic of computations within a single residual layer as presented in equation (4.37). We denote with · element wise multiplication of the vectors.

We provide an additional 'channel' which leaves the input vector to the layer $\vec{x}$ unaffected, allowing for input vector $\vec{x}$ to flow through the layer, by adding it to the non-linear computation H in the layer. The addition of this operation is minor, but can have large influences on the MSE as we will see in 4.4. The authors of [24] claim that this network has been successful with convolutions in particular.

4.3.2. Highway Network

We can generalise the residual network architecture from 4.3.1 by introducing an additional non-linearity $T(\vec{x}, W_T)$. This non-linearity will determine the amount of information flowing from the non-linearity H inside the layer and the amount of information 'carried' over from the input vector $\vec{x}$. We obtain the transformation

$$\vec{y} = H(\vec{x}, W_H) \cdot T(\vec{x}, W_T) + \vec{x} \cdot \Big[1 - T(\vec{x}, W_T)\Big]. \tag{4.38}$$

The role of T is to act as a convex combination between the input $\vec{x}$ and transformed input $H(\vec{x}, W_H)$. A visual representation of this layer is given in figure 4.5.

When approximating complex functions using neural networks, the depth of the network is proven to be a significant factor in the success of a network [28]. However, training deeper networks presents additional bottlenecks, such as vanishing gradients. The intuition behind the highway networks is to allow for unimpeded information flow across the layers [31].

From equation 4.38 we can see that for the boundary values of T,

$$\vec{y} = \begin{cases} \vec{x}, & T(\vec{x}, W_T) = \vec{0} \\ H(\vec{x}, W_H), & T(\vec{x}, W_T) = \vec{1}, \end{cases} \tag{4.39}$$

and consequently

$$\frac{d\vec{y}}{d\vec{x}} = \begin{cases} I, & T(\vec{x}, W_T) = \vec{0} \\ H'(\vec{x}, W_H), & T(\vec{x}, W_T) = \vec{1}. \end{cases} \tag{4.40}$$

Thus, the transform gate $T(\cdot)$ allows the highway layer to vary its behaviour between a standard MLP layer and that of an identity mapping, leaving $\vec{x}$ untouched.

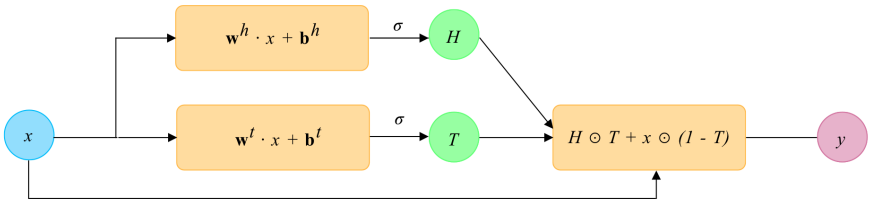

Figure 4.5: Schematic of computations within a single highway layer as presented in equation 4.38. We can see that this layer only consists of 2 non-linear activation functions, whereas the generalised highway layer has an additional non-linear transformation, weight matrix and bias vector. We denote with $\cdot$ element wise multiplication of the vectors.

4.3.3. Generalised Highway Network

Once more, we can generalise the highway network by removing the convex combination constraint. To this end, we define an additional non-linear transformation alongside $T(\vec{x}, W_T)$, which we call $C(\vec{x}, W_C)$, such that

$$\vec{y} = H(\vec{x}, W_H) \cdot T(\vec{x}, W_T) + \vec{x} \cdot C(\vec{x}, W_C). \tag{4.41}$$

We call T and C the transform and carry gate, respectively, as they regulate the amount of information passed into the next layer from the transformed and original input. This gating mechanism allows for information to flow along the layers of the network without attenuation. Figure 4.6 illustrates the layer transformations.

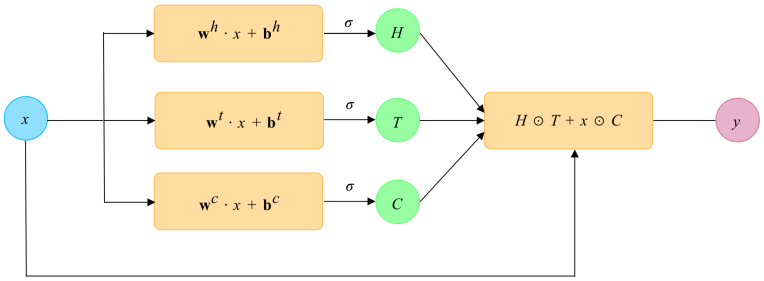

Figure 4.6: Schematic of computations within a single generalised highway layer as presented in equation 4.41. This layer is a generalised version of the highway layer (figure 4.5).

This is a generalised case of the highway layer (equation 4.41), in which we set $C = 1 - T$. In the remainder of this thesis, we refer to the general case where we have both T, C as the generalised highway network, and the specific case where $C = 1 - T$ as the highway network.

We create versions of each of the mentioned networks, and compare their performances. Specifically, we implement the general highway network consisting of layers with operations defined in equation 4.41, the highway network considered in [31] (equation 4.38) and the residual network (equation 4.37). An instance of each of the networks will be trained and compared against other networks.

4.3.4. DGM Network

The next network we are interested in is a relatively new type of network architecture, developed by [20] and further expanded on in [1], called the Deep Galerkin Method (DGM). The authors of [20] argue that this network architecture allows for 'sharp turns' in the target function, which can occur near the boundary and terminal condition of a PDE. This property would make the network suitable for the approximation of PDE's as discussed in chapter 5. In the following sections, we develop variations on this network architecture to find what could cause the flexibility of this architecture, and attempt to find potential improvements.

The architecture of a DGM network is similar to that of highway networks (section 4.3.2), in that we have transform and carry gates determining how much information from the previous layer should be used. The DGM network consists of an arbitrary amount of *layers*, which we will refer to as *DGM layers*.

DGM Layer

Within a DGM layer, several operations are executed. A schematic view of a single layer is given in figure 4.7.

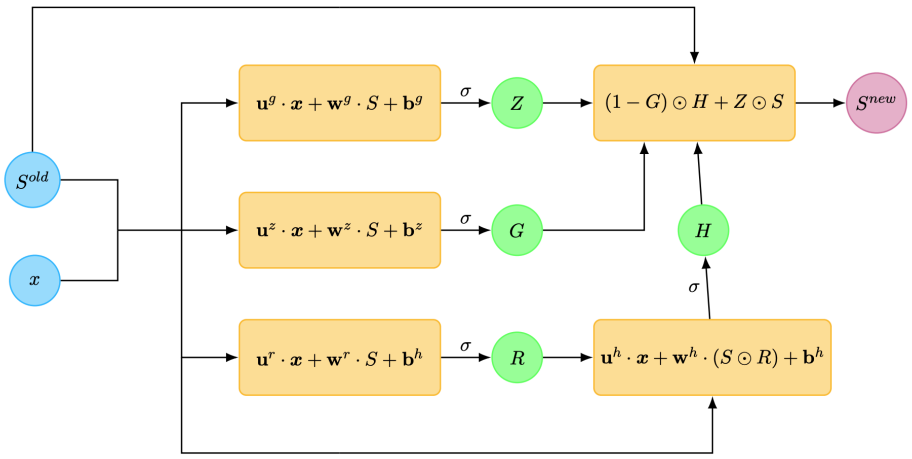

Figure 4.7: Schematic of computations within a single DGM layer. Adopted from [1].

In contrary to a highway layer, the DGM layer has two inputs, denoted in blue. S^{old} is the output from the previous layer. This can be either the first non-linear transformation S^1 or the output from a DGM layer (figure 4.8). With $\vec{x}$ we denote the original input vector, i.e. the untouched feature data. We use both layer inputs to create 4 vectors: Z, G, R and H. Using the vectors, we compute S^{new}, the output of the DGM layer. Consider the final operation in the layer

$$S^{new} = (1 - G) \cdot H + Z \cdot S, \tag{4.42}$$

where we denote with $\cdot$ element wise multiplication of two vectors with equal dimensionality. Notice that $\mathbf{w} \cdot S$ in figure 4.7 denotes matrix-vector multiplication, since $\mathbf{w}$ is a weight matrix. If we set $T = 1 - G$, $C = Z$ and denote with S the input to a highway layer, we have a similar computation in a highway layer

$$S^{new} = T \cdot H + C \cdot S. \tag{4.43}$$

where H denotes the transformed input inside the layer and S the output from the previous layer, denotes with $\vec{x}$ in the highway layer (section 4.3.2). The DGM network adds additional complexity compared to the highway layer in two ways:

1. Instead of computing H through one non-linear transformation as in a highway layer, we compute H using two non-linear transformations, where the output of the first is the input of the second. This can be seen as a subnetwork inside the DGM layer. In section 4.3.5 we exploit this property, by adding more non-linear transformations as a subnetwork to measure the relevance of this double transformation.

2. In each non-linear transformation for the 4 vectors, the original feature vector $\vec{x}$ is incorporated. This leads to a recurrent architecture that we also see in recurrent networks [3]. In section 4.3.6 we create a DGM architecture without this recurrence, and train it alongside the DGM to measure the impact of this recurrent architecture inside the layers.

Additionally, we note the following results of the DGM network architecture

- As argued in [20], the incorporation of repeated element wise multiplication of the nonlinear functions is useful in capturing sharp turns which can be present in complicated functions. Because of the similarity of the DGM layer and the highway layer, we have that the input $\vec{x}$ enters into the calculations of each intermediate step, reducing the probability of vanishing gradients inside the back propagation phase of the training cycle.
- Due to the addition of weight matrices and computations, a DGM layer contains roughly eight times as many parameters as a layer in a MLP. Additionally, four times as many activation functions are present in the network.

Overall DGM Architecture
Since the DGM layer that we examined in section 4.3.4 requires both $\vec{x}$ and some transformation S as input, we must compute the transformation S from the input before the layer. Additionally, we incorporate a final linear transformation to obtain $\vec{y}$.

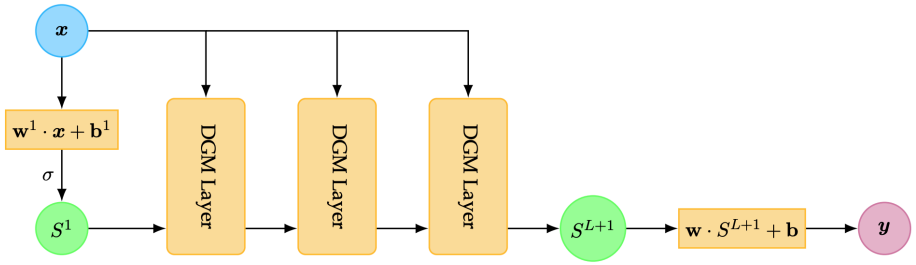

Figure 4.8: A high level overview of the DGM network. Each *DGM Layer* refers to the operations visualized in figure 4.7. Adopted from [1].

Figure 4.8 shows an overview of the DGM network. This is again similar to a highway network. Equation 4.41 requires the dimensionality of $\vec{x}, \vec{y}, H(\vec{x}, W_H)$ and $T(\vec{x}, W_T)$ to be equal. The result of this equality is that we cannot change the amount of nodes over chained highway layers. We must therefore use a dense layer to transform the input dimension to the dimension of the highway layers we want to use. A similar computation happens in the DGM architecture where we first compute

$$S^1 = \mathbf{w}^1 \cdot \vec{x} + \mathbf{b}^1, \tag{4.44}$$

to regulate the dimension of S^1.

4.3.5. Deep DGM Network

In this section we consider a variant of the DGM network with multiple nonlinear operations to compute H. A visualisation of the network is shown in figure 4.9. In this network, we can vary the number of nonlinear operations performed on R to obtain H as a hyperparameter. If we choose $n = 5$, this means that 5 additional weight matrices for both $\mathbf{w}$, $\mathbf{u}$ and $\vec{b}$ are required, for a total of 10 additional weight matrices and 5 bias vectors and non-linear functions. A deep DGM layer is therefore considerably more complex than the standard DGM layer. The goal of this network is to quantify the influence of deeper non-linearities within a DGM layer.

4.3.6. No-Recurrence DGM Network

The DGM network in which we omit dependence of $\vec{x}$ in all the layers other than the input layer considered is a simplified version of the DGM network. In figure 4.10 the schematic of this network is given.

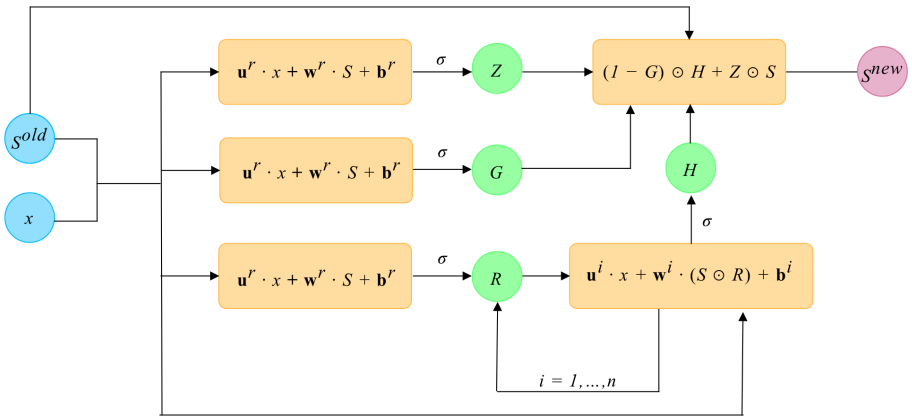

Figure 4.9: A schematic of the deep DGM layer. In each layer, we have n transformations

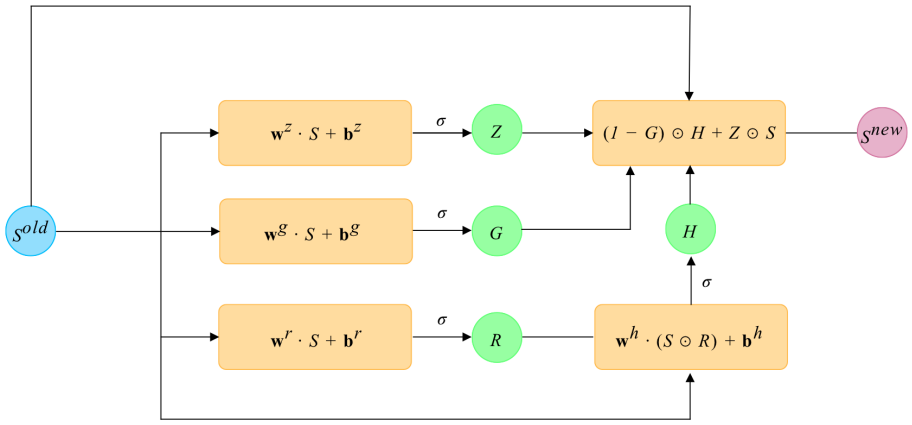

Figure 4.10: A schematic of the no-recurrence DGM network. This network is a simplified version of figure 4.7.

Notice that none of the operations depend on $\vec{x}$, greatly reducing the amount of parameters each layer contains as we do not require weight matrices $\mathbf{w}$ for all the layer dependencies on $\vec{x}$.

The structure of this layer closely resembles a highway layer (section 4.3.2). Indeed, the only notable difference is the way we compute H:

$$H = \sigma\Big(\mathbf{w}^h \cdot \Big[S \cdot \sigma\left(\mathbf{w}^r \cdot S + \mathbf{b}^r\right)\Big] + \mathbf{b}^h\Big), \tag{4.45}$$

whereas we have the following operation for H in the highway layer

$$H = \sigma\left(\mathbf{w}^r \cdot S + \mathbf{b}^r\right). \tag{4.46}$$

4.4. Network Analysis

In this section, we train the network architectures discussed in section 4.3 and apply them on three different problems (section 4.4.1). We then compare the performance of the networks against each other, with the aim to find an optimal network architecture to estimate solutions to these problems. All three problems originate from [25], in which only one network was considered. The implementation for every network was done using Google's *TensorFlow* library for Python [4].

4.4.1. Problem Descriptions

Black-Scholes European Call Option
We consider the results in [19], in which a DGM network is compared against classical MLP networks on approximating the one dimensional Black-Scholes PDE (equation 3.14) which has the analytical solution for a call option derived in equation 3.27. We define the following variables which we sample:

- $m = \frac{S}{K}$: The underlying price divided by the strike price. This is called the moneyness value.
- σ: The volatility of the underlying.
- r: The risk-free rate.
- $\tau = T - t$: A time transformation to obtain the time to maturity in years.

Using these variables we can calculate the solution using equation 3.27 and the transformation

$$V(\tau, m) = \frac{V(t,S)}{K} = m\Phi(d_1) - e^{-r\tau}\Phi(d_2), \tag{4.47}$$

$$d_2 = \frac{\log(m) + (r \pm \frac{1}{2}\sigma^2)\tau}{\sigma\sqrt{\tau}}, d_1 = d_2 + \sigma\tau. \tag{4.48}$$

For the Black-Scholes European call option problem, we train the networks to find the solution presented in equation 4.47. In the remainder of this thesis, we will refer to this problem as the Black-Scholes pricing problem.

Heston European Call Option
In addition to the Black-Scholes pricing problem, we also train networks to approximate the solution to the Heston PDE (equation 3.62). This problem is similar to the Black-Scholes pricing problem in that both approximations are solutions to an option pricing PDE. The difference is that the Heston PDE is a slightly more complex PDE in which the volatility is considered to be stochastic. Unlike the Black-Scholes PDE, the Heston PDE has no analytical solution, and approximation of the solution will be done using the COS method (section 3.4.1). We train the network architectures on this problem in addition to the Black-Scholes pricing problem to determine if the networks show consistent performance across pricing PDE's. In the remainder of this thesis, we refer to this problem as the Heston pricing problem.

Implied Volatility
For the third problem which we train and evaluate the networks on, we continue the analysis originating from [25], in which only one MLP is trained and evaluated on the implied volatility problem discussed in (3.3). This problem is related to pricing options because in finance we are often interested in quoting the option in terms of its implied volatility. Ideally, we want not only to price an option using neural networks, but also recover its implied volatility surface. This problem has no analytical solution, and the current industry standard is to apply iterative methods discussed in section 3.3.

In addition to the standard implied volatility dataset, in which we use the parameters from the directly inverted Black-Scholes pricing problem (table 4.3), we introduce a transformation on the dataset from the scaled call price (V/K) to the scaled time value of the option. The reasoning behind this transformation is the following: We train the networks to find the solution to equation 3.48. In section 3.3.2 we argued that the option's vega, i.e. its derivate with respect to the volatility, can become arbitrarily small for deep ITM or OTM options. In the context of neural networks, where we take the derivatives to compute the gradient of the network, the instability of vega may lead to large gradients, possibly causing significant prediction errors.

We briefly elaborate on this issue. For convenience, we write $V_c = f(\sigma)$, where f is the analytical solution to a one dimensional European Black-Scholes call option, depending on σ. The implied volatility problem concentrates on approximating the inverse of this problem, i.e. $\sigma = f^{-1}(V_c)$. During the back propagation phase, we must compute the gradient of $f^{-1}(V_c)$ with respect to V_c. That is, $\frac{d\sigma}{dV_c} = \frac{1}{dV_c/d\sigma}$, which is the reciprocal of vega. For arbitrarily small values of vega, this gradient will explode, leading to convergence problems.

We follow the gradient-squash approach from [25] to mitigate this problem, which motivate next.

An option can be divided into its *intrinsic value* as well as its *time value*. To obtain the time value, we subtract the intrinsic value (i.e. the no-arbitrage bound) from the option, yielding

$$\hat{V} = V - \max\{S - Ke^{-r\tau}, 0\}, \tag{4.49}$$

with $\hat{V}$ the time value. Finally, we apply a $\log$-transform to furthermore reduce the possible steepness of the gradient, resulting in the dataset shown in table 4.4.

Data Sampling

We treat each problem discussed above as a *supervised learning* problem. To this end, we require inputs and outputs which we can feed the network and calculate its loss on. In this section, we follow the configuration explained in [25] for all three problems. We generate a total of 1 million samples for each problem, which we split into a $80/20$ train and validation set. We generate an additional $100,000$ samples used for evaluation. When sampling from the space of input parameters, we can either define a joint distribution over the entire domain, or sample each variable separately. Both [19, 25] opt for latin hypercube sampling (LHS), in which values are sampled from a joint distribution, resulting in a better representation of the parameter space [27]. Thus, to stay consistent with previous research, we use LHS in this thesis to generate the data as well.

For the Black-Scholes pricing problem, we sample the parameters shown in table 4.1, and use the analytical solution (equation 4.47) to obtain the output labels.

	Parameters	Range
Input	Moneyness: S_0/K	$[0.4, 1.6]$
	Time to Maturity ($\tau = T - t$)	$[0.2, 1.1]$
	Riskfree rate (r)	$[0.02, 0.1]$
	Volatility (σ)	$[0.01, 1.0]$
Output	Scaled call price (V/K)	$(0.0, 0.9)$

Table 4.1: Black-Scholes data generating parameters, adopted from [25].

We use the same sampling parameters as in [19, 25], allowing us to compare the network performances to results in both works at a later stage.

For the Heston pricing problem, we require additional parameters, listed in table 4.2. We compute the call price using the COS method (section 3.4.1).

	Parameters	Range
Input	Moneyness: S_0/K	$[0.4, 1.6]$
	Time to Maturity ($\tau = T - t$)	$[0.2, 1.1]$
	Riskfree rate (r)	$[0.02, 0.1]$
	Correlation (ρ)	$[-0.95, 0.0]$
	Reversion speed (κ)	$[0, 2.0]$
	Long average variance ($\bar{v}$)	$[0, 0.5]$
	Volatility of volatility (γ)	$[0, 0.5]$
	Initial variance (v_0)	$[0.05, 0.5]$
Output	Call price (V)	$(0.0, 0.67)$

Table 4.2: Heston data generating parameters, adopted from [25].

For the default implied volatility problem, we re use the Black-Scholes pricing data and switch the volatility (σ) and scaled call price (V/K), obtaining the parameters listed in table 4.3.

	Parameters	Range
Input	Moneyness: S_0/K	$[0.4, 1.6]$
	Time to Maturity ($\tau = T - t$)	$[0.2, 1.1]$
	Riskfree rate (r)	$[0.02, 0.1]$
	Scaled call price (V/K)	$[0, 0.71]$
Output	Volatility (σ)	$(0.05, 1.0)$

Table 4.3: Implied volatility data generating parameters.

Finally, we transform the implied volatility dataset by applying the transformation described in equation 4.49, along with a $\log$ transform. The domain of the transformed implied volatility dataset where we used the scaled time value is given in table 4.4.

	Parameters	Range
Input	Moneyness: S_0/K	$[0.4, 1.6]$
	Time to Maturity ($\tau = T - t$)	$[0.2, 1.1]$
	Riskfree rate (r)	$[0.02, 0.1]$
	Scaled time value ($\log \hat{V}/K$)	$[-18.42, -0.95]$
Output	Volatility (σ)	$(0.01, 1.0)$

Table 4.4: Transformed implied volatility data generating parameters.

4.4.2. Multilayer Perceptron

In this section, we train a set of MLP's ranging from a very small network to a large network with similar configuration. The goal is to evaluate the performance of an MLP network architecture, as well as determining if the size of the network influences the performance over a fixed range of epochs. To this end, we define a set of twelve networks, with configuration listed in table 4.5. We fix the learning rate according to the optimal value found from the study conducted in [25]. The batch size is chosen as large as possible while keeping calculations on CPU fast. We found that a batch size of 64 works well for the supervised learning problems discussed in this thesis. Furthermore, each network in this section is trained for 200 epochs on the entire training dataset. We found that in most cases, increasing the amount of training epochs is beneficial. However, due to the large amount of network architectures that must be trained, we decided to terminate training after 200 epochs.

	Multilayer Perceptron (MLP)
Layers	$2, 3$
Nodes Per Layer	$50, 100, 150, 200, 250, 500$
Activation Function	ReLU
Loss Function	MSE
Learning Rate	10^{-5}
Batch Size	64

Table 4.5: An overview of multilayer perceptron configurations.

We allocate exactly one CPU core from DHPC [13] (Intel Xeon 3.0GHz) cluster to each model. Doing so allows us to measure the training time for each model and compare the performance trade-off to the off-line training time. Since we are training relatively small networks, GPU training does not speed up the process in most cases, as the overhead of distributing across the cores is too large. All networks are trained on the Black-Scholes and Heston pricing problems as well as the two implied volatility problems. The results for each problem are discussed below.

Black-Scholes Price

Figure 4.11 visualises the training time alongside the MSE of each network on the test set. We can see a clear relationship between the size of the network, which is increasing on the horizontal axis, and the reduction in MSE. Increasing the network size for the Black-Scholes problem decreases MSE and hence improves accuracy. The cost associated with this improvement is visible through the increase in training time.

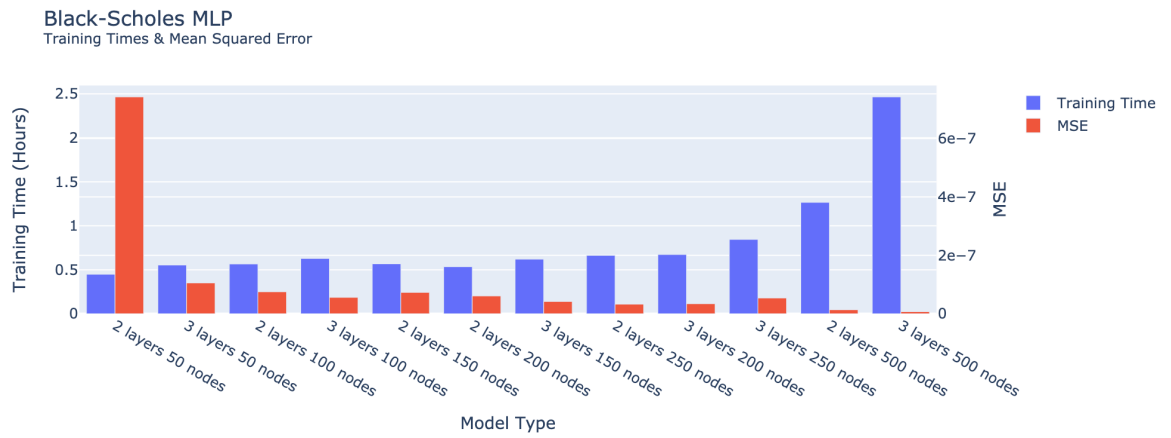

Figure 4.11: A comparison of the twelve networks from table 4.5 including MSE on the test set (red) and training time (blue) on the Black-Scholes problem.

Interestingly, the performance increase of a larger network is not immediately visible during training. Figure 4.12 visualises the training and validation losses over the 200 training epochs for the 2 layer networks with 50 nodes (small), 250 nodes (medium) and 500 nodes (large). Notice that even though the largest network is almost 5 times larger than the medium network and performs roughly 3.5 times as well as the medium network, the training loss is almost identical for both networks. This is an indication that losses during training do not represent actual performance on unseen data. Additionally, the validation loss for the medium and large networks is lower than the training loss. This indicates a possible performance improvement for both networks if training time is extended for more epochs, which aligns with findings in [25]. Since the Black-Scholes problem has an analytical solution, we expect to be able to approximate the solution using neural networks to any degree of precision, when choosing an appropriately sized network along with many training cycles. The results from this section support this hypothesis. We select the 3 layer 50 node network to compare against the highway and DGM networks discussed later.

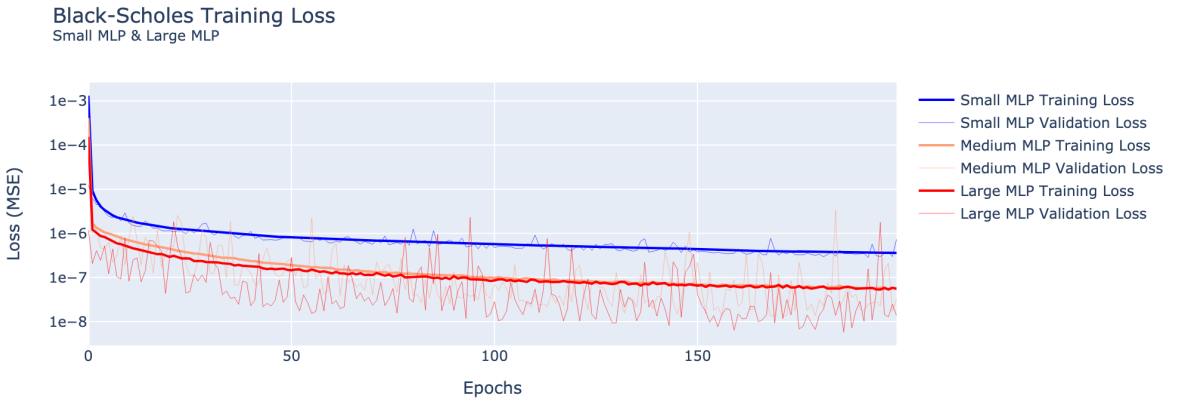

Figure 4.12: The training and validation loss of three networks. The small, medium and large network contain $2,851$, $64,251$ and $504,001$ parameters, respectively.

For completeness we list the configuration of each network in table 4.6, including training time and MSE.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
1	2	50	$2,851$	0.52	$2.3 \cdot 10^{-7}$
2	2	100	$10,701$	0.57	$7.5 \cdot 10^{-8}$
3	2	150	$23,551$	0.57	$7.3 \cdot 10^{-8}$
4	2	200	$41,401$	0.54	$6.1 \cdot 10^{-8}$
5	2	250	$64,251$	0.66	$3.3 \cdot 10^{-8}$
6	2	500	$253,501$	1.27	$1.4 \cdot 10^{-8}$
7	3	50	$5,401$	0.55	$1.1 \cdot 10^{-7}$
8	3	100	$20,801$	0.63	$5.6 \cdot 10^{-8}$
9	3	150	$46,201$	0.62	$4.2 \cdot 10^{-8}$
10	3	200	$81,601$	0.67	$3.4 \cdot 10^{-8}$
11	3	250	$127,001$	0.84	$5.4 \cdot 10^{-8}$
12	3	500	$504,001$	2.47	$7.0 \cdot 10^{-9}$
MLP [19]	6	200	$202,201$	-	$1.3 \cdot 10^{-7}$

Table 4.6: An overview of multilayer perceptron configurations along with training time and MSE for the Black-Scholes pricing problem. The best performing network is highlighted.

Heston Price

We find that for the MLP network architectures, the results for the Heston pricing problem align with the Black-Scholes problem. Since both problems are very similar and attempt to approximate the same quantity, the similarity in results are expected. Figure 4.13 shows the results of the various MLP's trained and evaluated on the Heston pricing problem.

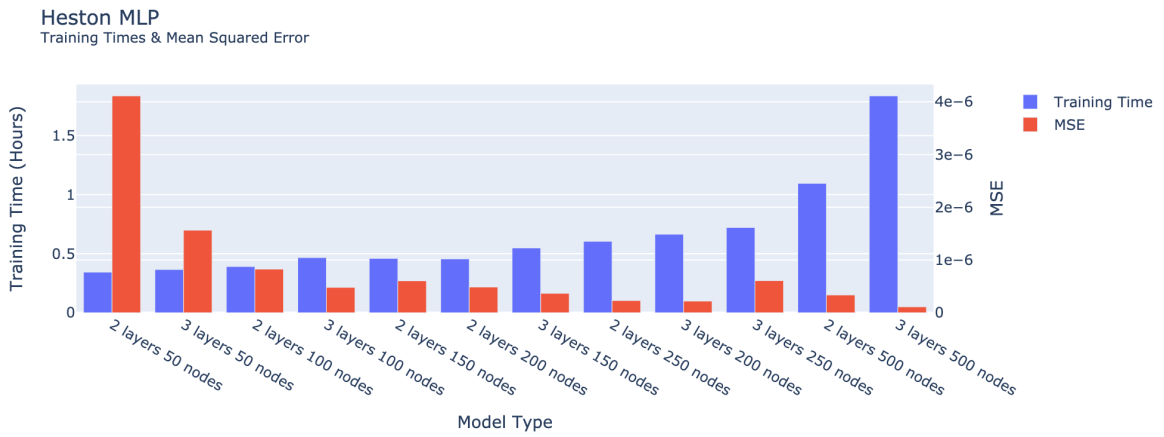

Figure 4.13: A comparison of the twelve network from table 4.5 including MSE on the test set (red) and training time (blue) on the Heston pricing problem.

The results from both the Black-Scholes as well as the Heston pricing problem exhibit, aside from the magnitude of the MSE's, many similarities (figures 4.11, 4.13). The MSE's and training times can be found in table 4.7.

In table 4.7, we observe that the MLP from [25] had a performance 10 times better than the best performing MLP that we trained. We trained each network for 200 epochs, whereas the authors of [25] train for 3000 epochs. Since the architecture of the networks are identical, this indicates once more that longer training on this problem improves performance, similar to what we find for the Black-Scholes pricing problem.

Implied Volatility

The main observation for the implied volatility problem is that contrary to the Black-Scholes problem, we cannot find evidence of larger networks performing better on the implied volatility problem. Indeed, the smallest network we train consists of 2 layers and 50 nodes per layer and containing a total of $2,851$ parameters, performs just as well after 200 epochs with identical configuration as the largest network

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
1	2	50	$3,051$	0.34	$4.1 \cdot 10^{-6}$
2	2	100	$11,101$	0.39	$8.3 \cdot 10^{-7}$
3	2	150	$24,151$	0.46	$6.0 \cdot 10^{-7}$
4	2	200	$42,201$	0.46	$4.9 \cdot 10^{-7}$
5	2	250	$65,251$	0.60	$2.3 \cdot 10^{-7}$
6	2	500	$255,501$	1.09	$3.4 \cdot 10^{-7}$
7	3	50	$5,601$	0.37	$1.6 \cdot 10^{-6}$
8	3	100	$21,201$	0.47	$4.8 \cdot 10^{-7}$
9	3	150	$46,801$	0.55	$3.7 \cdot 10^{-7}$
10	3	200	$82,401$	0.66	$2.2 \cdot 10^{-7}$
11	3	250	$128,001$	0.72	$6.1 \cdot 10^{-7}$
12	3	500	$506,001$	1.84	$1.1 \cdot 10^{-7}$
MLP [25]	4	400	$646,006$	-	$1.7 \cdot 10^{-8}$

Table 4.7: An overview of multilayer perceptron configurations along with training time and MSE for the Heston pricing problem. The best performing network is highlighted.

with 3 layers and 500 nodes containing $504,001$ parameters. Both networks have a MSE on the test set of $7.7 \cdot 10^{-4}$, while the small network took only 0.4 hours to train compared to the 2.17 hours for the largest network. All networks show comparable performance, as can be seen in figure 4.14.

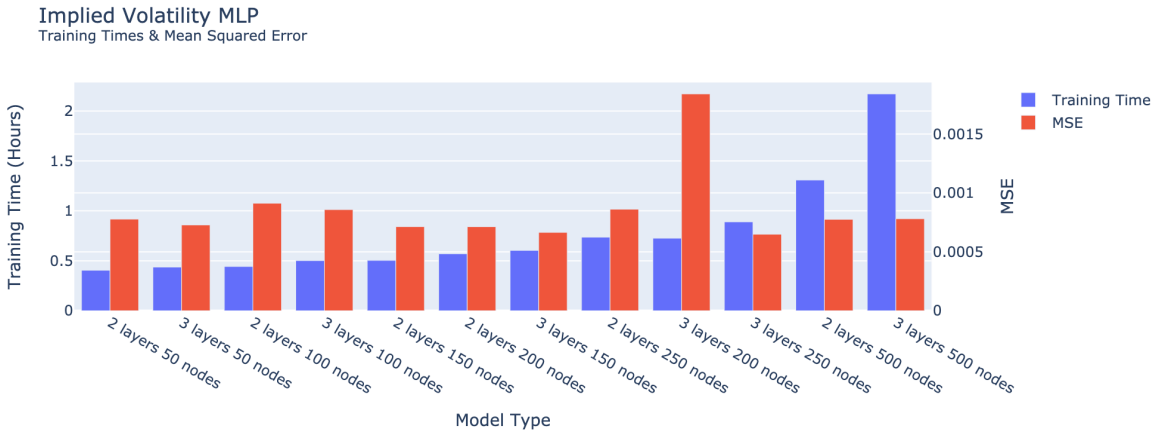

Figure 4.14: A comparison of the twelve network from table 4.5 including MSE on the test set (red) and training time (blue) on the implied volatility problem.

The identical performance of the networks during training is visualised in figure 4.15. The results we find align with the errors found in [19], in which the best performing MLP network containing $202,201$ parameters performed worse on the test data than the considerably smaller networks from figure 4.14.

Table 4.8 lists all the networks along with the training times, amount of parameters and MSE for the implied volatility problem. The results indicate that the networks are dealing with convergence issues, preventing them to optimise further, regardless of their size and architecture. We consider the transformed implied volatility dataset, where we aim to solve the steep gradient problems possibly causing the convergence issues by introducing a transformation from the scaled call price to the scaled time value of the option, explained in section 4.4.1.

Figure 4.16 visualises the performances of the MLP's on the transformed implied volatility problem. We notice that on the transformed problem, the MSE is decreasing as a function of the network size, similar to what we found for the previous option pricing problems. The reduction in MSE is not as consistent as the previous problems, indicating difficult convergence during training, possibly still due to a steep gradient not being entirely removed by the transformation. The results are considerably better than for the default implied volatility problem, using only a simple transformation. Table 4.9 shows the

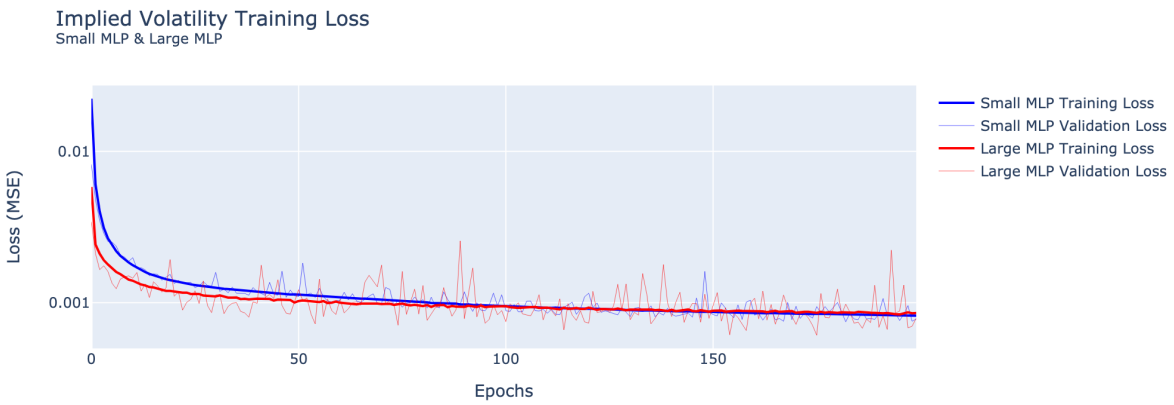

Figure 4.15: The training and validation loss of the smallest network compared to the largest network on the implied volatility data set.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
1	2	50	2,851	0.40	$7.7 \cdot 10^{-4}$
2	2	100	10,701	0.44	$9.1 \cdot 10^{-4}$
3	2	150	23,551	0.50	$7.1 \cdot 10^{-4}$
4	2	200	41,401	0.57	$7.1 \cdot 10^{-4}$
5	2	250	64,251	0.74	$8.6 \cdot 10^{-4}$
6	2	500	253,501	1.31	$7.8 \cdot 10^{-4}$
7	3	50	5,401	0.44	$7.3 \cdot 10^{-4}$
8	3	100	20,801	0.50	$8.5 \cdot 10^{-4}$
9	3	150	46,201	0.60	$6.6 \cdot 10^{-4}$
10	3	200	81,601	0.73	$1.8 \cdot 10^{-3}$
11	3	250	127,001	0.94	$7.6 \cdot 10^{-4}$
12	3	500	504,001	2.17	$7.7 \cdot 10^{-4}$

Table 4.8: An overview of multilayer perceptron configurations along with training time and MSE for the implied volatility problem. The best performing network is highlighted.

results of all MLP's along with their parameters and training times.

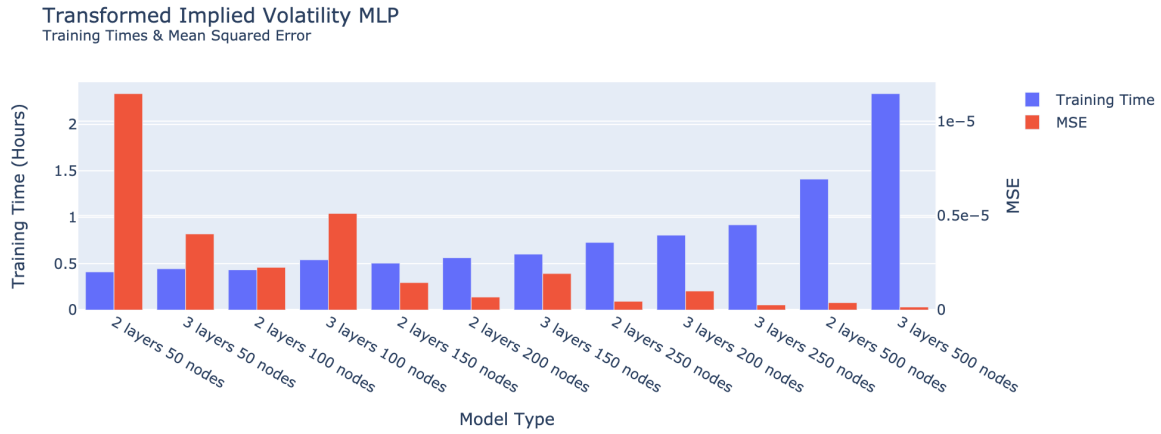

Figure 4.16: A comparison of the twelve network from table 4.5 including MSE on the test set (red) and training time (blue) on the transformed implied volatility problem.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
1	2	50	2,851	0.41	$1.1 \cdot 10^{-5}$
2	2	100	10,701	0.43	$2.3 \cdot 10^{-6}$
3	2	150	23,551	0.51	$1.5 \cdot 10^{-6}$
4	2	200	41,401	0.56	$1.9 \cdot 10^{-6}$
5	2	250	64,251	0.73	$6.9 \cdot 10^{-6}$
6	2	500	253,501	1.41	$3.9 \cdot 10^{-7}$
7	3	50	5,401	0.44	$4.0 \cdot 10^{-6}$
8	3	100	20,801	0.54	$5.1 \cdot 10^{-6}$
9	3	150	46,201	0.60	$1.9 \cdot 10^{-6}$
10	3	200	81,601	0.81	$1.0 \cdot 10^{-6}$
11	3	250	127,001	0.92	$2.7 \cdot 10^{-7}$
12	3	500	504,001	2.33	$1.6 \cdot 10^{-7}$

Table 4.9: An overview of multilayer perceptron configurations along with training time and MSE for the transformed implied volatility problem.

4.4.3. Highway Network

	Generalised Highway	Highway	Residual
Layers	3	3	3
Nodes Per Layer	50	50	50
Total Parameters	23,251	15,601	7,951
Initialiser	Glorot Normal	Glorot Normal	Glorot Normal
Carry/Transform Activation Function	$tanh$	$tanh$	-
Activation Function	$tanh$	$tanh$	$tanh$
Loss Function	MSE	MSE	MSE
Learning Rate	10^{-5}	10^{-5}	10^{-5}
Batch Size	64	64	64

Table 4.10: An overview of highway network variations.

In this section, we train each variant of the highway network discussed in section 4.3.2 with approximately equal configuration. Specifically, each network contains 3 layers each with 50 nodes. We train relatively small networks to compare across various architectures, in order to keep computation times feasible. Table 4.10 lists each highway network configuration.

Black-Scholes Price

Figure 4.17 visualises the training times (blue) and MSE on the test set (red) for each highway network. Both the highway and generalised highway networks perform better than their MLP counterpart with equal amount of nodes. More impressive, however, is the fact both networks required barely more training time than the small MLP, but score comparable on the test set with respect to the large MLP. This means that the highway architecture improves convergence on the Black-Scholes problem significantly. The difference in convergence speed can also be seen during training, shown in figure 4.18. In this case, training losses are a good indicator for performance on the test set, likely because the performance difference for both networks is so large. The loss during training is roughly one order of magnitude smaller for the generalised highway model, whereas the training time is only 23% longer.

The residual layer is most similar to the classic dense layer. We notice that the only difference is the carry from the input $\vec{x}$ (equation 4.37). Surprisingly, only carrying the input vector to the next layer decreases the MSE on the test set by 33% from $1.1 \cdot 10^{-7}$ to $6.7 \cdot 10^{-8}$. Since only one addition operation is added in each layer, the training time is almost identical. In [20], the authors argue that including the input vector in many operations allows the network to make 'sharper turns', resulting in a more flexible network. Indeed, we see that this feature becomes particularly useful when approximating the Black-Scholes price.

The difference in performance between the highway network and the residual network is also significant. The incorporation of one extra weight matrix and non-linear function multiplied with the input

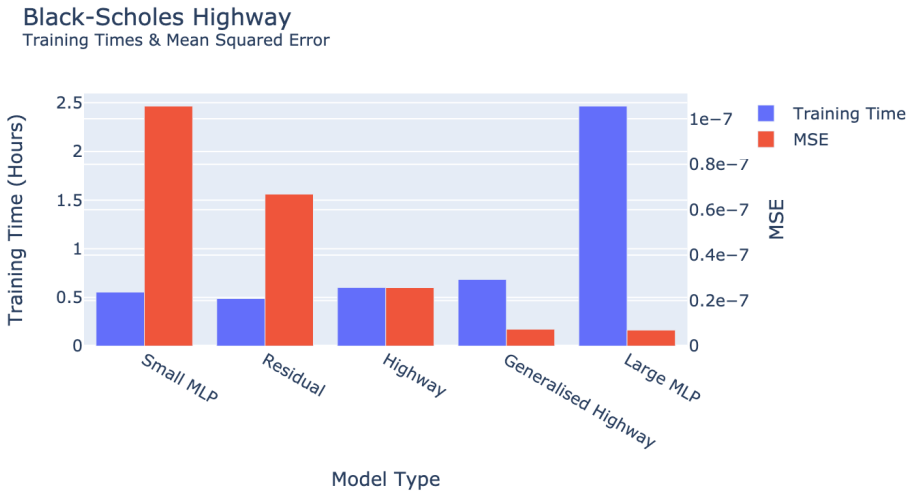

Figure 4.17: A comparison of highway networks from table 4.10 including the smallest and largest MLP with the MSE on the test set (red) and training time (blue) on the Black-Scholes pricing problem. The small MLP contains 3 layers with 50 nodes, and the large MLP contains 3 layers with 500 nodes. The results are listed ordered by their total parameters.

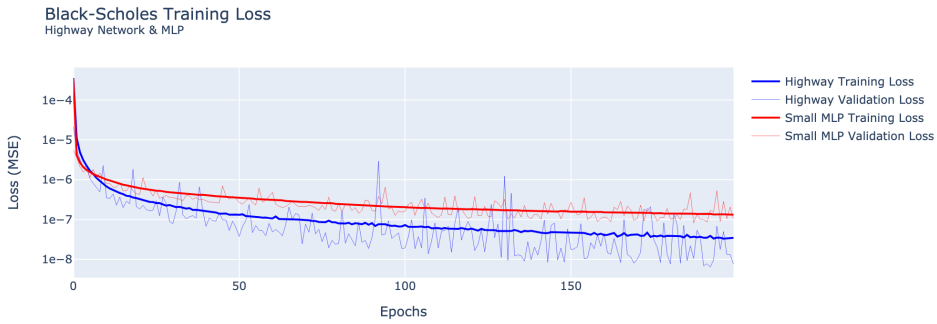

Figure 4.18: The training and validation loss over 200 epochs. The small MLP consisting of 3 layers and 50 nodes (2,851 parameters) in red, compared to the 3 layer 50 node generalised highway network (23,251 parameters) in blue.

vector $\vec{x}$ (equation 4.38) in each layer causes an MSE reduction of 61% from the residual network's $6.7 \cdot 10^{-8}$ to the highway network's $2.6 \cdot 10^{-8}$. Adding one additional weight matrix W_C and non-linearity C, yielding the generalised highway layer, improves MSE performance by another 61% compared to the highway layer. An overview of the error reductions, training times and operations in each layer are given in table 4.11.

	Time (H)	MSE	Reduction (%)	Layer Operation
MLP	0.55	$1.1 \cdot 10^{-7}$	-	$H(\vec{x}, W_H)$
Residual	0.55	$6.7 \cdot 10^{-8}$	33%	$H(\vec{x}, W_H) + \vec{x}$
Highway	0.60	$2.8 \cdot 10^{-8}$	61%	$H(\vec{x}, W_H) \cdot T(\vec{x}, W_T) + \vec{x} \cdot (1 - T(\vec{x}, W_T))$
Generalised Highway	0.68	$7.4 \cdot 10^{-9}$	73%	$H(\vec{x}, W_H) \cdot T(\vec{x}, W_T) + \vec{x} \cdot C(\vec{x}, W_C)$

Table 4.11: An overview of the highway network performance and error reductions for the MLP and highway network architectures on the Black-Scholes pricing problem.

Finally, we notice that a 24% increase in training time from the small MLP to the generalised highway network decreases the MSE by 92%. When comparing the results obtained for the highway networks to the results in [25, Table 6], in which a large MLP is trained and evaluated on identical data, we see that our small generalised highway network already outperforms the MLP. In fact, as shown in table 4.14, the generalised highway network improves the MSE by 9.8% while reducing the amount of parameters

by more than 96% compared to the MLP from [25].

Heston Price

Figure 4.19 visualises the performance of the highway networks on the Heston pricing problem. We again see a similar trend as in the Black-Scholes pricing problem (figure 4.17). A major difference is the performance of the residual network, which reduced the MSE by 33% (table 4.11) compared to the MLP on the Black-Scholes problem. On the Heston pricing problem, we see no increases in performance. A possible explanation for this could be that the Heston pricing problem might be too involved for the residual architecture to make an improvement.

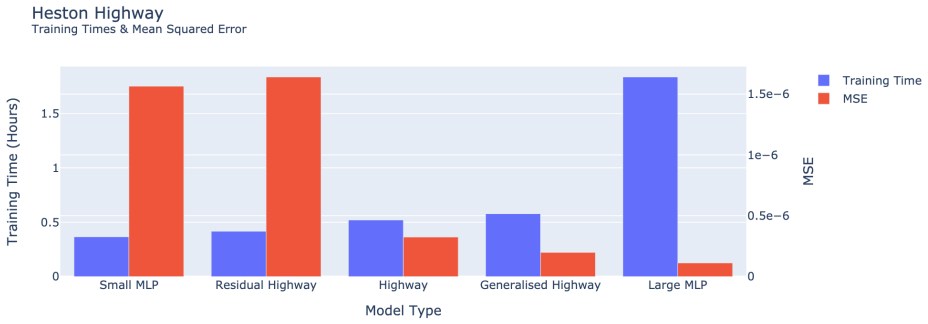

Figure 4.19: A comparison of highway networks from table 4.10 including the smallest and largest MLP with the MSE on the test set (red) and training time (blue) on the Heston pricing problem. The small MLP contains 3 layers with 50 nodes, and the large MLP contains 3 layers with 500 nodes.

The highway and generalised highway networks do, in fact, improve performance significantly compared to the MLP networks, with the generalised highway network performing almost as well as the largest MLP, while requiring only 31% of the computation time. Table 4.12 lists the reductions that each highway variation contributes to the MSE.

	Time (H)	MSE	Reduction (%)	Layer Operation
MLP	0.37	$1.6 \cdot 10^{-6}$	-	$H(\vec{x}, W_H)$
Residual	0.42	$1.7 \cdot 10^{-6}$	-5%	$H(\vec{x}, W_H) + \vec{x}$
Highway	0.58	$3.3 \cdot 10^{-7}$	80%	$H(\vec{x}, W_H) \cdot T(\vec{x}, W_T) + \vec{x} \cdot (1 - T(\vec{x}, W_T))$
Generalised Highway	0.68	$2.0 \cdot 10^{-7}$	39%	$H(\vec{x}, W_H) \cdot T(\vec{x}, W_T) + \vec{x} \cdot C(\vec{x}, W_C)$

Table 4.12: An overview of the highway network performance and error reductions for the MLP and highway network architectures on the Heston pricing problem.

We notice that the improvement of the generalised highway network on the Heston problem is less significant than on the Black-Scholes problem, where the generalised highway network was the best performing network. In section 4.4.4 we compare the highway networks to their DGM counterpart on the Heston pricing problem, and consider equally sized networks in terms of parameters.

Implied Volatility

In the previous section we found very significant reductions in MSE on the test set when comparing the highway networks to the MLP's on the Black-Scholes and Heston pricing problems. In this section, we do the same comparison for the implied volatility problem. The results are visualised in figure 4.20. In contrast to the Black-Scholes price problem, in which the highway networks outperformed the MLP's, this is not the case for the implied volatility problem. None of the highway networks outperform on the test set compared to the MLP. As in the MLP scenario, this indicates again that the steep gradient also poses a problem when optimising the highway networks. To mitigate this issue, we consider the performance of the networks on the transformed implied volatility problem.

In figure 4.21 the performance of the highway networks on the transformed implied volatility dataset is shown. On this dataset, we find similar relative MSE's and computation times as the Black-Scholes and Heston pricing problems. In fact, the generalised highway network relative to the computation time

Figure 4.20: A comparison of highway networks from table 4.10 including the smallest and largest MLP with the MSE on the test set (red) and training time (blue) on the implied volatility problem. The small MLP contains 3 layers with 50 nodes, and the large MLP contains 3 layers with 500 nodes.

is again the superior network. However, just like in the Heston pricing problem scenario, the large MLP scores lower in absolute MSE. This indicates that the network performance on the implied volatility problem is also prone to larger network sizes. We expect to see performance improvements when training highway networks with more nodes and layers.

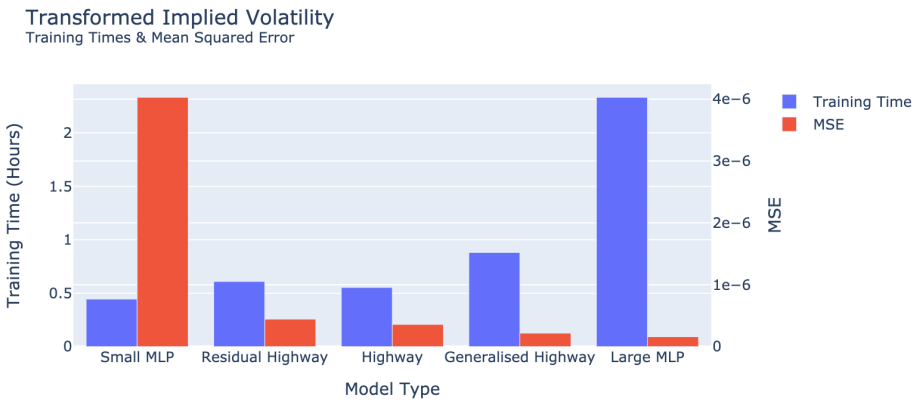

Figure 4.21: A comparison of highway networks from table 4.10 including the smallest and largest MLP with the MSE on the test set (red) and training time (blue) on the transformed implied volatility dataset. The small MLP contains 3 layers with 50 nodes, and the large MLP contains 3 layers with 500 nodes.

It becomes clear how much influence the network architecture has on the performance when considering the difference between the small MLP and residual highway network. Recall that the operation of a residual network is simply the addition of the input vector to the layer (equation 4.37). This operation reduces the MSE by roughly 89%, while adding only 38% to the training time. We also notice that the training times for an identical network architecture can differ per dataset. To see this, consider the 14% increase in training time from the small MLP to the residual network on the Heston problem (table 4.12), and the 0% increase on the Black-Scholes problem (table 4.11).

4.4.4. DGM Network

In this section we train the DGM network proposed for solving PDE models in [20] and compare its performance to the highway and MLP networks. We train the DGM network with the configuration listed in table 4.13.

	DGM Network
Layers	3
Nodes Per Layer	50
Total Parameters	33,459
Activation Function	$tanh$
Initialiser	Glorot Normal
Loss Function	MSE
Learning Rate	10^{-5}
Batch Size	64

Table 4.13: The configuration of the trained DGM network used for analysis in this section.

Black-Scholes Price

The results of the DGM network performance on the Black-Scholes problem are visualised in figure 4.22. We note that more operations must be computed inside a DGM network than in a highway network, resulting in slightly more parameters and hence in longer training time. This aligns with the numerical experiments.

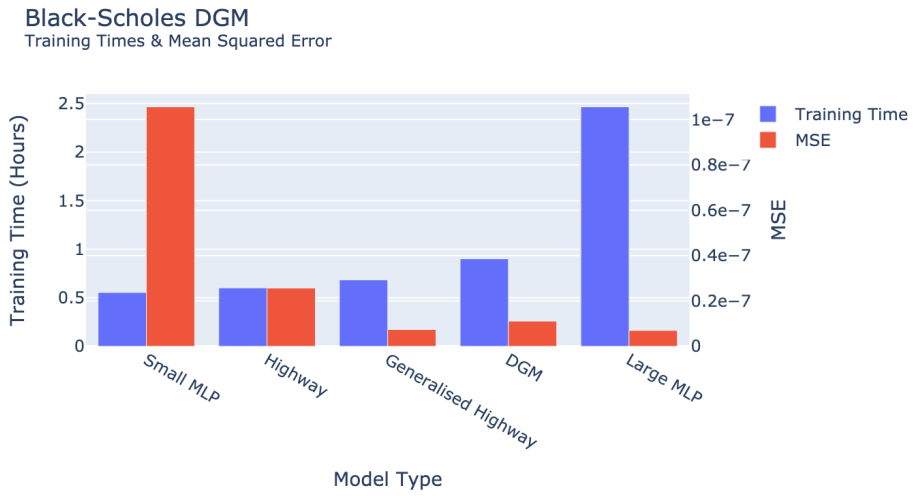

Figure 4.22: The MLP, highway and DGM networks compared in terms of MSE on the test set (red) and training time (blue) for the Black-Scholes pricing problem. The small MLP contains 3 layers with 50 nodes, and the large MLP contains 3 layers with 500 nodes.

The main difference between the highway layer and the DGM layer is that the original input $\vec{x}$ is used in each DGM layer, whereas the highway layer only depends on the output of the previous layer. The dependence on $\vec{x}$ introduces 4 additional weight matrixes (figure 4.7) per layer. In figure 4.22 no significant improvements of this dependence compared to the generalised highway network can be found. Possibly, the Black-Scholes problem is not complex enough for the additional non-linearity and recurrence in the DGM layers to cause improvements. When comparing the DGM network to the MLP and highway network in the training phase (figure 4.23), we notice that the DGM network consistently outperforms the highway and MLP networks on the training and validation sets. This observation may suggest that the DGM network is more flexible and can fit the training data better, possibly leading to overfitting, resulting in underperformance on the test set compared to the generalised highway network.

In order to gain more insights into the impact of the operations inside the DGM network on the performance, we also train the deep DGM network and the no-recurrence DGM network with the same configuration as in table 4.13, with the deep DGM layer containing 3 sublayers. In figure 4.24 the MSE's of the DGM variations are compared against the DGM and highway networks. As expected from previous observations, the addition of sublayers inside the DGM layer, resulting in the deep DGM network (section 4.3.5), does not bring significant performance improvements, while increasing training time by almost 50%. Removing the dependence on the input vector $\vec{x}$ in all layers, resulting in the no-

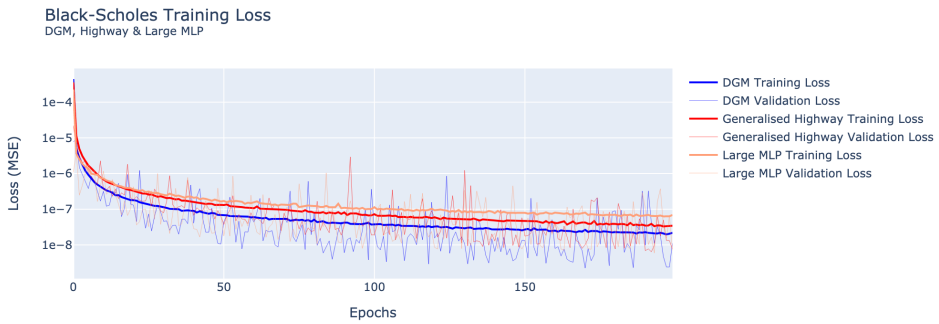

Figure 4.23: The MLP, highway and DGM networks compared during the training cycles.

recurrence DGM network (section 4.3.6), more than tripled the network error. This suggests that for the Black-Scholes pricing problem, the DGM network admits its performance benefits from the recurrence in the network.

The operations inside the no-recurrence network are very similar to those in a highway network. However, the MSE's on the test set are nowhere close. To verify that this was not a numerical issue, the no-recurrence network was trained multiple times, yielding similar results. The layer operations for the no-recurrence network and the generalised highway network after renaming weight matrices and activations are

$$\vec{y} = \left[1 - G(\vec{x}, W_G)\right] \cdot H\left(R(\vec{x}, W_R) \cdot \vec{x}\right) + Z(\vec{x}, W_Z) \cdot \vec{x} \tag{4.50}$$

and

$$\vec{y} = \left[1 - Z(\vec{x}, W_Z)\right] \cdot H(\vec{x}, W_H) + Z(\vec{x}, W_Z) \cdot \vec{x}, \tag{4.51}$$

respectively. In this case, $\cdot$ denotes element wise multiplication. Hence, the difference in performance likely originates from the additional non-linearity R or G.

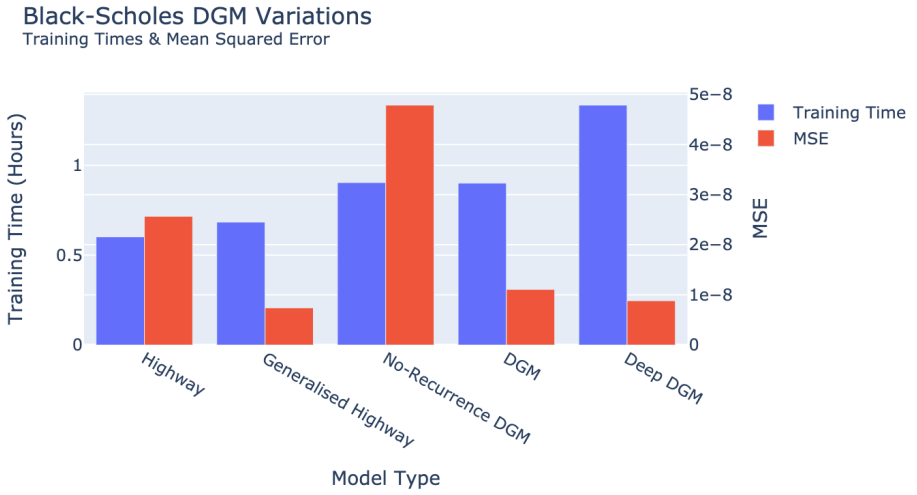

Figure 4.24: The highway and DGM networks, including DGM variations (deep DGM and no-recurrence DGM) compared in terms of MSE on the test set (red) and training time (blue) for the Black-Scholes pricing problem.

The fact that the introduction of two additional operations have such a negative influence in accuracy suggests that choosing an optimal network architecture can be a process subject to high sensitivity.

For completeness, table 4.14 lists the Black-Scholes pricing MSE results for the most important networks, alongside the performances of the networks from [19, 25]. We notice that the generalised

highway networks outperforms all networks in terms of performance compared to total parameters and training time, and can be considered as the superior network.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
Small MLP	3	50	5,401	0.55	$1.1 \cdot 10^{-7}$
Highway	3	50	15,601	0.60	$2.6 \cdot 10^{-8}$
Generalised Highway	3	50	23,251	0.68	$7.4 \cdot 10^{-9}$
No-Recurrence DGM	3	50	31,059	0.90	$4.8 \cdot 10^{-8}$
DGM	3	50	33,459	0.90	$1.1 \cdot 10^{-8}$
Deep DGM	3	50	49,959	1.33	$8.8 \cdot 10^{-9}$
Large MLP	3	500	504,001	2.47	$7.0 \cdot 10^{-9}$
MLP [19]	6	200	202,201	-	$1.3 \cdot 10^{-7}$
DGM [19]	5	100	210,601	-	$1.6 \cdot 10^{-7}$
MLP* [25]	4	400	644,401	-	$8.2 \cdot 10^{-9}$

*Trained for 3000 epochs, whereas the other networks are trained for 200.

Table 4.14: An overview of MLP, highway and DGM network configurations along with training time and MSE for the Black-Scholes pricing problem. The best performing networks are highlighted. We also consider the networks from [19] and [25].

Heston Price

In figure 4.25 the performance of the DGM network alongside the smallest and largest 3 layer MLP and highway networks on the Heston pricing problem is shown. We notice that unlike in the Black-Scholes problem, the DGM network achieves a lower MSE than both the large MLP as well as the generalised highway network. The reason could be that the DGM has slightly more parameters, which we noticed in section 4.4.2 can lead to performance improvements.

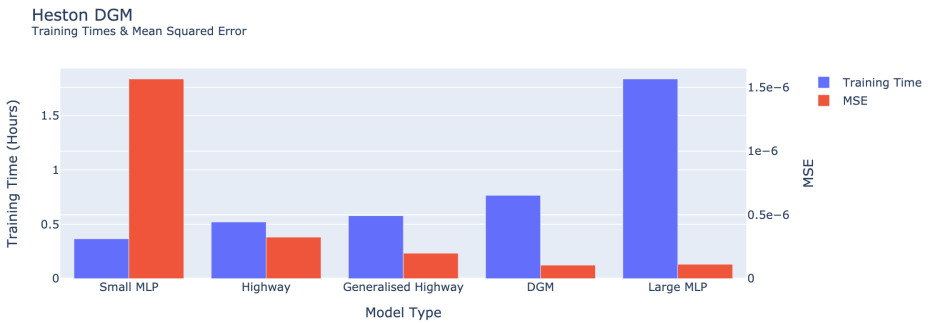

Figure 4.25: The highway and DGM networks, including DGM variations (Deep DGM and No-Recurrence DGM) compared in terms of MSE on the test set (red) and training time (blue) for the Black-Scholes pricing problem.

To validate this assumption, we train a highway network and DGM network with approximately equal amount of parameters to the generalised highway network by varying the amount of layers. The configuration of these networks is shown in table 4.15.

	Highway	Generalised Highway	DGM
Layers	4	3	2
Nodes Per Layer	50	50	50
Total Parameters	20,901	23,451	24,467

Table 4.15: The changed layer configuration for the highway and DGM network, in order to have comparable amount of parameters.

The results are visualised in figure 4.26. We can see that when the amount of parameters are comparable, the generalised highway network outperforms the highway and DGM network in both

training time as well as MSE. Hence, the reason the DGM network is performing better in figure 4.25 is likely due to the increased number of parameters.

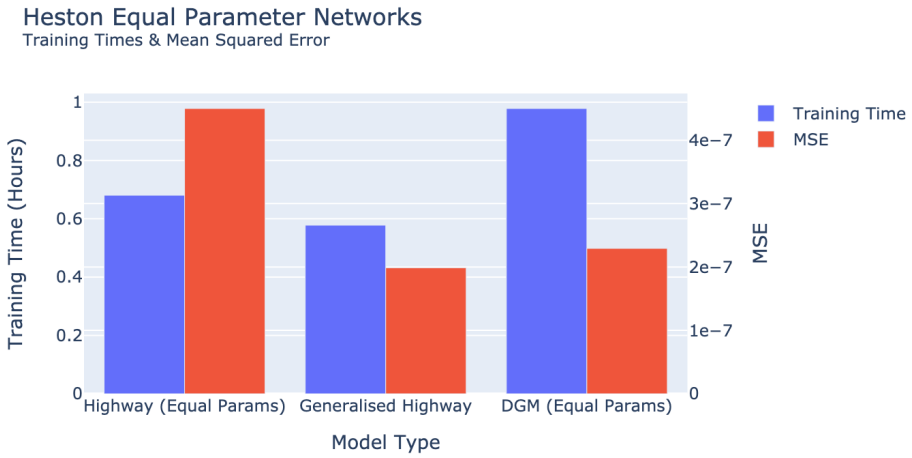

Figure 4.26: The highway, generalised highway and DGM networks where we compare the networks with comparable amounts of parameters on the Heston pricing problem. The generalised highway network performs best in both MSE and training time.

We can conclude that the generalised highway network also performs well for the Heston price problem, when the amount of parameters is increased. The improvement in MSE with respect to the DGM network is marginal, but the computation time required for training a generalised highway network is considerably lower.

Finally, in figure 4.27 the results of the deep DGM and the no-recurrence DGM networks are shown. We notice that neither the no-recurrence network nor the deep DGM network significantly improve performance or training time. Interestingly, the no-recurrence DGM also requires slightly longer training time, despite a fewer amount of parameters in the network, compared to the DGM network.

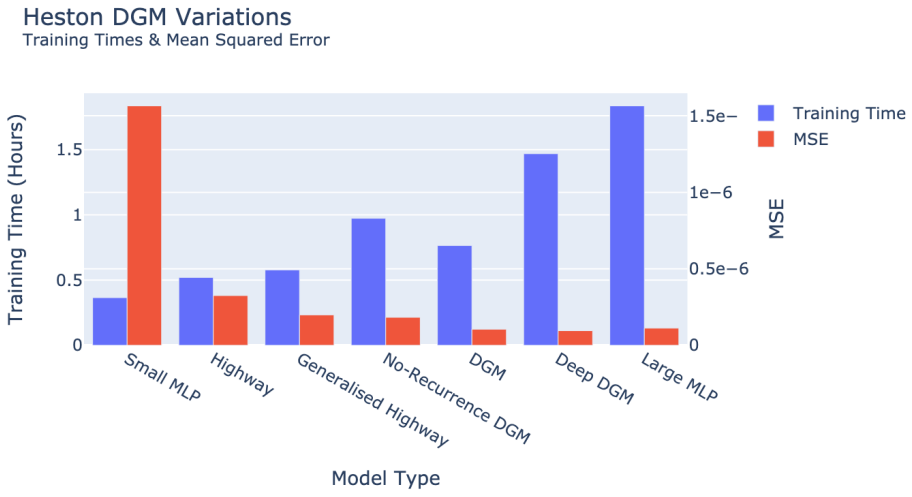

Figure 4.27: The highway and DGM networks, including DGM variations (Deep DGM and No-Recurrence DGM) compared in terms of MSE on the test set (red) and training time (blue) for the Heston pricing problem.

We can conclude that the DGM network outperforms both variations. Additionally, we find that while the recurrence in the DGM network has decreased the MSE, the additional complexity it brings along does not make it suitable for option pricing problems in the tested setup.

Implied Volatility

Finally, we examine the performance of the DGM networks on the implied volatility problem. We have seen in the previous sections that the implied volatility problem has not been susceptible to performance increases with different architectures, which is likely due to the steep gradient problem present in the default implied volatility dataset. We again find this result when considering the DGM networks.

Figure 4.28: The highway, MLP and DGM network including DGM variations (deep DGM and no-recurrence DGM) compared in terms of MSE on the test set (red) and training time (blue) for the implied volatility problem.

Table 4.16 lists the training times and MSE on the implied volatility problem for the networks considered in figure 4.28. We notice that the 3 layer 150 node MLP performs best in terms of MSE and training time. We find comparable performance from the networks trained in [19, 25].

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
Small MLP	3	50	$5,401$	0.44	$7.3 \cdot 10^{-4}$
Highway	3	50	$15,601$	0.51	$7.4 \cdot 10^{-4}$
Generalised Highway	3	50	$23,251$	0.62	$1.0 \cdot 10^{-3}$
No-Recurrence DGM	3	50	$31,059$	0.90	$8.1 \cdot 10^{-4}$
DGM	3	50	$33,459$	0.87	$8.7 \cdot 10^{-4}$
Best MLP	3	150	$46,201$	0.60	$6.6 \cdot 10^{-4}$
Deep DGM	3	50	$49,959$	1.33	$6.6 \cdot 10^{-4}$
Large MLP	3	500	$504,001$	2.17	$7.8 \cdot 10^{-4}$
MLP [19]	6	200	$202,201$	-	$8.1 \cdot 10^{-4}$
DGM [19]	5	100	$210,601$	-	$6.5 \cdot 10^{-4}$
MLP* [25]	4	400	$644,401$	-	$6.4 \cdot 10^{-4}$

*Trained for 3000 epochs, whereas the other networks are trained for 200.

Table 4.16: An overview of MLP, highway and DGM network configurations along with training time and MSE for the implied volatility problem. The best performing networks are highlighted. We also consider the networks from [19] and [25].

Since the results we find on the implied volatility dataset is probably a result of convergence problems originating from the dataset, we consider DGM performance on the transformed implied volatility problem in the remainder of this section. We observed in section 4.4.3 that the highway architectures showed strong signs of decreased MSE with little increase in computation time. Figure 4.29 shows the results of the DGM alongside highway and MLP reference networks.

We notice, similar to the Heston pricing problem, that the DGM network outperforms the other networks in terms of absolute MSE. However, the generalised highway network performs better when computational training time is taken into account. Contrary to the Heston pricing problem, where we observed that when equating the amount of parameters in each network the generalised highway network was the superior network, we find that the DGM might be the better network architecture for the transformed implied volatility problem. Indeed, figure 4.30 visualises the MSE reduction of 35% from

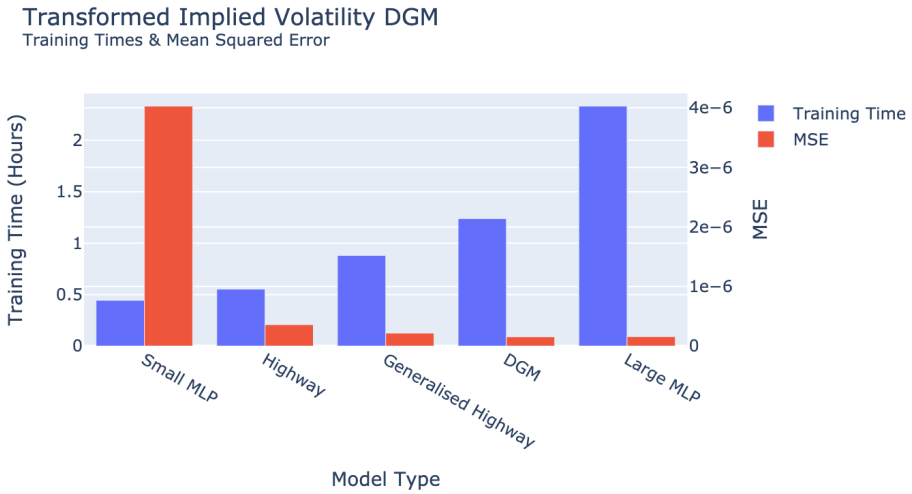

Figure 4.29: The highway, MLP and DGM networks compared in terms of MSE on the test set (red) and training time (blue) for the transformed implied volatility problem.

the generalised highway network to the DGM network, only requiring a 25% increase of computation time.

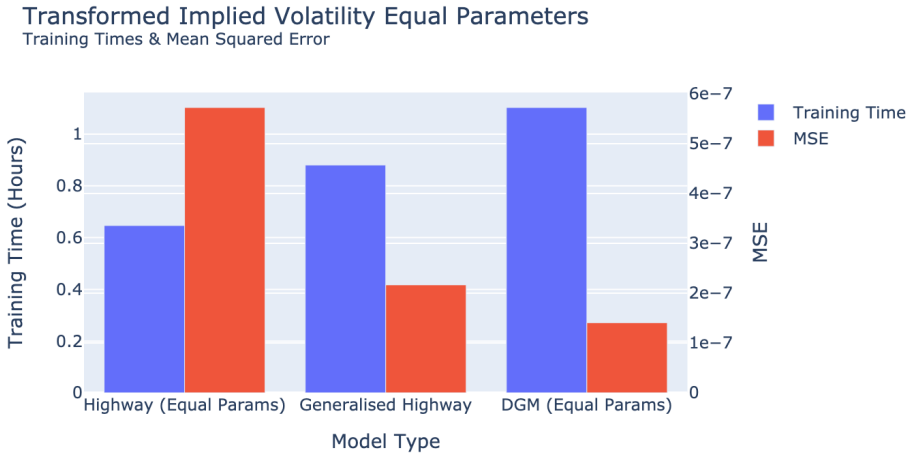

Figure 4.30: The highway, generalised highway and DGM networks where we compare the networks with comparable amounts of parameters on the transformed implied volatility problem.

When we also include the results of the DGM variations (figure 4.31), we find that the no-recurrence DGM shows even better performance than the DGM network. This network architecture is similar to the generalised highway architecture, but contains one additional non-linear operation. For all the previous problems, the additional non-linear operation caused reductions in performance compared to the simpler generalised highway architecture. For the transformed implied volatility problem, however, we find that the no-recurrence architecture shows a 51% reduction in MSE compared to the DGM network, and a 66% reduction compared to the generalised highway. Table 4.17 shows the MSEs, training times and parameters for all the networks of interest.

4.4.5. Conclusions

From the past experiments we can draw a few conclusions that can assist us when choosing a network architecture for related problems.

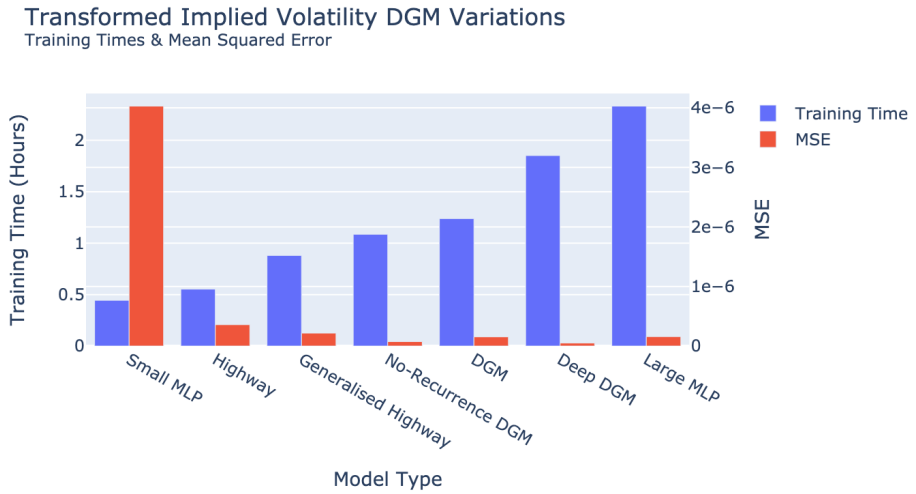

Figure 4.31: The highway, MLP and DGM network including DGM variations (deep DGM and no-recurrence DGM) compared in terms of MSE on the test set (red) and training time (blue) for the transformed implied volatility problem.

Model	Layers	Nodes	Parameters	Training Time (H)	MSE
Small MLP	3	50	$5,401$	0.44	$4.0 \cdot 10^{-6}$
Highway	3	50	$15,601$	0.55	$3.6 \cdot 10^{-7}$
Generalised Highway	3	50	$23,251$	0.88	$2.2 \cdot 10^{-7}$
No-Recurrence DGM	3	50	$31,059$	1.09	$7.4 \cdot 10^{-8}$
DGM	3	50	$33,459$	1.24	$1.5 \cdot 10^{-7}$
Deep DGM	3	50	$49,959$	1.33	$5.0 \cdot 10^{-8}$
Large MLP	3	500	$504,001$	2.33	$1.6 \cdot 10^{-7}$
MLP* [25]	4	400	$644,401$	-	$1.6 \cdot 10^{-8}$

*Trained for 3000 epochs, whereas the other networks are trained for 200.

Table 4.17: An overview of MLP, highway and DGM network configurations along with training time and MSE for the implied volatility problem. The best performing networks are highlighted. We also consider the network from [25].

- The first observation we made is that increasing parameters in the considered networks influenced the performance on the Black-Scholes, Heston and transformed implied volatility problem. We find that in many scenario's, increasing the amount of parameters in combination with a suitable architecture does in fact decrease the MSE and therefore increase the performance.
- We did not find evidence that losses during training were a good indicator for the performance of the network during evaluation. Instead, we found that networks performing well on the training data did in no case outperform the other networks on the testing data (i.e. the DGM network on the Black-Scholes pricing problem, figures 4.22 and 4.23). We would have liked to see such a relationship, as it allows us to terminate the training cycle prematurely when no improvement is made on the training set compared to other networks, saving valuable computation time.
- During training, the validation loss was often lower than the training loss. This indicates we could possibly improve the accuracy of the networks by training for more epochs. In [25], results on the Heston pricing problem were more accurate, training for 3000 cycles compared to 200 in this thesis, while using an identical configuration for the MLP. This suggests we could improve the network performance by training for more epochs. Additionally, we find in chapter 5 that a variable learning rate could be beneficial for these problems, as also considered in [20].
- For the Black-Scholes and Heston pricing problems, we found that the generalised highway network architecture consistently outperformed the other networks when we compared the MSE relative to the computation time. In the case of the transformed implied volatility problem, we also found good performance results for the generalised highway problem, outperformed only by the

no-recurrence DGM architecture. This architecture is similar to the generalised highway network, but did not yield good results on the pricing problems. In conclusion, small architecture variations can have large impacts on network performance.

- When training networks on the implied volatility dataset, we found poor performance for all networks, and no improvements when using more complex architectures. When transforming the dataset slightly, as explained in section 4.4.1, the prediction accuracy increased significantly, and we found similar results in network architecture performance compared to the Black-Scholes and Heston pricing problems. This finding suggests that the training of networks is highly dependent on the dataset. When training neural networks, we should try to ensure that gradients are bounded. A possible way to do this is to transform the input variables so that the gradient with respect to the variables is consistent.
- Finally, we found that for the considered problems, the recurrence behaviour of the DGM networks did not yield benefits when compared to the generalised highway network. Instead, the computational complexity caused by the increase of operations inside the DGM layers made it less attractive in terms of relative performance to off-line required computation power than the generalised highway network. In chapter 5, we examine the DGM network in a different problem setting, and see that the DGM network with its recurrent behaviour provides serious benefits.

5

PDE Models

In section 4.4 we conducted an empirical study on the performance of neural networks on option pricing methods. In particular, we sampled data pairs $(\vec{x}, y)$ according to the Black-Scholes and Heston models, which we used to train the networks in a *supervised learning* fashion. Specifically, we used features $\vec{x}$ and corresponding labels y, and iteratively update our approximation function f_n based on the loss $J = \|f_n(\vec{x}) - y\|$. The downside to this technique is that we must first evaluate the stochastic model using expensive computations to obtain y, after which we must learn the network, which is yet again a computationally expensive task.

In this chapter, we introduce a *semi-supervised learning* method, in which we need not evaluate the labels y for training the network. Let $f(\vec{x}, t; \theta)$ be our neural network with parameters θ. This method considers parabolic PDE's along with an initial and a boundary condition, and creates an objective function that minimises the error $f(\vec{x}, t; \theta)$ on the PDE, initial and boundary conditions. Due to the way we sample the parameters, this method is still feasible in higher order dimensions, allowing pricing of multiple correlated options simultaneously. The PDE model is initially discussed in [20], and further extended in [22]. We implement both techniques, and conduct an analysis of the impact of various network architectures on the solution.

5.1. Algorithm

As in [20], we consider a parabolic PDE in d spatial dimensions

$$\frac{\partial V}{\partial t}(t, \vec{x}) + \mathcal{L}V(t, \vec{x}) = 0, \qquad (t, \vec{x}) \in [0, T] \times \Omega, \tag{5.1}$$

$$V(0, \vec{x}) = h(\vec{x}), \tag{5.2}$$

$$V(t, \vec{x}) = g(t, \vec{x}), \qquad \vec{x} \in \partial\Omega, \tag{5.3}$$

where $\vec{x} \in \Omega \subset \mathbb{R}^d$ and $\mathcal{L}$ is any differential operator satisfying the requirements of a parabolic PDE. The goal is to construct a neural network $f(t, \vec{x}; \theta)$ with $\theta \in \mathbb{R}^K$ the parameters of the network, which satisfies the PDE differential operator, initial condition and boundary conditions. To turn the PDE into an optimisation problem on which we can apply stochastic gradient descent, we consider the loss function

$$J(f) = \left\| \frac{\partial f}{\partial t}(t, \vec{x}; \theta) + \mathcal{L}f(t, \vec{x}; \theta) \right\|^2_{[0,T]\times\Omega,\nu_1} + \left\| f(t, \vec{x}; \theta) - g(t, \vec{x}) \right\|^2_{[0,T]\times\partial\Omega,\nu_2} + \left\| f(0, \vec{x}; \theta) - h(\vec{x}) \right\|^2_{\Omega,\nu_3}. \tag{5.4}$$

In (5.4) we denote

$$\|f(y)\|^2_{\mathcal{Y},\nu} = \int_{\mathcal{Y}} f(y)\nu(y)dy, \tag{5.5}$$

where $\nu(y)$ is a probability density function defined on $\mathcal{Y}$. Because of the way we defined the loss function (5.4) we are free to choose the probability density functions $\nu_i, i = 1, 2, 3$. For instance, we can choose a distribution which attempts to fill the parameter space Ω as evenly as possible, using a technique such as latin hypercube sampling [27]. Alternatively, a distribution can be obtained by fitting

on historical data. Notice that in contrary to classical methods, in which we often form a mesh of the state space, we only sample from the state space according to a distribution. This avoids exponential complexity when increasing the dimension of the state space. Using the sequence of sampled points, we can minimise $J(f)$ with traditional methods such as stochastic gradient descent.

We list the procedure for the algorithm from [20]:

(i) Using densities ν_1, ν_2 and ν_3, we sample the points (t_n, x_n) from $[0,T]\times\Omega$, (τ_n, z_n) from $[0,T]\times\partial\Omega$ and ω_n from Ω. Denote with $s_n = \{(t_n, x_n), (\tau_n, z_n), \omega_n\}$ the set of sampled points.

(ii) Using the network $f(t,x;\theta)$, we calculate the loss of f on the sampled points

$$G(\theta_n, s_n) = \left(\frac{\partial f}{\partial t}(t_n, x_n, \theta_n) + \mathcal{L}f(t_n, x_n, \theta_n)\right)^2 + \Big(f(\tau_n, z_n; \theta_n) - g(\tau_n, z_n)\Big)^2 + \Big(f(0, \omega_n, \theta_n) - h(\omega_n)\Big)^2. \tag{5.6}$$

(iii) We calculate the gradient of $G(\theta_n, s_n)$ and take a gradient descent step at the point s_n

$$\theta_{n+1} = \theta_n - \alpha_n \nabla_\theta G(\theta_n, s_n). \tag{5.7}$$

(iv) Repeat for $n = 1, \dots$, until the loss is minimised sufficiently.

5.2. Computations of Second Derivatives

The PDE from (5.1) contains second order derivatives in the differential operator $\mathcal{L}$. As an example, consider the multidimensional Black-Scholes PDE (3.33), in which the differential operator in d dimensions with interest rate r, correlations ρ_{ij} and volatilities σ_i reads

$$\mathcal{L}_x f(t,x;\theta) = rf(t,x;\theta) - \sum_{i=1}^{d}\left(r - \frac{\sigma_i^2}{2}\right)\frac{\partial f}{\partial x_i}(t,x;\theta) - \sum_{i,j=1}^{d}\frac{\rho_{ij}\sigma_i\sigma_j}{2}\frac{\partial^2 f}{\partial x_i \partial x_j}(t,x;\theta). \tag{5.8}$$

Computation of the operator $\mathcal{L}$ requires us to evaluate d^2 second derivatives. When increasing the batch size and the complexity of the network, it becomes computationally more expensive to evaluate derivatives. This problem increases in complexity further due to the requirement of the gradient for SGD, effectively requiring the third-order derivatives

$$\nabla_\theta \frac{\partial^2 f}{\partial x^2}(t,x;\theta). \tag{5.9}$$

Instead of directly evaluating the derivatives, a Monte-Carlo scheme is developed in [20] used to approximate $\nabla_\theta G(\theta_n, s_n)$, which we present in this section. Suppose that the second derivatives of the differential operator $\mathcal{L}$ are of the form

$$\frac{1}{2}\sum_{i,j=1}^{d}\rho_{ij}\sigma_i\sigma_j\frac{\partial^2 f}{\partial x_i \partial x_j}(t,x;\theta), \tag{5.10}$$

with $[\rho_{i,j}]_{i,j=1}^d$ a positive definite correlation matrix and $\sigma = (\sigma_i, \dots, \sigma_d)$ diffusion coefficients. Then,

$$\sum_{i,j=1}^{d}\rho_{ij}\sigma_i\sigma_j\frac{\partial^2 f}{\partial x_i \partial x_j}(t,x;\theta) = \lim_{\Delta\to 0}\mathbb{E}\left[\sum_{i,j=1}^{d}\frac{\frac{\partial f}{\partial x_i}(t, x+\sigma W_\Delta;\theta) - \frac{\partial f}{\partial x_i}(t,x;\theta)}{\Delta}\sigma_i W_\Delta^i\right], \tag{5.11}$$

with $W_t \in \mathbb{R}^d$ a Brownian motion and Δ discretisation step size. Using Taylor expansion, one finds that the convergence rate is $\mathcal{O}(\sqrt{\Delta})$. Let

$$G_1(\theta_n, s_n) := \left(\frac{\partial f}{\partial t}(t_n, x_n, \theta_n) + \mathcal{L}f(t_n, x_n, \theta_n)\right)^2, \tag{5.12}$$

the first part of the loss function (5.6). Define the first order differential operator, by subtracting the second order terms, as

$$\mathcal{L}_1 f(t_n, x_n; \theta_n) := \mathcal{L} f(t_n, x_n; \theta_n) - \frac{1}{2} \sum_{i,j=1}^{d} \rho_{ij} \sigma_i \sigma_j \frac{\partial^2 f}{\partial x_i \partial x_j}(t, x; \theta). \tag{5.13}$$

We can approximate G_1 by

$$\tilde{G}_1(\theta_n, s_n) := 2 \left(\frac{\partial f}{\partial t}(t_n, x_n; \theta_n) + \mathcal{L}_1 f(t_n, x_n; \theta_n) + \sum_{i,j=1}^{d} \frac{\frac{\partial f}{\partial x_i}(t, x + \sigma W_\Delta; \theta) - \frac{\partial f}{\partial x_i}(t, x; \theta)}{\Delta} \sigma_i W_\Delta^i \right) \tag{5.14}$$

$$\times \nabla_\theta \left(\frac{\partial f}{\partial t}(t_n, x_n; \theta_n) + \mathcal{L}_1 f(t_n, x_n; \theta_n) + \sum_{i,j=1}^{d} \frac{\frac{\partial f}{\partial x_i}(t, x + \sigma \tilde{W}_\Delta; \theta) - \frac{\partial f}{\partial x_i}(t, x; \theta)}{\Delta} \sigma_i \tilde{W}_\Delta^i \right), \tag{5.15}$$

where $Cov[W_\Delta^i, W_\Delta^j] = \rho_{ij}\Delta$, W_Δ and $\tilde{W}_\Delta$ are independent with identical distribution. Using $\tilde{G}_1$, we obtain the approximated gradient of the loss function as

$$\nabla_\theta \tilde{G}(\theta_n, s_n) = \tilde{G}_1(\theta_n, s_n) + \nabla_\theta \Big(f(\tau_n, z_n; \theta_n) - g(\tau_n, z_n) \Big)^2 + \nabla_\theta \Big(f(0, \omega_n, \theta_n) - h(\omega_n) \Big)^2. \tag{5.16}$$

The procedure of approximating the second-order partial derivatives in the PDE using Monte-Carlo simulation when computing the gradient becomes beneficial in higher dimensions, where otherwise computation time would become too expensive.

5.3. Multidimensional Black-Scholes European Option Pricing

In this section, we apply the PDE model on a European Black-Scholes option pricing problem in 2 dimensions. The multidimensional Black-Scholes PDE is derived in section (3.2.2) and a Gauß-Hermite reference pricing technique is constructed in section (3.2.3). We fix $\vec{S}_0 = (1,1)^T$, $K = 1$, $r = 0.1$, $\sigma_1 = 0.25$, $\sigma_2 = 0.4$ and $\rho = 0.6$. We set $\Omega = [0,4]$, and sample (t_n, x_n) from $[0,2] \times \Omega$. The reason for analysing this problem in 2 dimensions is twofold. First, we can still visualise the approximations in 2 dimensions, which allows for a better intuitive explanation for the results. Second, direct integration using Gauß-Hermite polynomials (3.2.3) is very accurate in lower dimensions, and takes away the need for Monte-Carlo evaluation, which is computationally more expensive and less accurate.

We train each model on the same parameter set for 1000 epochs. Each epoch contained 20 steps, each with a batch size of 5000. A combined total of 100 million points were sampled to train every model. We adjust the learning rate according to a piecewise decaying function

$$\alpha_n = \begin{cases} 10^{-4} & n \leq 4,000 \\ 5 \times 10^{-5} & 4,000 < n \leq 8,000 \\ 10^{-5} & 8,000 < n \leq 12,000 \\ 5 \times 10^{-6} & 12,000 < n \leq 16,000 \\ 10^{-6} & n > 16,000 \end{cases}, \tag{5.17}$$

where n denotes the amount of steps. We found that this learning rate schedule was more effective than others (we considered exponential decay, polynomial decay). A piecewise learning rate also aligns with the findings in [20]. We allocate exactly twenty CPU cores from DHPC [13] (Intel Xeon 3.0GHz) for each PDE model. Using the same resources across the PDE model training processes, we can compare the training time.

We consider two error measures. In the first one, we partition the state space and evaluate the model on this partition with $t = 0$. We compare the resulting surface with the Gauß-Hermite solution and calculate the mean squared error over the surface. The second measure we compute relates to the error reported in [20], in which the relative error $f(0, \vec{S}_0; \theta)$ with respect to the reference pricing technique is computed. In this case, $\vec{S}_0 = (1,1)^T$. Denote with V_R the reference pricing approximation.

This relative error is computed as

$$L(f, V_R) = \frac{|f(0, \vec{S}_0) - V_R(0, \vec{S}_0)|}{|V_R(0, \vec{S}_0)|} \cdot 100 \tag{5.18}$$

For all the networks trained, we found that using the hyperbolic tangent $tanh$ function as activation function yielded superior performance in terms of training loss compared to $ReLU$ and the sigmoid function. Additionally, the evaluation speed of each network is almost identical, regardless of the different architecture. Since the network allows for vectorised inputs, computation of multiple samples simultaneously does not increase evaluation time significantly. This on the other hand is not the case for the Gauß-Hermite solution, where each grid point has to be calculated sequentially. We find that for the evaluation of 2500 grid points using a network takes approximately 300 milliseconds. Since the aim of this thesis is not to compare the network evaluation speed to reference pricing techniques, we only list this value to emphasise the evaluation speed of a neural network.

For the Black-Scholes problems, we consider the payoff function to be a basket call option, e.g.

$$V(T, S) = \max\Big\{\frac{1}{d}\sum_{i=0}^{d} S_i(T) - K, 0\Big\}. \tag{5.19}$$

This function is chosen arbitrarily. In section (5.4) we select another payoff function, namely the call on min payoff, e.g.

$$V(T, \vec{S}) = \max\{\min\{S_1(T), ..., S_d(T)\} - K, 0\}. \tag{5.20}$$

5.3.1. MLP Network

We train an MLP network architecture with the configuration as shown in table (5.1).

	MLP
Layers	4
Nodes Per Layer	75
Total Parameters	$23,176$
Activation Function	$tanh$
Initialiser	Glorot Normal

Table 5.1: The configuration of the trained MLP network on the Black-Scholes European call price problem.

In figure (5.1) the approximation of the MLP network on the two dimensional Black-Scholes PDE is shown on the left, in comparison to the Gauß-Hermite solution on the right at $t = 0$ on a grid of 50×50 points. We notice visual differences in the surface between the two approximators. The MLP approximation is not smooth and has some irregularities around $[0, 0.5] \times [0, 0.5]$.

Aside from the inability to represent the smooth price around the ATM values, the prices for ITM values are relatively accurate. We find the error measures of the MLP compared to the Gauß-Hermite approximation listed in table (5.2). Indeed, the relative error at the initial value $\vec{S}(t_0) = \vec{S}_0 = (1, 1)^T$ is very large and provides insufficient accuracy.

Error Type	Value
MSE on Surface	$1.04 \cdot 10^{-3}$
Relative Initial Value Error	13.78%
Training Time (H)	0.20

Table 5.2: MLP errors compared to the Gauß-Hermite approximation.

5.3.2. Highway Network

Next, we train a highway network with similar configuration as the MLP, shown in table (5.3). Due to the increased complexity of the highway layers, the amount of parameters is roughly doubled.

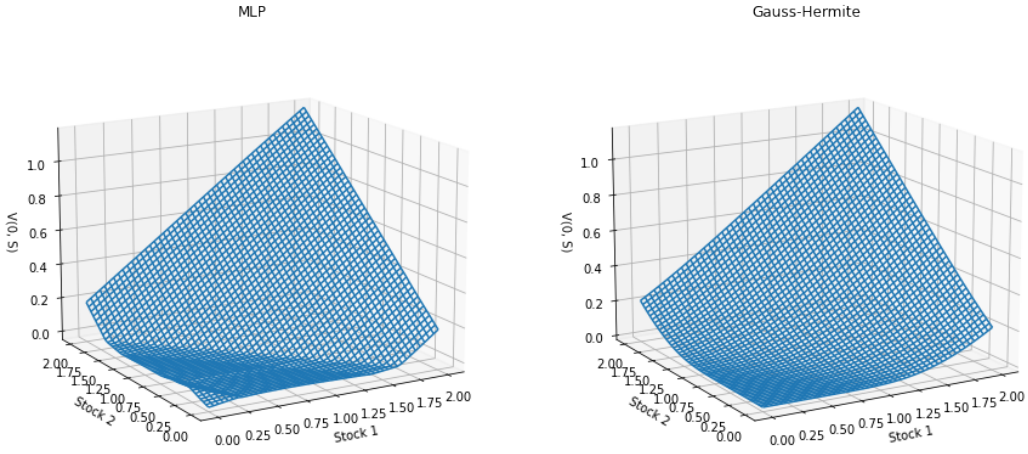

Figure 5.1: The approximation of the MLP network on the two dimensional Black-Scholes PDE on the left, in comparison to the Gauß-Hermite solution on the right at $t = 0$ on a grid of 50×50 points.

	Highway
Layers	4
Nodes Per Layer	75
Total Parameters	49,675
Activation Function	$tanh$
Transform Activation Function	$tanh$
Initialiser	Glorot Normal

Table 5.3: The configuration of the trained highway network on the Black-Scholes European call price problem.

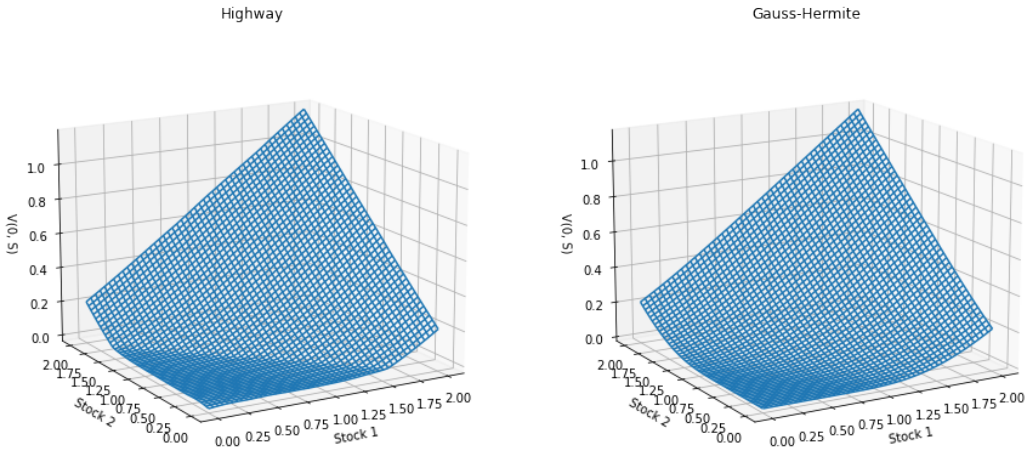

Figure 5.2: The approximation of the highway network on the two dimensional Black-Scholes PDE on the left, in comparison to the Gauß-Hermite solution on the right at $t = 0$ on a grid of 50×50 points.

The approximation of the Highway network against the Gauß-Hermite approximation is visualised in figure (5.2). We see that the irregularities present in the MLP approximation (5.1) have disappeared, but the smoothness around the ATM values remains a problem.

We find that the highway network considerably reduces both the relative initial value error as well as the MSE on the surface. The values are listed in table (5.4). Due to the non-smoothness around the ATM values, we still find a high value for the relative initial value error of 3.79%.

Error Type	Highway
MSE on Surface	$2.95 \cdot 10^{-4}$
Relative Initial Value Error	3.79%
Training Time (H)	0.97

Table 5.4: Highway errors compared to the Gauß-Hermite approximation.

5.3.3. DGM Network

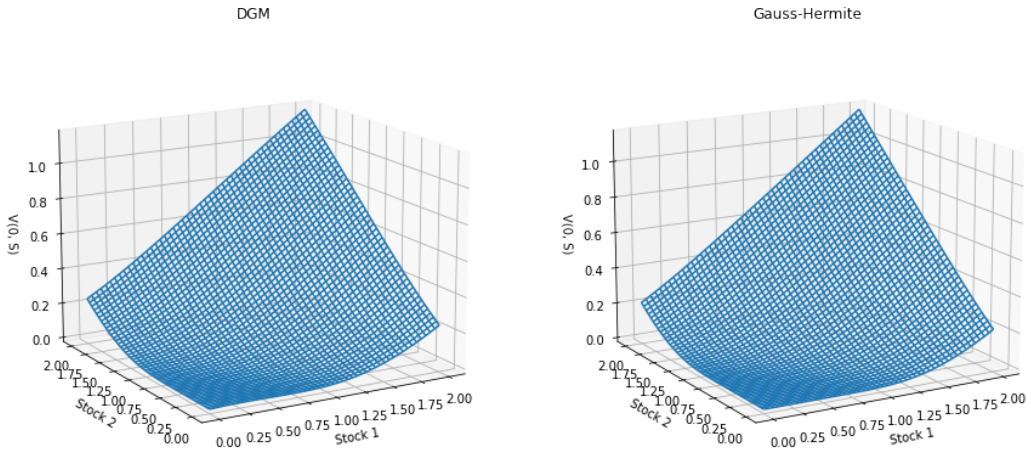

Figure 5.3: The approximation of the DGM network on the two dimensional Black-Scholes PDE on the left, in comparison to the Gauß-Hermite solution on the right at $t = 0$ on a grid of 50×50 points.

The final network we train on the two dimensional Black-Scholes European call option PDE is the DGM network. The configuration is given in table (5.5)

	DGM
Layers	4
Nodes Per Layer	75
Total Parameters	$95,332$
Activation Function	$tanh$
Initialiser	Glorot Normal

Table 5.5: The configuration of the trained DGM network on the Black-Scholes European call price problem.

In section (4.4) we considered the Black-Scholes one dimensional call option as well as the implied volatility problem, where we found that the DGM network requires more computational power without yielding significantly better results than the generalised highway network. The network was initially designed in [20] for approximating PDE's. Indeed, we find that the DGM network has the ability to approximate the smooth behaviour of the arithmetic mean, in contrast to the MLP and highway network. Figure (5.3) illustrates this ATM smoothness.

Error Type	MLP	Highway	DGM
MSE on Surface	$1.04 \cdot 10^{-3}$	$2.95 \cdot 10^{-4}$	$2.35 \cdot 10^{-4}$
Relative Initial Value Error	13.78%	3.79%	2.75%
Training Time (H)	0.20	0.97	1.42

Table 5.6: MLP, highway & DGM errors compared to the Gauß-Hermite approximation.

Despite the DGM network's ability to approximate the smooth surface, we find an MSE over the surface that is relatively comparable to the highway network. The DGM network required 46.4% more computation time, whereas the MSE over the surface only decreased by 20.0%.

5.3.4. Training Loss

During training, we noticed interesting behaviour in all networks. Specifically, the training loss at certain epoch ranges were fluctuating a lot, as shown in figure (5.4). The loss spikes are caused by high fluctuations in the loss on the terminal condition. The MLP was most susceptible to this instability. Additionally, the fluctuations often happened before the learning rate was decreased according to the piecewise decay (equation 5.17). This observation might suggest that the learning rate was too high to optimise the terminal loss, leading to the fluctuations. After the learning rate is decreased, we again see a consistent reduction in loss. We find that adjusting the learning rate for the PDE models allows for a smaller error when training many epochs, as it prevents the loss from stagnating due to smaller gradient steps.

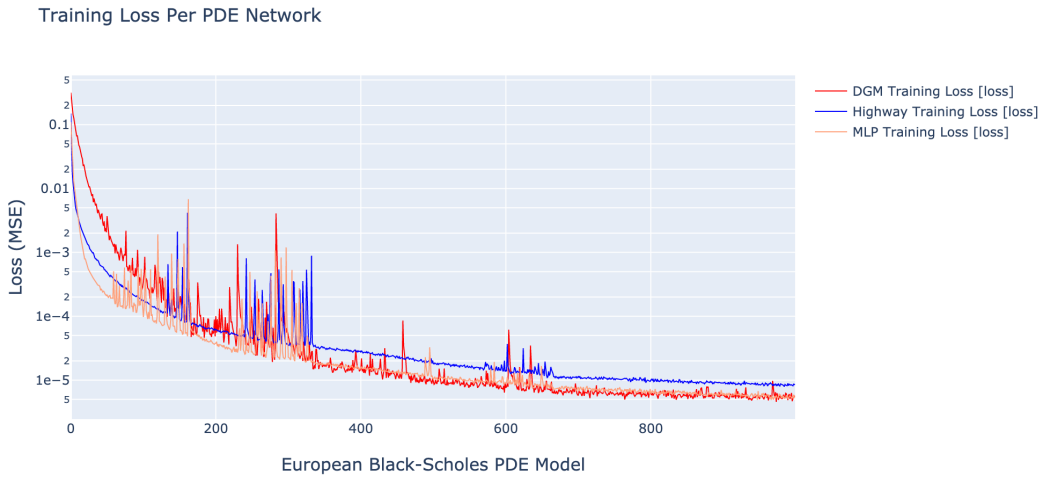

Figure 5.4: Training loss for the MLP, highway and DGM networks.

5.3.5. The Effects of Longer Training

We found in the previous sections that the DGM network had the ability to approximate the smoothness of the pricing surface, as seen in figure 5.3. However, we did not see a significant reduction in MSE over the entire surface. As an experiment, we select the highway network and DGM network from the previous section, and train both networks for an extended amount of cycles. Specifically, we employ a batch size of 10000, use 5000 epochs each containing 20 steps, with a learning rate schedule of

$$\alpha_n = \begin{cases} 10^{-4} & n \leq 20,000 \\ 5 \times 10^{-5} & 20,000 < n \leq 40,000 \\ 10^{-5} & 40,000 < n \leq 60,000 \\ 5 \times 10^{-6} & 60,000 < n \leq 80,000 \\ 10^{-6} & n > 80,000 \end{cases}, \tag{5.21}$$

where n is the total amount of steps. We sample a total of $10000 \cdot 5000 \cdot 20 = 1 \cdot 10^9$ points.

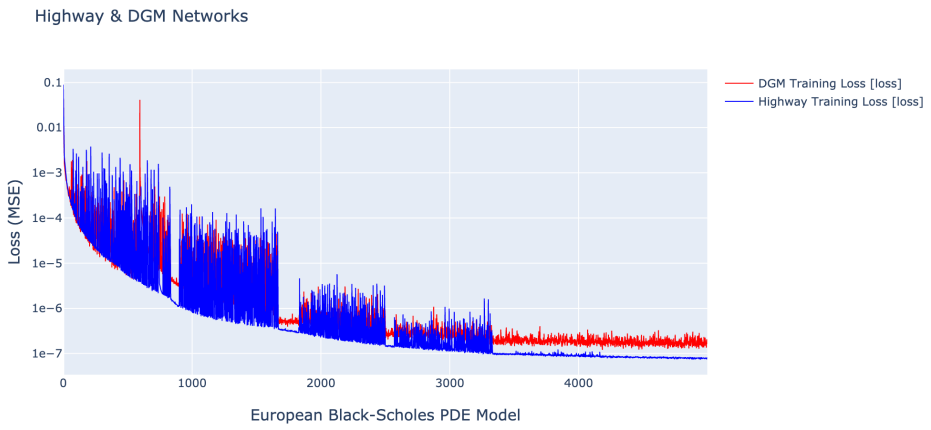

Figure 5.5: Training loss for the highway and DGM networks on $1 \cdot 10^9$ simulation points, spread out over 5000 epochs.

When comparing the two training loss plots (figures 5.4 and 5.5) we see that the training loss keeps decreasing as we simulate more points, which is expected, as the simulation space is sampled more often. Aside from this observation, we again find high loss spikes in the terminal loss condition during training. This instability is likely caused by the learning rate being too high to reduce the loss further.

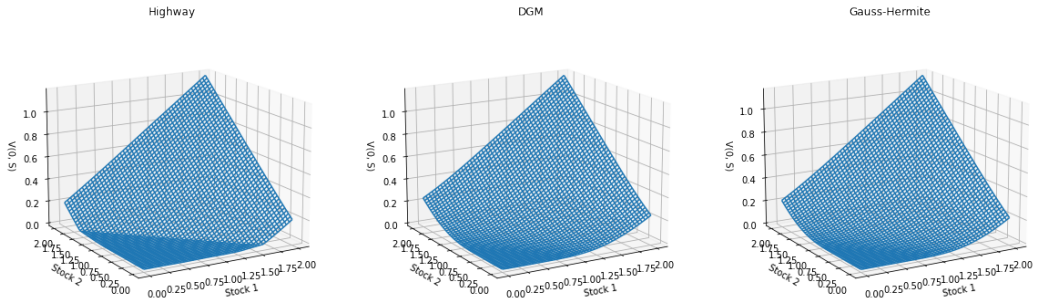

Figure 5.6: The surface approximation of the long trained Highway and DGM networks compared to the Gauß-Hermite reference pricer.

Interestingly, when comparing the performances of the networks on the price surface at $t = 0$ (figure 5.6), we do not find any improvements for the highway network. We can conclude that this network architecture is not capable of incorporating the smoothness of the pricing surface, despite long training. Table 5.7 shows the results of the network errors. We note that both errors have increased for both networks. The highway MSE surface error has doubled (100.3%), and the DGM MSE surface error has also increased by 13.6%.

Intuitively, more training should yield more accurate predictions, which is clearly not the case for these models. The reduction of accuracy can be explained. In the PDE model, the network attempts to

Error Type	Highway	DGM
MSE on Surface	$5.91 \cdot 10^{-4}$	$2.67 \cdot 10^{-4}$
Relative Initial Value Error	8.44%	5.84%
Training Time (H)	4.99	16.23

Table 5.7: Long trained highway & DGM errors compared to the Gauß-Hermite approximation.

optimise the loss, which is a sum of the interior loss, the boundary loss and the terminal loss (equation 5.6). In the European Black-Scholes PDE, we do not have a boundary condition and therefore the network only needs to optimise the interior and the terminal loss. In figure 5.7 the terminal and interior loss of the DGM network are visualised. The terminal loss is consistently higher than the interior loss. Therefore, SGD algorithm attempts to optimise the terminal loss since it accounts for the majority of the loss. This process could decrease the accuracy of the interior approximation.

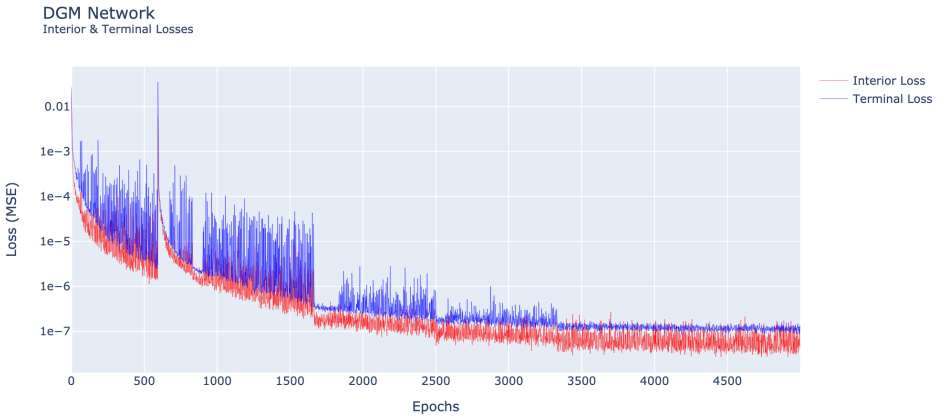

Figure 5.7: The training loss of the long trained DGM network. The interior loss (red) and terminal loss (blue) over 5000 epochs. The scale is log-transformed.

To validate this assumption, we evaluate both the DGM network from section 5.3.3 as well as the longer trained DGM network on $t = T$. The results are presented in table 5.8. Indeed, the longer trained DGM network performs significantly better on the terminal condition, while performing less accurate than the 1000-epoch DGM on the interior. We define the absolute terminal error as

$$L(f, V_R) = |f(T, \vec{S}_T) - V_R(T, \vec{S}_T)|, \tag{5.22}$$

with V_R denoting the reference pricer.

Error Type	DGM (1000 Epochs)	DGM (5000 Epochs)
MSE on Surface	$1.36 \cdot 10^{-5}$	$7.86 \cdot 10^{-7}$
Absolute Terminal Error ($S_T = (1,1)^T$)	$1.51 \cdot 10^{-2}$	$5.17 \cdot 10^{-3}$

Table 5.8: The errors of the DGM network trained on 1000 epochs compared to those of the DGM network trained on 5000 epochs.

A solution to this problem could be to introduce a hyperparameter η, which scales the terminal loss appropriately, in order to prevent the network from losing accuracy on the interior loss. Equation 5.4 would then become

$$\tilde{J}(f) = \left\| \frac{\partial f}{\partial t}(t, \vec{x}; \theta) + \mathcal{L}f(t, \vec{x}; \theta) \right\|^2_{[0,T]\times\Omega,\nu_1} + \left\| f(t, \vec{x}; \theta) - g(t, \vec{x}) \right\|^2_{[0,T]\times\partial\Omega,\nu_2} + \eta \left\| f(0, \vec{x}; \theta) - h(\vec{x}) \right\|^2_{\Omega,\nu_3}. \tag{5.23}$$

The value of η could be decreased over consecutive epochs, just like we do for the learning rate α.

5.3.6. Increasing Dimensionality

In this section we increase the dimensionality of the Black-Scholes pricing problem, and train a MLP, highway and DGM network on the 20-dimensional problem. Table 5.9 lists the configuration of each network. The learning rate for each network is given in equation 5.17.

	MLP	Highway	DGM
Epochs	1000	1000	1000
Batch Size	5000	5000	5000
Steps Per Epoch	20	20	20
Layers	4	4	4
Nodes Per Layer	200	100	100
Total Parameters	$165,401$	$120,418$	$199,543$
Activation Function	$tanh$	$tanh$	$tanh$
Initialiser	Glorot Normal	Glorot Normal	Glorot Normal
Payoff	Arithmetic Mean	Arithmetic Mean	Arithmetic Mean

Table 5.9: Configuration of the MLP, highway & DGM networks on the high dimensional Black-Scholes problem.

Compared to the networks in the two-dimensional case, we increased the amount of parameters significantly in order to allow the network to incorporate more non-linearity for the increased number of inputs. Since direct integration using Gauß-Hermite is not feasible in 20 dimensions, we evaluate the approximations using Monte-Carlo simulation. To this end, we uniformly sample n space points from the space set Ω, and compute the MSE on this set for all the networks. The relative initial value error measure remains identical. Table 5.10 shows the results of both error measures.

Error Type	MLP	Highway	DGM
MSE on MC samples ($n = 100$)	$1.04 \cdot 10^{-3}$	$1.3 \cdot 10^{-3}$	$6.9 \cdot 10^{-5}$
Relative Initial Value Error	7.17%	6.49%	4.41%
Training Time (H)*	0.45	2.21	4.79

* Each network was trained on 24 CPU cores (Intel Xeon 3.0GHz) on DHPC [13].

Table 5.10: MLP, highway & DGM errors with Monte-Carlo simulation as reference. We take $n = 100$ uniform space samples and compute the MSE ($d = 20$).

Interestingly, the DGM network seems to be unaffected by increases in dimensionality when considering the MSE. The MSE of the highway and MLP networks is slightly increased compared to the two-dimensional case (table 5.6). It seems like the DGM network can provide accurate approximations even when the dimensionality is increased, as long as the network size is increased appropriately. This observation aligns with the results found in [20, Table 1], where the relative initial value errors are reported to be even lower for the 20-dimensional case than the 3-dimensional case. Possibly, the networks can learn some form of consistency in higher dimensional spaces which cause the approximations to remain consistent.

5.4. Multidimensional Heston European Option Pricing

In the previous section, we developed PDE models for the multidimensional Black-Scholes pricing PDE. In this section, we train the same networks on the Heston pricing PDE, which is a more involved pricing PDE, and evaluate their performance. In section 3.4.2 we derive the multidimensional Heston PDE, and in section 3.4.3 we present a Monte-Carlo scheme for the multidimensional Heston model as well as a Fourier based approximation method. For this problem, we use as payoff function the minimum of the underlying assets.

$$V(T, \vec{S}) = \max\{\min\{S_1(T), \dots, S_d(T)\} - K, 0\}. \tag{5.24}$$

5.4.1. Network Results

In this section, we present the results of the trained network architectures on the Heston PDE model. For this problem, we use the same network configurations as in the Black-Scholes problem (section 5.3).

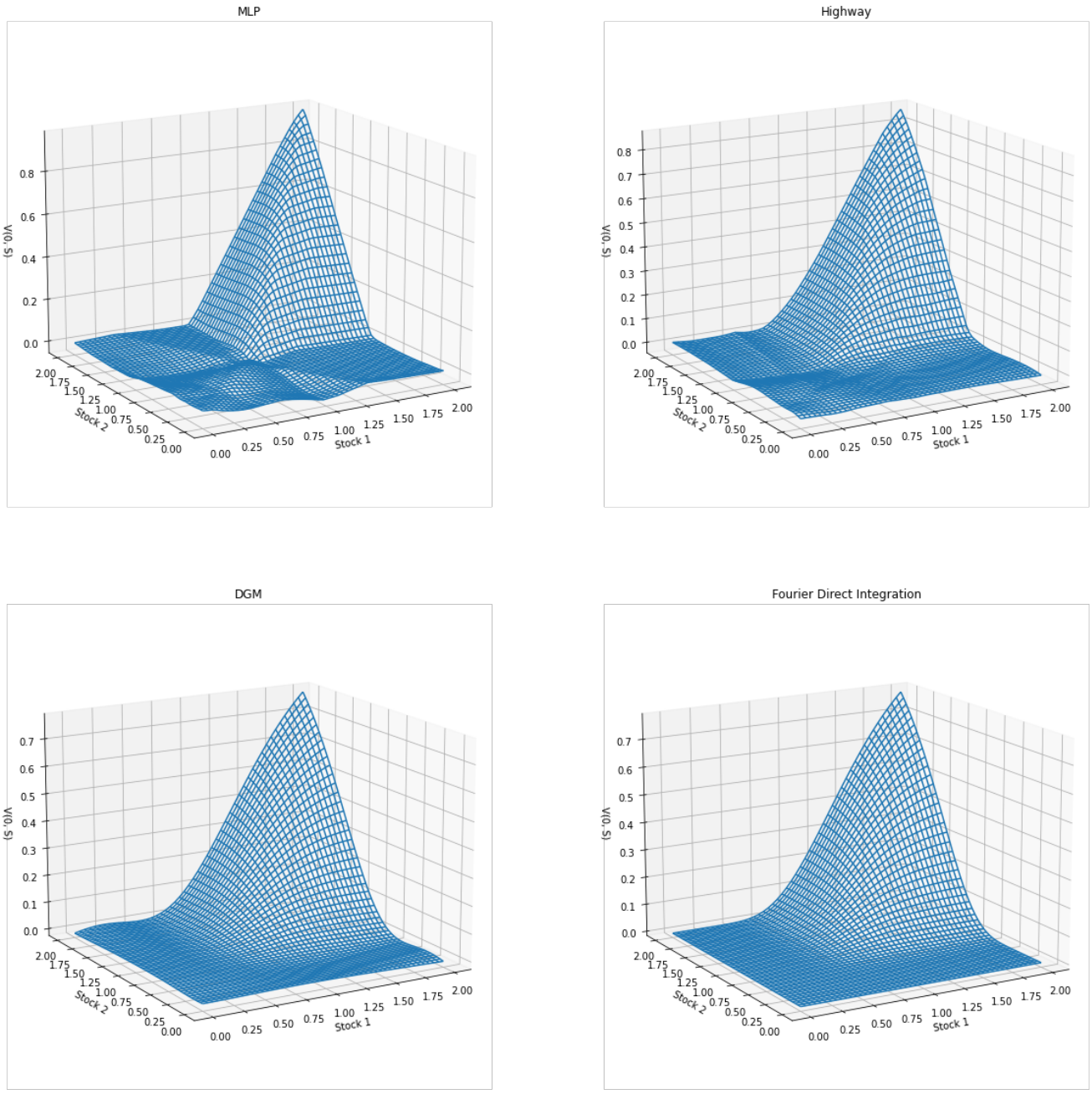

Figure 5.8: The approximation of the MLP network (top left), highway network (top right), DGM network (bottom left) and the Fourier reference pricer (bottom right) on the two dimensional Heston PDE at $t = 0$ on a grid of 50×50 points.

For each network, we visualise the approximation of the pricing surface in figure 5.8. We notice again that the MLP network is inadequate in approximating the smoothness of the solution. The highway network, on the other hand, seems to incorporate this smoothness slightly better than in the Black-Scholes case (section 5.3.2). We do, however, notice another problem occurring in the Heston model which we did not see in the Black-Scholes case, namely the prediction of negative values for the option price. This violates the no-arbitrage bounds: the value of an option should always be nonnegative. Hence, prediction of negative values should be punished by the loss function, as we know for sure this will not be a sensible prediction. Of course, the PDE itself should already punish negative predictions, but we could increase this punishment by adding a loss term to equation 5.4 such as

$$G_4(\theta) := \beta \max\{-f(t, \vec{x}; \theta), 0\}, \tag{5.25}$$

with β an additional regularisation hyperparameter for the PDE model. G_4 incorporates the lower no-arbitrage bound in the PDE.

Table 5.11 lists the results of the MSE surface and relative initial errors for the networks on the Heston PDE model.

Error Type	MLP	Highway	DGM
MSE on Surface	$1.90 \cdot 10^{-3}$	$4.81 \cdot 10^{-4}$	$9.04 \cdot 10^{-6}$
Relative Initial Value Error	129.38%	55.89%	3.21%
Training Time (H)	0.45	2.04	3.01

Table 5.11: Network errors of the Heston PDE compared to the Fourier approximation.

We make the following observations:

- The increased complexity of the Heston PDE causes both the MLP and the highway network to predict the initial value very poorly. Indeed, the MLP initial value price approximation is off by more than double the value of the Fourier approximation.
- Both the MLP and the highway network have a tendency to overestimate the price. This can be seen especially in figure 5.8. The predictions of the MLP and highway network both exceed 0.8, whereas the DGM network and the Fourier solution do not exceed 0.75. This error likely contributes to the high MSE on the pricing surface for both networks.
- The DGM network is not susceptible to the previous two observations. Even though the PDE model is increased in complexity, the DGM network only increased 0.46% in initial value error compared to the Black-Scholes PDE. Additionally, the MSE on the surface is considerably lower. Possibly, the increased accuracy on the surface is related to the change in payoff function (arithmetic mean v.s. minimum). This would imply that the performance of the network is influenced by the terminal condition. When examining the terminal loss of this payoff function (figure 5.9), we see much smaller fluctuations compared to the Black-Scholes arithmetic payoff (figure 5.4), which supports this statement.
- Due to the more complex model, the training times for identical size networks, hyperparameters and sample sizes takes roughly twice as long. This is likely a direct consequence of the increased amount of partial derivatives appearing in the multidimensional Heston PDE (equation 3.79). Increasing the complexity of the PDE, in particular adding partial derivatives, increases the computational complexity of the model.

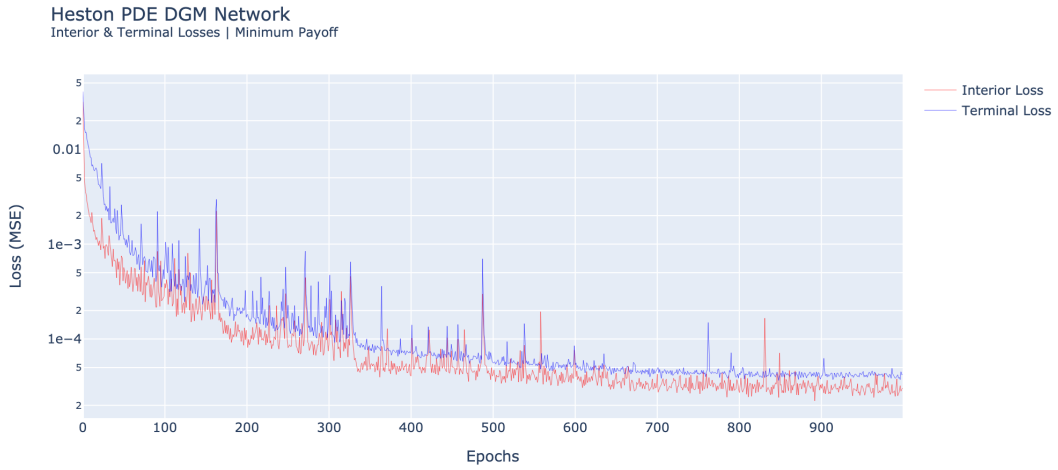

Figure 5.9: The interior and terminal loss of the Heston PDE DGM network over 1000 epochs.

5.5. Parametric Models

In the previous section we trained multidimensional PDE models where we fixed the parameters. For instance, in the case of the Black-Scholes PDE model (section 5.3), we fixed $r, \sigma_1, \dots, \sigma_d$ as well as the correlations $\{\rho_{ij} : 1 \leq i \leq d, 1 \leq j \leq d\}$. In the market, these values are continuously changing, which requires us to retrain a PDE model for each scenario and interpolate the solution between the fixed models. This procedure requires a considerable amount of pre-trained fixed models and introduces another approximation error. In this section, we examine parametric PDE models [22]. In such models, we treat the parameters of the pricing PDE as inputs to the network, and simulate them identically to the space and time inputs. The resulting network is increased in dimensionality, since each price process requires a volatility variable and correlation variables to other price processes, adding to the dimensionality. In the following section, we focus on an efficient input set of parameters to the network, where the goal is to keep the dimensionality as low as possible for training convergence purposes. This concept can be applied to any PDE, in particular to multidimensional Heston processes, including the ones with multiple variance processes. The additional inputs that must be supplied are the parameters of each variance process as well as the correlations with respect to the underlying and variance processes.

5.5.1. Parametrisation of the PDE

We denote with $\mathcal{P}$ the space of parameters. The vector $\mu \in \mathcal{P}$ should contain all the parameters we require to model a generalised multidimensional Black-Scholes model. In this problem, we have the risk free rate r, volatilities σ_i and the correlations between the assets ρ_{ij}. One might suggest that the strike price K should also be contained in μ. However, as noted in [22], we can rescale the asset prices to obtain a fixed strike K. We can therefore omit K as a parameter. The correlation matrix can be simplified by parametrising it as a Cholesky decomposition, resulting in a triangular matrix with $d(d-1)/2$ factors. The parameter vector μ is then given as

$$\mu = (r, \sigma_1, \dots, \sigma_d, l_{ij}), \tag{5.26}$$

where l_{ij} are the nonzero Cholesky factors of the decomposed correlation matrix.

An alternative way to parametrise the correlation matrix suggested in [22], is to use the practical approach where we provide independent pairwise correlations $\hat{\rho}_i = \rho_{i,i+1}$ to the network. We then compute the missing entries of the correlation matrix by taking the products

$$\rho_{ij} = \rho_{ji} = \prod_{k=i}^{j-1} \hat{\rho}_k, j > i. \tag{5.27}$$

This technique is fully explained in [29]. Using the pairwise correlations, we can reduce the parameter vector μ to

$$\mu = (r, \sigma_1, \dots, \sigma_d, \hat{\rho}_1, \dots, \hat{\rho}_d), \tag{5.28}$$

5.5.2. Parametric Black-Scholes PDE

We again consider the multidimensional Black-Scholes PDE with $d = 2$. For this problem, our vector μ is parametrised as

$$\mu = (r, \sigma_1, \sigma_2, \rho),\ r \in [0, 0.1], \sigma_1, \sigma_2 \in [0.01, 0.2], \rho \in [-0.5, 0.5]. \tag{5.29}$$

Due to the increased complexity of a parametric PDE model, we increase the amount of nodes, layers and epochs for all the networks. Specifically, we employ the values listed in table 5.12.

Due to the additional computational resources required to train a parametric network alongside the superior performance of the DGM network which we observed in the non-parametric PDE models, we focus on a DGM network only.

Figure 5.10 visualises the approximation of the parametric DGM network on the surface, along with the absolute error in the right plot. We note that over the entire surface, the absolute error is at most 0.01. The error is highest at the ATM values where one of the initial value stock prices is low and the other is high. Since oftentimes we are interested in the ATM values, this is a considerable limitation in the network approximation, and should be addressed in order to provide reliable option prices.

Network Parameter	
Batch Size	5.000
Epochs	5.000
Steps Per Epoch	40
Nodes	100
Layers	4

Table 5.12: General configuration for the parametric PDE networks.

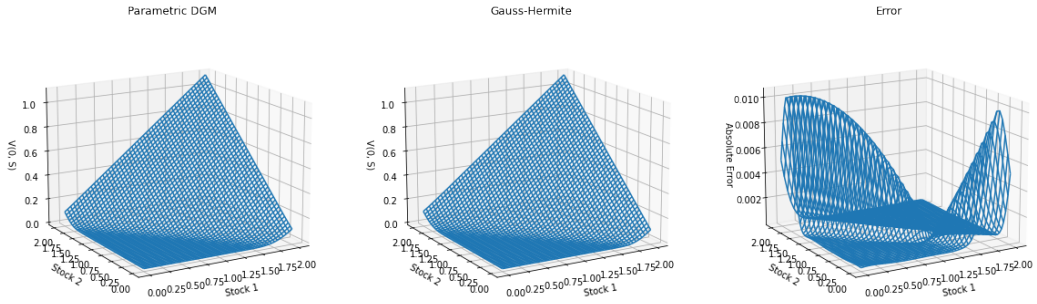

Figure 5.10: The parametric DGM network predictions compared to the Gauß-Hermite direct integration method. The parameters are $\mu = (0.05, 0.05, 0.07, 0.25)$. On the right, the absolute error is visualised.

Another issue we encounter is the approximation of the network on the boundaries of the parameter set. We consider the parameter set $\mu = (0.1, 0.2, 0.2, -0.5)$ which is on the boundary of the parameter domain $\mathcal{P}$. This is empirically one of the worst parameter sets we could find. Figure 5.11 visualises the approximation along with the error.

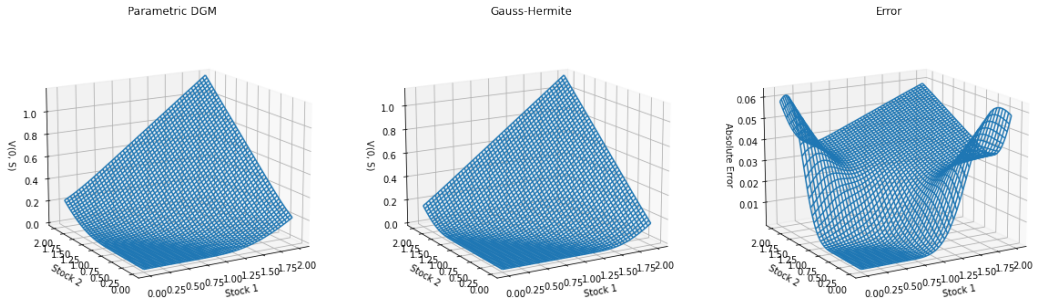

Figure 5.11: The parametric DGM network predictions compared to the Gauß-Hermite direct integration method. The parameters are $\mu = (0.1, 0.2, 0.2, -0.5)$. On the right, the absolute error is visualised.

We observe again that the errors are largest for ATM values where stock values are reaching their bounds. Additionally, we find an absolute error of 0.06, which is roughly 6 times larger than the absolute

error on the interior parameter domain. We could possibly solve this problem by training the network on a wider parameter domain. We adopt the idea from [25], in which a network is trained on a wide space domain, and only evaluated on a much smaller subset of this domain. It is shown that the procedure of training on a large set and evaluating on a subset leads to more accurate predictions. Since we sample points, sampling from a larger parameter domain would mean that the approximation is also continuous at the boundary, which is a desirable property. The downside of this approach is that the network will be trained on fewer samples from the smaller set of parameters, i.e. the set we will actually evaluate the network on. Longer training cycles could mitigate this problem.

6

Discussion & Conclusions

6.1. Discussion

The main results originate from chapters 4 and 5, in which we started by observing that neural networks can be used as a numerical approximation method in combination with a stochastic model, rather than as a standalone pricer trained on historical market data. We analysed neural network architectures in a supervised and semi-supervised setting, for which we will discuss the results separately.

6.1.1. Supervised Setting

In the supervised setting, we trained an MLP, highway networks and DGM networks including variants on three problems. We observed on all three problems that the generalised highway network architecture outperformed the other architectures in terms of MSE compared to computation time. In combination with the observation that network size positively influences performance of the network on all problems, we can conclude that a larger generalised highway network is likely the best performing network. In table 4.14 it is shown that the generalised highway network outperforms larger MLP's trained on more training cycles in [25] on the Black-Scholes European call option problem. The generalised highway network improved the MSE by 9.8% while reducing the network size by over 96% when comparing with the MLP from [25]. Figure 4.26 visualises the performance of the highway, generalised highway and DGM network with comparable amounts of parameters. It can be seen that the generalised highway network is superior in both MSE as well as computation time. Furthermore, we did not find consistent evidence of training loss to be a good indicator for testing performance. This is noticeable in figure 4.23, in which the DGM network has lower training loss than the generalised highway network on the training set, but higher on the test set, shown in figure 4.22.

We trained two variants of the DGM network. The first variant, named the deep DGM network, contained n additional non-linearities. We included this network to see if increasing non-linear complexity inside a DGM layer is beneficial. In second variant, which we named the no-recurrence network, omitted the input vector $\vec{x}$ in its calculations, effectively removing the recurrent pattern present in the DGM network. We trained this variant to quantify the effect of recurrence in the DGM network. Without the recurrent behaviour, the no-recurrence DGM network was similar to the generalised highway network, but contained more non-linearity in each layer. We found that in no case the MSE scores relative to the required computation time was superior to other networks for the deep DGM network. For the no-recurrence network, we found comparable MSE scores relative to the computation time to the generalised highway network. This was somewhat expected, as the architectures are similar, and the complexity of the implied volatility problem might benefit from the additional non-linearity. Further research should be conducted to analyse the performance of the no-recurrence network for other more complex problems. Since the DGM network was always outperformed by networks with no recurrent behaviour, we can conclude that on the considered problems, recurrence inside a network does not boost performance.

On the implied volatility problem, we observed poor performance across all networks. Not a single network managed to score a lower MSE than $6.6 \cdot 10^{-4}$ (table 4.16), translating to roughly 0.02 in MAE. This error makes use of networks as implied volatility numerical method infeasible. After transforming

the implied volatility dataset to include the scaled time value instead of the option value, we found significantly better performance. We reasoned in section 4.4.1 that this likely has to do with a large gradient, caused by the vega of the option value to be arbitrarily close to 0 in some cases. After this transformation, we found comparable results to the Black-Scholes and Heston European option pricing problems. Specifically, network size had a positive influence on the performance for all the networks, and the generalised highway network showed to be superior in terms of MSE relative to computation time. As mentioned, the no-recurrence network, with an MSE of $7.4 \cdot 10^{-8}$, was comparable to the generalised highway network, with an MSE of $2.2 \cdot 10^{-7}$, but a shorter training time of 0.88 against the 1.09 for the no-recurrence network. Table 4.17 lists the results of all the networks on the transformed implied volatility problem. In absolute terms, the deep DGM performed the best, with an MSE of $5.0 \cdot 10^{-8}$. However, in figure 4.31 the training time of this network can be seen to be considerably larger than the no-recurrence and generalised highway networks.

6.1.2. Semi-supervised Setting

We found in table 5.6 that the network architecture for the semi-supervised learning problems is significant for the accuracy of the approximations. In this setting, the DGM network outperformed the other architectures, likely due to its recurrent behaviour allowing for smooth and non-smooth approximations. The difference with respect to the approximations of other networks is visualised in figure 5.2. The main observation is that network architectures must allow for the prediction of both interior and boundary constraints separately. Furthermore, we did not find evidence of larger networks performing better on the semi-supervised learning problem. This is contrary to the observations in the supervised setting, in which a clear relationship between the network size and the performance could be noticed (i.e., figure 4.11).

In case of the Heston problem, which was slightly more complex due to the stochastic volatility assumption, we found that only the DGM network was able to provide acceptable approximations. Table 5.11 shows the MSE and relative initial value errors of the three considered networks. Both the MLP and highway network were unable to correctly approximate the reference pricer solution, with relative initial value errors of 129% and 55%, respectively. The DGM network did not suffer from the complexity of the Heston model and scored comparable to the Black-Scholes model, with a relative initial value error of 3.21%. This difference is possibly a result of the DGM architecture, allowing for the optimisation of all the constraints in de PDE simultaneously.

In section 5.3.5 we trained the MLP and DGM network for 5000 epochs, and compared their performance to identical networks trained on 1000 epochs. Interestingly, the performance of the networks decreased with longer training. We identified that the terminal loss is reduced at the cost of the interior loss, resulting in poorer approximations on the interior, which is of main interest. As a possible solution, we opted to introduce a hyper parameter η, which would serve as a weighing parameter of the terminal loss in the loss function, as in equation 5.23.

We found that all networks, with the DGM network in particular, showed consistent performance in higher dimensionality. Table 5.10 shows the relative initial value error and MSE for the MLP, highway and DGM network for a 20-dimensional Black-Scholes European basket call option with arithmetic payoff. Especially the relatively small DGM network manages to achieve a relative initial value error of 4.41, compared to the 2.75% in the 2-dimensional case. None of the networks provide sufficiently accurate approximations to be used in production, but appropriately longer training times will likely reduce these errors.

Finally, when considering the parametrised PDE models, we found that the training parameter set should be chosen wider than the evaluation parameter set, in order to correctly calibrate the network. Training on a wide parameter parameter set mitigates the occurrences of high errors at the boundaries of the set. Additionally, we encountered high errors at ATM initial values, especially in cases where underlying prices were far apart (figure 5.10). A solution to this problem should be studied in depth in future research.

6.1.3. Further Research

In this work, it is shown that neural networks can serve as an effective numerical pricing technique for option pricing problems, both in a supervised and semi-supervised setting. In fact, the semi-supervised learning method, in which we approximate the solution to the PDE by minimising the error over the constraints, has produced relatively good approximations even in higher dimensions. These results align

with the findings in [20] and [22], in which prices in up to 200 dimensions were evaluated. However, further research in this area must be conducted to research alternative network architectures, specifically designed for the optimisation of multiple constraints. Additionally, equation 5.23 suggests a solution to balance the optimisation problem in order to prevent terminal loss to be minimised at the cost of interior loss. Further studies should quantify the effectiveness of this solution, and perhaps suggest different approaches. Finally, the parametric PDE method suffers from the major problem in which ATM predictions in edge cases contain large errors. This is shown in figure 5.10. In this figure, where we optimise a PDE model with an arithmetic payoff, ATM option approximations with large differences in underlying prices caused the largest errors. This problem should be addressed in further research.

The use of networks as a numerical pricing technique have proven to be very effective. Specifically, the use of neural networks to approximate parametric PDEs is an emerging field, which reaches far beyond finance. This thesis has contributed to this field by analysing different network architectures as well as addressing problems and possible solutions to this learning method.

6.2. Conclusions

In this thesis, we focused on applying neural networks to option pricing problems, including implied volatility approximations. We examined various network architectures in chapter 4, and found that the generalised highway network (section 4.3.3) outperformed the other networks in terms of MSE relative to its computational complexity in almost all of the supervised learning problems. Specifically, the Black-Scholes and Heston European call option pricing problems. In fact, the generalised highway network outperformed the networks used in [25] on the Black-Scholes problem, requiring a significantly smaller sized network, as well as fewer training cycles. The generalised highway network also performed well on a transformation of the implied volatility problem, in which we considered the log scaled time value of the option. On this problem, the only outperforming network was the no-recurrence network, a variation on the DGM network developed in section 4.3.6. The architecture of the no-recurrence network is very similar to the generalised network, containing only one additional non-linear operation. We concluded that the choice of network architecture can be decisive for the performance of the network, and is possibly more influential than the size of the network. We discovered that all networks suffered from poor performance on the implied volatility problem. Not a single network architecture managed to achieve sufficient accuracy to be considered for real purposes. When training on a transformed dataset, in which the option value was replaced by the log scaled time value of the option through a straightforward transformation, we found that issues with network approximations were resolved. On this transformed implied volatility problem we again found that the generalised highway network was the superior network architecture in MSE relative to training computational complexity. We concluded from this result that the dataset is also of considerable importance when training networks. In all of the problems including the implied volatility problem after the transformation, we found that the networks can provide an effective alternative to current industry standards for option pricing, especially when taking into account the real-time evaluation speed of the networks. The DGM network developed in [20], which focuses on solving PDE's, with its increased complexity did not perform better than the other networks on the supervised learning problems, in which we trained the networks on a precomputed dataset.

Alongside the one dimensional supervised learning problems, we also used multiple network architectures to approximate solutions to multidimensional partial differential equations (chapter 5). To this end, we used the network to optimise over the PDE as well as the terminal and boundary condition(s). For this problem, we found that contrary to the supervised learning problems, the DGM network is the superior network architecture. This is likely the result of the DGM's recurrent architecture, allowing for passage of the input vector throughout the entire network. This feature is especially useful for approximation of the terminal and boundary conditions, which are often non-smooth in option pricing problems, whereas the PDE on the interior domain is smooth. The MLP and highway architectures both suffered from approximating non-smooth functions due to their inability to optimise both the terminal constraints as well as the interior constraint.

On the PDE model approximations, we observed that longer training times did not yield performance improvements (section 5.3.5). This was caused by the definition of the loss function, which is an unweighted sum of the errors on the interior and boundary/terminal conditions. We found that weighing this sum using a regularisation factor may solve this problem. Additionally, we concluded in section

5.3.6 that the performance of the networks were unaffected by the increased dimensional problem, in which we approximated the Black-Scholes pricing problem with 20 underlying assets. We found that, where traditional methods suffer from the curse of dimensionality, the performance of the network did not decrease with respect to the 2-dimensional problem. We conclude that also for the approximation of multidimensional PDE's, neural networks may provide accurate results.

Finally, we considered parametric PDE's, in which we treated not only the space and time as variables, but also parameters present in the PDE. This increased the computational complexity of the problem significantly. However, the incorporation of parameters as variables allows us to evaluate the network on a large domain in real-time, without retraining the network. We concluded that even with additional dimensions caused by the parameters, the DGM network showed acceptable performance on the interior domain of the parameter set. However, remaining problems must be addressed in order to improve the feasibility of this technique, discussed in 6.1.

Bibliography

[1] A. Al-Aradi, A. Correia, D. Naiff, G. Jardim and Y. Saporito. Solving Nonlinear and High-Dimensional Partial Differential Equations via Deep Learning. *arXiv: Computational Finance*, 2018.

[2] A. Krizhevsky, I. Sutskever and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012.

[3] A. Sherstinsky. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 2020.

[4] Abadi, M., Agarwal, A.,. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Computer software], 2015. URL `https://www.tensorflow.org/`.

[5] B. Horvath, A. Muguruza and M. Tomas. Deep Learning Volatility. *Quantitative Finance*, 2021.

[6] C. Bayer, B.C. Hammouda, A. Papapantoleon, M. Samet, and R. Tempone. Optimal Damping with Hierarchical Adaptive Quadrature for Efficient Fourier Pricing of Multi-Asset Options in Lévy Models. *arXiv:2203.08196*, 2022.

[7] C. Bayer, M. Siebenmorgen and R. Tempone. Smoothing the payoff for efficient computation of Basket option prices. *Quantitative Finance*, 2018.

[8] C. Pötz. *Function approximation for option pricing and risk management Methods, theory and applications.* PhD thesis, Queen Mary University of London, 2020.

[9] R. Cassani. Multilayer perceptron example, 2017. URL `https://github.com/rcassani/mlp-example`.

[10] C.W. Oosterlee and L.A. Grzelak. *Mathematical Modeling and Computation in Finance*. World Scientific Publishing Europe Ltd, 2019.

[11] D. C. Ciresan, U. Meier, L. M. Gambardella and J. Schmidhuber. Deep, Big, Simple Neural Nets for Handwritten Digit Recognition. *Neural Computation*, 2010.

[12] D. Hendrycks, Dan and K. Gimpel. Gaussian Error Linear Units (GELUs). *arXiv:1606.08415*, 2016.

[13] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). `https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1`, 2022.

[14] E. Eberlein, K. Glau and A. Papapantoleon. Analysis of Fourier Transform Valuation Formulas and Applications. *Applied Mathematical Finance*, 2010.

[15] F. Black and M. Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 1973.

[16] F. Fang and C.W. Oosterlee. *The COS Method: An Efficient Fourier Method for Pricing Financial Derivatives*. PhD thesis, Delft University of Technology, 2010.

[17] F. Flesher. Stochastic Processes And The Feynman-Kac Theorem. [unpublished], 2014.

[18] J. Brogaard, T. Hendershott and R. Riordan. High Frequency Trading and Price Discovery, 2013.

[19] J. Papazoglou-Hennig. Internship Notes: Elementary Stochastic Calculus, Application to Financial Mathematics, Neural Network Methods for Financial Models, 2020.

[20] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

[21] J. Jordan. Gradient descent., 2017. URL `https://www.jeremyjordan.me/gradient-descent/`.

[22] K. Glau and L. Wunderlich. The Deep Parametric PDE Method: Application to Option Pricing. *Applied Mathematics and Computation*, 2020.

[23] K. He, X. Zhang, S. Ren and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[24] K. He, X. Zhang, S. Ren and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[25] L. Shuaiqiang, C. Oosterlee and S. Bohte. Pricing Options and Computing Implied Volatilities using Neural Networks. *Risks*, 7, 2019.

[26] M. A. H. Dempster, S. S. G. Hong. *Spread Option Valuation and the Fast Fourier Transform*, pages 203–220. Springer Berlin Heidelberg, 2002.

[27] M. D. McKay, R. J. Beckman and W. J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, (2), 1979.

[28] M. Telgarsky. Benefits of Depth in Neural Networks. In *29th Annual Conference on Learning Theory*. PMLR, 2016.

[29] P. Doust. Two Useful Techniques for Financial Modelling Problems. *Applied Mathematical Finance*, 2010.

[30] Q. Liu and D. A. Pierce. A Note on Gauss-Hermite Quadrature. *Biometrika*, 1994.

[31] R. K. Srivastava, K. Greff and J. Schmidhuber. Highway Networks. *arXiv:1505.00387*, 2015.

[32] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, 2015.

[33] S. Shreve. *Stochastic Calculus for Finance II*. Springer New York, NY, 2004.

[34] S. Wiesler and H. Ney. A Convergence Analysis of Log-Linear Training. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2011.

[35] G. Saporito. What is a perceptron?, 2019. URL `https://towardsdatascience.com/what-is-a-perceptron-210a50190c3b`.

[36] T. Guillaume. On the multidimensional Black–Scholes partial differential equation. *Risk in Financial Economics*, 2018.

[37] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*.

[38] W.S. Wadman. An advanced Monte Carlo method for the multi-asset Heston model. Master's thesis, Delft University of Technology, 2010.

[39] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. PMLR, 2010.

[40] Z. Dai, and Z. Yang, Y. Yang, J. Carbonell, Q. V. Le and R. Salakhutdinov. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019.