Imperial College London

MEng Individual Project

Department of Computing

Imperial College of Science, Technology and Medicine

# Towards Robust and Stable Deep Learning Algorithms for Forward Backward Stochastic Differential Equations

*Author:*
Aadhithya A Naarayan

*Supervisor:*
Dr Panos Parpas

*Second Marker:*
Dr Ruth Misener

June 17, 2024

Submitted in partial fulfillment of the requirements for the MEng degree in Computing Management and Finance of Imperial College London

**Abstract**

Before the 2008 financial crisis, the valuation of financial derivative contracts significantly underestimated risks involving counterparty defaults and funding costs, leading to substantial losses or bankruptcy. This exposed the need for a more comprehensive risk management framework governed by valuation adjustments known as xVAs. The valuation of financial contracts, such as options and forwards, and their corresponding xVAs can be represented as Partial Differential Equations (PDEs). Most of these PDEs do not have closed-form solutions, necessitating methods such as finite differences, spectral methods, and Monte Carlo simulations to solve them. These methodologies suffer from the "curse of dimensionality" as they do not scale well in high dimensions, making the algorithms computationally intensive. These PDEs can be converted into a set of Forward-Backward Stochastic Differential Equations, which can be approximated using deep learning. This project proposes an alternative methodology to solve the Black-Scholes Equation, used in the valuation of options and forward derivative contracts with both correlated and uncorrelated underlying asset prices, and uses these values to solve the xVA backward stochastic differential equations using a deep learning approach.

This project explores a residual neural network-based architecture called Nais-Net and Multi-Level Monte Carlo discretisation schemes to improve the model's performance. Through experiments involving these techniques, we were able to reduce the training time by approximately $85\%$ from the initial setup while maintaining a similar level of accuracy. The final algorithms proposed for the valuation of portfolios of call options and forward contracts, including the xVAs and simulation of collateral accounts, achieve an accuracy of approximately $0.1\%$ for the initial xVA values compared to another state-of-the-art approach, which achieves an accuracy of approximately $1\%$.

# Acknowledgments

I would like to express my sincere gratitude to my supervisor, Dr. Panos Parpas, for his support, guidance, and encouragement. His insights, feedback, and expertise have been instrumental in shaping this work. I am deeply appreciative of the time and effort he has invested in mentoring me through the duration of this project. I would also like to thank Dr. Ruth Misener for her valuable feedback on my interim report for this project.

My acknowledgments would not be complete without recognising my parents to whom I am eternally grateful for their support throughout my degree.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

Following the 2008 financial crisis, it has become accepted by practitioners and academics that the valuation of financial products must take into account the possibility of default of all parties involved in a transaction. This consideration has been incorporated into valuation equations via valuation adjustments. Valuation adjustments, commonly termed xVAs, were first explored by Duffie and Huang [1] to study the credit risk asymmetry for defaultable swap contracts. Bielecki et al. [2] and Brigo et al. [3] [4] [5] explore the concepts where the default risks of the bank and the counterparty are treated symmetrically using Credit Valuation Adjustments (CVA) and Debt Valuation Adjustments (DVA). A single funding stream operating under a unique, risk-free interest rate is no longer a realistic assumption, as trading is financed through different liquidity sources, especially for financial derivative contracts like options and forwards. Funding Valuation Adjustments (FVA) take this into account while pricing derivatives and portfolios, and are derived in Pallavicini et al. [6] and further discussed in Bielecki et al. [7]. Biagini et al. [8] and Bichuch et al. [9] [10] provide a framework to calculate these xVA adjustments by representing them as backward stochastic differential equations.

The Basel III accord [11] and The Fundamental Review of the Trading Book [12] necessitate that banks perform timely and accurate xVA calculations, which are required for stress testing and simulating multiple scenarios to understand potential risks and inform strategic planning. Joshi et al. [13] and Karlsson et al. [14] introduced a regression-based least squares Monte Carlo approach to solve these xVA equations, which is computationally expensive. Albanese et al. [15] provide a GPU-based implementation using Monte Carlo methods to solve this problem, thereby improving computational performance. However, these approaches suffer from the "curse of dimensionality", especially when considering calculations for large portfolios that banks manage daily. This can be overcome by developing a deep learning approach to approximate the solution of the xVA backward stochastic differential equation. A deep learning approach could also improve a bank's competitive advantage by allowing them to price their products accurately and reduce operational costs associated with these computationally intensive techniques.

Since their introduction by Jean-Michel Bismut [16] for the linear case and Pardoux and Peng [17] for the general case, backward stochastic differential equations have been widely used in mathematical finance. El Karoui et al. [18] explain the relevance of backward stochastic differential equations in the theory of derivative pricing and contingent claim valuation used for xVA calculations. The relationship between forward-backward stochastic differential equations and parabolic partial differential equation was introduced by Pardoux et al. [19]. In this project we represent the Black Scholes partial differential equation and xVA equation as a system of forward-backward stochastic differential equations which we can approximate using a neural network.

Traditional methods for solving partial differential equations tend to encounter the "curse of dimensionality", a problem experienced by practitioners with financial contract valuations and xVA calculations in high dimensions. Finite difference methods require every dimension to be discretised whereas a Monte Carlo approach requires sampling exponentially from a probability distribution. This issue of the "curse of dimensionality" stems from the reliance of these approaches on spatio-temporal grids, as the number of computational operations increases exponentially with the dimensions of the equation (see figure 1.1).

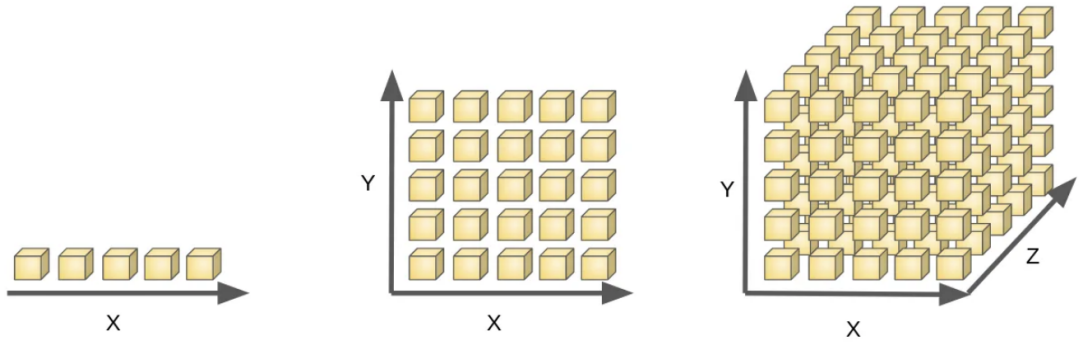


**Figure 1.1:** This figure illustrates an example where $5$ discretised points suffice to sample a unit interval. We would need to sample $5^d$ points for d dimensions indicating an exponential growth in the sampling operation (figure from [20]).

Many new stochastic approximation methods have been proposed and studied to circumvent this issue. These include backward stochastic differential equation-based approximation methods, in which nested conditional expectations are discretised through regression methods, as proposed by Gobet et al. [21] [22] and Bouchard et al. [23], and deep learning techniques, as proposed by Uchiyama et al. [24], M Raissi [25], Han et al. [26] [27], Sirignano et al. [28], and Beck et al. [29], all exploring different algorithms to solve partial differential equations, including the Black-Scholes equation, which is relevant to this report. These models learn to predict future paths originating from the same distribution as the training set, which consists of time-sequenced data generated from random Brownian motions. Through this project, we build upon the work of Batuhan Guler et al. [30][31], which has added to the work of Raissi [25][32] (refer to section 3.1.9 for a detailed discussion about the reasons for choosing to add to this approach over the others).

## 1.2 Contributions

Our main contribution is that we developed a deep learning-based solution to solve the Black-Scholes equation used to price options and forward contracts in high dimensions in a reasonable amount of time, and a deep learning-based algorithm to solve the xVA backward stochastic differential equation for portfolios of these contracts, taking into account the credit default, funding, and collateral-based risks of all parties involved in the transaction. We first replicated the work of Raissi [25] and Batuhan Guler et al. [30] and conducted further experiments using the Nais-Net architecture as proposed by Ciccone et al. [33]. Through these experiments, we identified that this architecture achieves the same relative error levels compared to the closed-form solution of the Black-Scholes-Barenblatt equation in 25k iterations, compared to 100k iterations of the feed-forward model proposed in [25]. We then proposed utilising a Non-Geometric Multi-Level Monte Carlo discretisation technique, which reduced the training time by approximately 85%. Furthermore, the models in [25] and [30] focused on the uncorrelated Black-Scholes-Barenblatt equation for a terminal payoff as the squared value of the underlying at the terminal time. Our approach works with more realistic payoffs such as portfolios of European-style forward contracts and call options on correlated and uncorrelated underlying assets. Finally, we proposed an algorithm to solve the xVA backward stochastic differential equation that uses the results from our earlier experiments. We compared our results to those of Gnoatto et al. [34], who achieved an error of approximately 1% for the initial xVA value for portfolios of both forward contracts and call options, compared to the values obtained using Monte Carlo simulations. Our algorithm achieved an error of approximately 0.1% for the same experiments.

## 1.3 Summary of the Report

The first part of this report (chapter 2) introduces the mathematical background required to derive the equations used in the neural network and the closed-form solution of the Black-Scholes equation for European-style options and forward contracts, which we use to evaluate the solution of our neural network in subsequent chapters. The second part (chapter 3) explains the structure of the neural network and how it solves the pricing problem. We identify and explore methodologies like the Nais-Net architecture in section 3.2 and Multi-Level Monte Carlo techniques in section 3.4 to improve accuracy and reduce computational complexity. We also explore the bias-variance trade-off for different network architectures and hyper-parameters in section 3.5 and extend our network to approximate the pricing for portfolios having correlated underlying assets in section 3.6. The third part (chapter 4) explains the mathematics behind the risk valuation equations, which are used to include the valuation adjustments to the prices obtained using the neural network in chapter 3. In this chapter, we also propose algorithms to solve the xVA backward stochastic differential equation in section 4.2.4. The final part of the report (chapter 5) focuses on the numerical experiments that compare the performance of the proposed model and algorithms to another state-of-the-art implementation by Gnoatto et al. [34] for portfolios of European-style call options and forward contracts.

# Chapter 2

# Background

This chapter focuses on the mathematical background required to set up the neural network and derive the equations necessary for the approximation of forward backward stochastic differential equations with a focus on the Black Scholes equation used for pricing financial derivative contracts. We also derive the closed form solution of this equation for the pricing of call options and forward contracts which we use to test the accuracy of our neural network in the subsequent chapters.

## 2.1 Markov Process

A random process $\{X(t), t \in T\}$ is called a first-order Markov process if for any $t_0 < t_1 < \cdots < t_n$ the conditional CDF of $X(t_n)$ for given values of $X(t_0), X(t_1), \ldots, X(t_{n-1})$ depends only on $X(t_{n-1})$ [35]. That is,

$$P(X(t_n) \leq x_n \mid X(t_{n-1}) \leq x_{n-1}, X(t_{n-2}) \leq x_{n-2}, \ldots, X(t_0) \leq x_0) \\ = P(X(t_n) \leq x_n \mid X(t_{n-1}) \leq x_{n-1})$$

## 2.2 Wiener Process

A Wiener process, also known as a Brownian motion, is a type of Markov process with the following properties:

1. It can be defined as a symmetric random walk $W(t)$ over $N$ periods of time $\Delta t$, with $\Delta t \to 0$, such that,

$$W(t_{k+1}) = W(t_k) + \varepsilon(t_k)\sqrt{\Delta t}$$

$$t_{k+1} = t_k + \Delta t$$

   for $k = 0, 1, 2, \ldots, N$, where $W(0) = 0$ and the disturbance follows a standard normal distribution: $\varepsilon(t_k) \sim \mathcal{N}(0, 1)$.

2. $W(t)$ is a continuous time limit of a discrete random walk so it is continuous in $t$ with independent and stationary increments.

3. For all $t_1 < t_2 \leq t_3 < t_4$ the random variables $W(t_2) - W(t_1)$ and $W(t_4) - W(t_3)$ are independent.

4. For all $j < k$ we have $W(t_k) - W(t_j) \sim \mathcal{N}(0, t_k - t_j)$.

5. It is not differentiable with respect to time which can be proved as follows:

   Consider the variance of the increment ratio as $h \to 0$,

$$\begin{aligned}\mathbb{E}\left[\left(\frac{W(t+h) - W(t)}{h}\right)^2\right] &= \frac{1}{h^2}\text{Var}[W(t+h) - W(t)] \\ &= \frac{1}{h} \xrightarrow{h \to 0} \infty\end{aligned}$$

## 2.3 Stochastic Differential Equations

The general form of a stochastic differential equation (SDE) is given by

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dW_t$$

where:

- $X_t$ is the stochastic process that is the solution to the SDE.
- $\mu(t, X_t)$ is a function representing the drift term, which describes the deterministic trend of the process.
- $\sigma(t, X_t)$ is a function representing the diffusion term, which models the random fluctuations. It's often associated with the volatility in financial models.
- $dW_t$ is the increment of a Wiener process (or Brownian motion), representing the random input to the system.
- $dt$ represents an infinitesimal increment in time.

Multi-dimensional SDEs involve vectors for $X_t$, $\mu$, and $\sigma$, and a matrix for the diffusion term to model the interactions between different components of the system (See figure 2.1).

### 2.3.1 Itô Process

An Itô process $X_t$ is defined as follows:

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dW_t \tag{2.1}$$

where $\mu(t, X_t)$ and $\sigma(t, X_t)$ are two continuous functions representing the drift and volatility terms, respectively, and $W_t$ is a Wiener process.

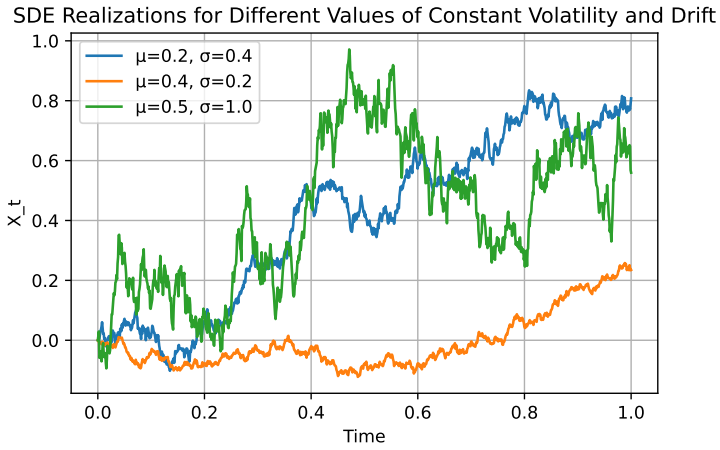


**Figure 2.1:** Stochastic process simulation

Now, consider a function $f(t, X_t)$ where $X_t$ is an Ito process. Using a Taylor expansion, we get

$$f(t+dt, X_t+dX_t) = f(t, X_t) + \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial X_t}dX_t + \frac{1}{2}\frac{\partial^2 f}{\partial X_t^2}(dX_t)^2 + \text{higher order terms}$$

Approximating by taking the first order terms, rearranging and substituting $X_t$ from equation 2.1 we get

$$df(t, X_t) \approx \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial X_t}(\mu dt + \sigma dW_t) + \frac{1}{2}\frac{\partial^2 f}{\partial X_t^2}(\mu dt + \sigma dW_t)^2$$

In the expansion of the term $(\mu dt + \sigma dW_t)^2$, $dt^2$ and $dt.dW$ are ignored due to second order degree whereas as the variance of a Wiener process is $dt$ we have $dW^2 = dt$. Applying this to the above equation and rearranging the terms we get Itô's Lemma:

$$df(t, X_t) = \left(\frac{\partial f}{\partial t} + \mu\frac{\partial f}{\partial X_t} + \frac{1}{2}\sigma^2\frac{\partial^2 f}{\partial X_t^2}\right)dt + \sigma\frac{\partial f}{\partial X_t}dW_t \tag{2.2}$$

### 2.3.2 Linking Partial Differential Equations and Stochastic Differential Equations

Consider a function $f(t, x)$ that is a solution to the following partial differential equation (PDE):

$$\frac{\partial f}{\partial t} + \mu(t, x)\frac{\partial f}{\partial x} + \frac{1}{2}\sigma^2(t, x)\frac{\partial^2 f}{\partial x^2} = 0$$

with terminal condition $f(T, x) = g(x)$, where $T$ is a fixed time and $g$ is a known function.

Let $X_t$ be a stochastic process described by the SDE:

$$dX_t = \mu(t, X_t)\,dt + \sigma(t, X_t)\,dW_t$$

where $\mu$ and $\sigma$ are the drift and volatility functions, respectively, and $W_t$ is a Wiener process.

Itô's Lemma tells us how to find the differential $df$ of a function $f$ that depends on a stochastic process. Applying Itô's Lemma as given in equation 2.2, we get:

$$df(t, X_t) = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial x}dX_t + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(dX_t)^2$$

Substituting $dX_t$ from the SDE into the above equation, we get:

$$df(t, X_t) = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial x}(\mu dt + \sigma dW_t) + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(\sigma^2 dt)$$

Simplifying and regrouping terms, and using the fact that $f$ solves the PDE, all terms involving $dt$ cancel out, leaving:

$$df(t, X_t) = \frac{\partial f}{\partial x}\sigma(t, X_t)dW_t$$

Integrating both sides from $t$ to $T$, we obtain:

$$f(T, X_T) - f(t, X_t) = \int_t^T \frac{\partial f}{\partial x}(s, X_s)\sigma(s, X_s)dW_s$$

Taking expectations and noting that the expectation of the stochastic integral of a non-anticipative function with respect to Brownian motion is zero, we find:

$$\mathbb{E}[f(T, X_T) \mid X_t] = f(t, X_t)$$

Finally, using the terminal condition $f(T, X) = g(X)$:

$$f(t, X_t) = \mathbb{E}[g(X_T) \mid X_t]$$

This result tells us that $f(t, X_t)$ can be interpreted as the conditional expectation of $g(X_T)$ given the current state $X_t$. This was introduced as the Feynman-Kac formula and the expectation is traditionally calculated using a Monte Carlo simulation as in Bouchard et al. [23].

### 2.3.3 Forward Stochastic Differential Equations

The general form of a Forward Stochastic Differential Equation is as follows:

$$X_t = X_0 + \int_0^t \mu(s, X_s)\, ds + \int_0^t \sigma(s, X_s)\, dW_s$$

where $\mu(s, X_s)$ and $\sigma(s, X_s)$ are two continuous functions representing the drift and volatility terms, respectively, and $X_s$ is a Wiener process.

### 2.3.4 Backward Stochastic Differential Equations

As explained in Gobet [36], consider the stochastic processes $X_t$, $Y_t$ and $Z_t$ defined as follows:

$$\begin{aligned} dX_t &= \mu(t, X_t)dt + \sigma(t, X_t)dW_t, \\ Y_t &= u(t, X_t) \quad \text{and}, \\ Z_t &= \nabla u(t, X_t)\sigma(t, X_t). \end{aligned}$$

with the terminal condition $u(T, x) = g(x)$ and function $u(t, x)$ such that:

$$\frac{\partial u}{\partial t} + \mu(t, x)\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2(t, x)\frac{\partial^2 u}{\partial x^2} + f(t, x, u(t, x), \nabla u(t, x)) = 0 \tag{2.3}$$

Applying Itô's lemma to $Y_t$, we get

$$dY_t = \left(\frac{\partial u}{\partial t} + \mu(t, X_t)\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2(t, X_t)\frac{\partial^2 u}{\partial x^2}\right) dt + \nabla u(t, X_t)\sigma(t, X_t)dW_t.$$

Utilising the PDE in equation 2.3 and the definition of $Z_t$, we get

$$dY_t = -f(t, X_t, Y_t, Z_t[\sigma(t, X_t)^{-1}])dt + Z_t dW_t.$$

By integrating between time periods t and T, we get

$$Y_T - Y_t = -\int_t^T f(s, X_s, u(s, X_s), Z_s[\sigma(s, X_s)^{-1}]\, ds + \int_t^T Z_s\, dW_s$$

using the terminal condition we derive the Backward SDE given by

$$Y_t = g(X_T) + \int_t^T f(s, X_s, u(s, X_s), Z_s[\sigma(s, X_s)^{-1}]\, ds - \int_t^T Z_s\, dW_s. \tag{2.4}$$

Together, the following set of equations are called Forward Backward Stochastic Differential Equations (FBSDEs).

$$\begin{aligned} X_t &= X_0 + \int_0^t \mu(s, X_s)\, ds + \int_0^t \sigma(s, X_s)\, dW_s \\ Y_t &= g(X_T) + \int_t^T f(s, X_s, u(s, X_s), Z_s[\sigma(s, X_s)^{-1}]\, ds - \int_t^T Z_s\, dW_s \end{aligned} \tag{2.5}$$

Backward SDEs are inherently complex to solve due to several key challenges. Unlike forward SDEs, defined by an initial condition, backward SDEs are characterised by a terminal condition, necessitating solutions that accommodate future information using backward induction methods. Their solutions often involve a coupled system of equations for the processes $Y_t$ and $Z_t$, intertwined in complex, nonlinear relationships, complicating analytical and numerical approaches to solve them. These computationally intensive methods require careful handling to ensure stability and accuracy. The use of neural networks to solve a set of FBSDEs as an alternative to traditional methods, as proposed by Raissi [25] and extended by Guler et al. [30], which involves approximating the processes $Y_t$ and $Z_t$ (by differentiating $Y_t$ with respect to $X_t$) leads to reduced computational complexity and produces promising results. So, we build on this work in chapter 3 focusing on the Black-Scholes equation for financial derivative valuation.

## 2.4 Black-Scholes Equation

The Black-Scholes Equation proposed by Black and Scholes [37] provides a model to price financial derivative contracts and liabilities which is widely used as the foundation for derivative pricing and risk management today. In this report, we will use the PDE representation of this equation with a defined terminal condition to predict prices and develop a risk framework using a neural network.

### 2.4.1 Partial Differential Equation Derivation

This section is based on the work of Hull [38].

Consider a stock with price $S$ that follows an Itô Process given by

$$dS = \mu S\, dt + \sigma S\, dW$$

Where $\mu$ represents the drift, which describes the deterministic trend of the process and $\sigma$ represents the diffusion which models the volatility of the process.
Suppose $f$ is a twice differentiable function of $S$ and $t$ and denotes the price of a derivative contract contingent on $S$. Using Itô's lemma as in equation 2.2 we get

$$df = \left(\mu S\frac{\partial f}{\partial S} + \frac{\partial f}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2}\right) dt + \sigma S \frac{\partial f}{\partial S} dW$$

The discretised versions of the above equations over a time interval $\Delta t$ are

$$\Delta S = \mu S \Delta t + \sigma S \Delta W \tag{2.6}$$

and

$$\Delta f = \left(\mu S\frac{\partial f}{\partial S} + \frac{\partial f}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2}\right) \Delta t + \sigma S \frac{\partial f}{\partial S} \Delta W \tag{2.7}$$

respectively.

From the above equation, we can see that the processes representing the underlying $S$ and $f$ are the same. So we can construct a portfolio of the stock and the derivative to eliminate the Wiener process to effectively price the derivative contract. This portfolio involves selling one derivative contract and buying $\frac{\partial f}{\partial S}$ of the underlying asset. We can formally define the the value of the portfolio $\Pi$ as

$$\Pi = -f + \frac{\partial f}{\partial S} S \tag{2.8}$$

Over a time interval $\Delta t$ we have

$$\Delta \Pi = -\Delta f + \frac{\partial f}{\partial S} \Delta S$$

From equations 2.6 and 2.7 we get

$$\Delta\Pi = -\left(\left(\mu S\frac{\partial f}{\partial S} + \frac{\partial f}{\partial t} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 f}{\partial S^2}\right)\Delta t + \sigma S\frac{\partial f}{\partial S}\Delta W\right) + \frac{\partial f}{\partial S}(\mu S\Delta t + \sigma S\Delta W)$$

On rearranging and cancelling out like terms we get

$$\Delta\Pi = -\frac{\partial f}{\partial t}\Delta t - \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\sigma^2 S^2\Delta t = \left(-\frac{\partial f}{\partial t} - \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\sigma^2 S^2\right)\Delta t \tag{2.9}$$

This equation does not involve $\Delta W$, so the portfolio is risk-less during time $\Delta t$. Assuming there are no transaction costs, all securities are perfectly divisible, there are no dividends during the life of the derivative, there are no risk-less arbitrage opportunities, security trading is continuous and the risk-free rate of interest, $r$, is constant and the same for all maturities, this portfolio will earn the rate of return $r$ over the period $\Delta t$. That is,

$$\Delta\Pi = r\Pi\Delta t$$

Substituting equations 2.8 and 2.9 in the above equation we get,

$$\left(-\frac{\partial f}{\partial t} - \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\sigma^2 S^2\right)\Delta t = r\left(-f + \frac{\partial f}{\partial S}S\right)\Delta t$$

On rearranging and cancelling $\Delta t$ on both sides we get the Black-Scholes Partial Differential Equation given by

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 f}{\partial S^2} = rf \tag{2.10}$$

We can now use different boundary conditions depending on the payoff of the derivative contract to arrive at a specific solution to this equation.

### 2.4.2 Closed Form Solution

In this section we transform the Black-Scholes PDE to an initial value problem in the form of the heat equation. Suppose we have a fixed expiry time $T$, the value of $f(T, S)$ is known and the exercise price of the derivative contract is $K$. Consider two new independent variables $\tau$ and $x$ such that

$$\begin{aligned} t &= T - \frac{2\tau}{\sigma^2}, \\ S &= Ke^x. \end{aligned} \tag{2.11}$$

We also replace the variable $f$ with $V$ given by

$$f(t, S) = KV(\tau, x) = KV(\frac{1}{2}\sigma^2(T - t), log(\frac{S}{K})). \tag{2.12}$$

We now have an initial value problem where $V$ is known as time $\tau = 0$ and we are interested in the behaviour of $V$ for $\tau > 0$.

Since $x = log(\frac{S}{K})$ and $\tau = \frac{1}{2}\sigma^2(T-t)$, we have

$$\begin{aligned} \frac{\partial x}{\partial S} &= \frac{1}{(S/K)} \times \frac{1}{K} = \frac{1}{S}, & \frac{\partial x}{\partial t} &= 0, \\ \frac{\partial \tau}{\partial t} &= -\frac{1}{2}\sigma^2, & \frac{\partial \tau}{\partial S} &= 0. \end{aligned} \tag{2.13}$$

Using the equations in 2.13 we get

$$\begin{aligned} \frac{\partial f}{\partial t} &= K\left(\frac{\partial V}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial V}{\partial \tau}\frac{\partial \tau}{\partial t}\right) = -\frac{1}{2}\sigma^2 K\frac{\partial V}{\partial \tau}, \\ \frac{\partial f}{\partial S} &= K\left(\frac{\partial V}{\partial x}\frac{\partial x}{\partial S} + \frac{\partial V}{\partial \tau}\frac{\partial \tau}{\partial S}\right) = \frac{K}{S}\frac{\partial V}{\partial x} \end{aligned} \tag{2.14}$$

and

$$\begin{aligned} \frac{\partial^2 f}{\partial S^2} &= K\frac{\partial}{\partial S}\left(\frac{1}{S}\frac{\partial V}{\partial x}\right) \\ &= K\left(-\frac{1}{S^2}\frac{\partial V}{\partial x} + \frac{1}{S}\frac{\partial^2 V}{\partial x^2}\frac{\partial x}{\partial S}\right) \\ &= K\left(-\frac{1}{S^2}\frac{\partial V}{\partial x} + \frac{1}{S^2}\frac{\partial^2 V}{\partial x^2}\right) \end{aligned} \tag{2.15}$$

Substituting equations 2.14 and 2.15 in the Black-Scholes Partial Differential Equation in equation 2.10 we get

$$-\frac{1}{2}\sigma^2 K\frac{\partial V}{\partial \tau} + \frac{1}{2}\sigma^2 S^2 K\left(-\frac{1}{S^2}\frac{\partial^2 V}{\partial x^2}\right) + rSK\frac{\partial V}{\partial x} - rKV(\tau, x) = 0.$$

We can simplify this equation by taking

$$k = 2\frac{r}{\sigma^2}$$

to get

$$\frac{\partial V}{\partial \tau} = \frac{\partial^2 V}{\partial x^2} + (k-1)\frac{\partial V}{\partial x} - kV(\tau, x). \tag{2.16}$$

To further simplify let us take

$$V(\tau, x) = e^{\alpha x + \beta \tau}u(\tau, x) \tag{2.17}$$

where $\alpha$ and $\beta$ are constants.

Differentiating equation 2.17 we get

$$\begin{aligned} \frac{\partial V}{\partial \tau} &= e^{\alpha x+\beta\tau}\left(\beta u + \frac{\partial u}{\partial \tau}\right), \\ \frac{\partial V}{\partial x} &= e^{\alpha x+\beta\tau}\left(\alpha u + \frac{\partial u}{\partial x}\right), \\ \frac{\partial^2 V}{\partial x^2} &= e^{\alpha x+\beta\tau}\left(\alpha^2 u + 2\alpha\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2}\right). \end{aligned} \tag{2.18}$$

Substituting equation 2.18 in equation 2.16 and cancelling $e^{\alpha x+\beta\tau}$ we get

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2} + (k-1+2\alpha)\frac{\partial u}{\partial x} + (\alpha^2 + \alpha k - \alpha - k - \beta)u. \tag{2.19}$$

We choose values for $\alpha$ and $\beta$ as follows

$$\begin{aligned} \alpha &= \frac{1-k}{2}, \\ \beta &= -\frac{1}{4}(k+1)^2. \end{aligned} \tag{2.20}$$

to reduce the Black-Scholes PDE to the heat equation given by

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2} \tag{2.21}$$

Assuming at time $\tau = 0$, $u(0, x)$ is known to be

$$u(0, x) = u_0(x)$$

The heat equation admits the following solution:

$$u(\tau, x) = \frac{1}{\sqrt{4\pi k\tau}} \int_{-\infty}^{\infty} u_0(s) \exp\left(-\frac{(x-s)^2}{4k\tau}\right) ds \tag{2.22}$$

### 2.4.3 European Call Option Pricing

A European call option gives the holder the right but not the obligation to buy the underlying asset for fixed price at maturity. The boundary conditions for a European call option on the Black-Scholes PDE in equation 2.10 with strike price $K$ and a maturity $T$ with the risk-free interest rate assumed to be fixed and same for all maturities at $r$ are as follows:

$$\begin{cases} f(t,0) = 0 & (f = 0 \text{ whenever } S_t = 0 \text{ for any } t) \\ \lim_{S\to\infty} f(t,S) = S - Ke^{-r(T-t)} & (\text{discounted exercise price}) \\ f(T,S) = max(S-K, 0) & (\text{payoff at expiration}) \end{cases}$$

From equations 2.17 and 2.20 we have

$$V(\tau, x) = e^{\frac{1}{2}(1-k)x - \frac{1}{4}(k+1)^2\tau} u(\tau, x)$$

As $f(t, x) = KV(t, x)$, the boundary conditions now become:

$$\begin{cases} \lim_{x\to-\infty} V(0,x) = 0 & (\text{V = 0 whenever } S_t = 0 \text{ for any } t) \\ \lim_{x\to\infty} V(t,x) = e^x - e^{-r(\frac{2\tau}{\sigma^2})} = e^x - e^{-k\tau} & (\text{discounted exercise price}) \\ V(0,x) = max(e^x - 1, 0) & (\text{payoff at expiration}) \end{cases}$$

So, in terms of $u$ we get

$$u_0(x) = e^{-\frac{1}{2}(1-k)x}\, V(0,x) \quad = max(e^{\frac{k+1}{2}x} - e^{\frac{k-1}{2}x}, 0) \tag{2.23}$$

Substituting equation 2.23 in the solution to the heat equation given in equation 2.22 we get

$$u(\tau, x) = \frac{1}{\sqrt{4\pi\tau}} \int_{-\infty}^{\infty} max(e^{\frac{1}{2}(k+1)s} - e^{\frac{1}{2}(k-1)s},\ 0) \exp\left(-\frac{(x-s)^2}{4\tau}\right) ds$$

Substituting $y = \frac{s-x}{\sqrt{2\tau}}$ ,

$$\begin{aligned} u(\tau, x) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \left(e^{\frac{1}{2}(k+1)(\sqrt{2\tau}y+x)} - e^{\frac{1}{2}(k-1)(\sqrt{2\tau}y+x)}\right) e^{-\frac{1}{2}y^2}\, dy \\ &= \frac{1}{\sqrt{2\pi}} \int_{-x/\sqrt{2\tau}}^{\infty} \left(e^{\frac{1}{2}(k+1)(\sqrt{2\tau}y+x)} - e^{\frac{1}{2}(k-1)(\sqrt{2\tau}y+x)}\right) e^{-\frac{1}{2}y^2}\, dy \\ &= \frac{1}{\sqrt{2\pi}} \int_{-x/\sqrt{2\tau}}^{\infty} e^{\frac{1}{2}(k+1)(\sqrt{2\tau}y+x)} e^{-\frac{1}{2}y^2}\, dy \\ &\quad - \frac{1}{\sqrt{2\pi}} \int_{-x/\sqrt{2\tau}}^{\infty} e^{\frac{1}{2}(k-1)(\sqrt{2\tau}y+x)} e^{-\frac{1}{2}y^2}\, dy. \end{aligned}$$

We can simplify this integral as follows:

$$\begin{aligned} u(\tau, x) = e^{\frac{1}{4}(k+1)^2\tau + \frac{1}{2}(k+1)x} \cdot N(\frac{x}{\sqrt{2\tau}} + \frac{1}{2}(k+1)\sqrt{2\tau}) \\ - e^{\frac{1}{4}(k-1)^2\tau + \frac{1}{2}(k-1)x} \cdot N(\frac{x}{\sqrt{2\tau}} + \frac{1}{2}(k-1)\sqrt{2\tau}) \end{aligned}$$

where $N$ denotes the normalised Gaussian integral:

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{z^2}{2}} dz$$

Let $\mathrm{d1} = \frac{x}{\sqrt{2\tau}} + \frac{1}{2}(k+1)\sqrt{2\tau}$ and $\mathrm{d2} = \frac{x}{\sqrt{2\tau}} + \frac{1}{2}(k-1)\sqrt{2\tau}$. We have

$$\begin{aligned} V(\tau, x) &= e^{\frac{1}{2}(1-k)x - \frac{1}{4}(k+1)^2\tau} \cdot (e^{\frac{1}{4}(k+1)^2\tau + \frac{1}{2}(k+1)x} \cdot N(\mathrm{d1}) - e^{\frac{1}{4}(k-1)^2\tau + \frac{1}{2}(k-1)x} \cdot N(\mathrm{d2})) \\ &= e^x \cdot N(\mathrm{d1}) - e^{-k\tau} \cdot N(\mathrm{d2}) \end{aligned}$$

Substituting based on equations 2.11 and 2.12 we get the price of a European call option given by:

$$f(t, x) = S \cdot N(\mathrm{d1}) - Ke^{-r(T-t)} \cdot N(\mathrm{d2}) \tag{2.24}$$

with

$$\begin{aligned} d_1 &= \frac{ln(S/K) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}} \\ d_2 &= \frac{ln(S/K) + (r - \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}} \end{aligned}$$

### 2.4.4 European Put Option Pricing

A European put option gives the holder the right but not the obligation to sell the underlying asset for fixed price at maturity. The price of this derivative contract can be derived similarly to the European call option price given in the previous section. The price of a European put option is:

$$f(t,x) = Ke^{-r(T-t)} \cdot N(-\mathrm{d2}) - S \cdot N(-\mathrm{d1}) \tag{2.25}$$

with

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

### 2.4.5 Forward Contract Pricing

A forward contract gives the holder the obligation to buy the underlying asset at a specified price at maturity. The price of a forward contract can be derived from the theory of put-call parity (see figure 2.2). The put-call parity can be expressed as follows for a call and put option with strike price K, maturity T and initial price of the underlying S:

$$Forward(K,T) = Call(K,T) - Put(K,T)$$

So, the value of a forward contract with forward price K, maturity T and initial price of the underlying S can be replicated by creating a portfolio that is long a call option and short a put option with strike K, maturity T and S is the initial price of the underlying. Using this theorem and the equations for the price of a call and put option given by equations 2.24 and 2.25 respectively we have

$$\begin{aligned} Forward(K,T) &= S \cdot N(\mathrm{d1}) - Ke^{-r(T-t)} \cdot N(\mathrm{d2}) - Ke^{-r(T-t)} \cdot N(-\mathrm{d2}) + S \cdot N(-\mathrm{d1}) \\ &= \frac{S}{\sqrt{2\pi}} \cdot \left(\int_{-\infty}^{\mathrm{d1}} e^{-\frac{z^2}{2}} dz + \int_{-\infty}^{-\mathrm{d1}} e^{-\frac{z^2}{2}} dz\right) - \\ &\qquad \frac{Ke^{-r(T-t)}}{\sqrt{2\pi}} \cdot \left(\int_{-\infty}^{\mathrm{d2}} e^{-\frac{z^2}{2}} dz + \int_{-\infty}^{-\mathrm{d2}} e^{-\frac{z^2}{2}} dz\right) \end{aligned}$$

As the Gaussian integral is symmetric about 0, we have

$$\int_{-\infty}^{-x} e^{-\frac{z^2}{2}} dz = \int_{\infty}^{x} e^{-\frac{z^2}{2}} dz$$

Therefore,

$$\begin{aligned} Forward(K,T) &= \frac{S}{\sqrt{2\pi}} \cdot (\int_{-\infty}^{\mathrm{d1}} e^{-\frac{z^2}{2}} dz + \int_{\infty}^{\mathrm{d1}} e^{-\frac{z^2}{2}} dz) - \\ &\qquad \frac{Ke^{-r(T-t)}}{\sqrt{2\pi}} \cdot (\int_{-\infty}^{\mathrm{d2}} e^{-\frac{z^2}{2}} dz + \int_{\infty}^{\mathrm{d2}} e^{-\frac{z^2}{2}} dz) \\ &= \frac{S}{\sqrt{2\pi}} \cdot (\int_{-\infty}^{\infty} e^{-\frac{z^2}{2}} dz) - \frac{Ke^{-r(T-t)}}{\sqrt{2\pi}} \cdot (\int_{-\infty}^{\infty} e^{-\frac{z^2}{2}} dz) \\ &= S - Ke^{-r(T-t)} \end{aligned}$$

Therefore, the value of a forward contract with forward price K, maturity T and initial price of the underlying S is given by

$$Forward(K,T) = S - Ke^{-r(T-t)} \tag{2.26}$$

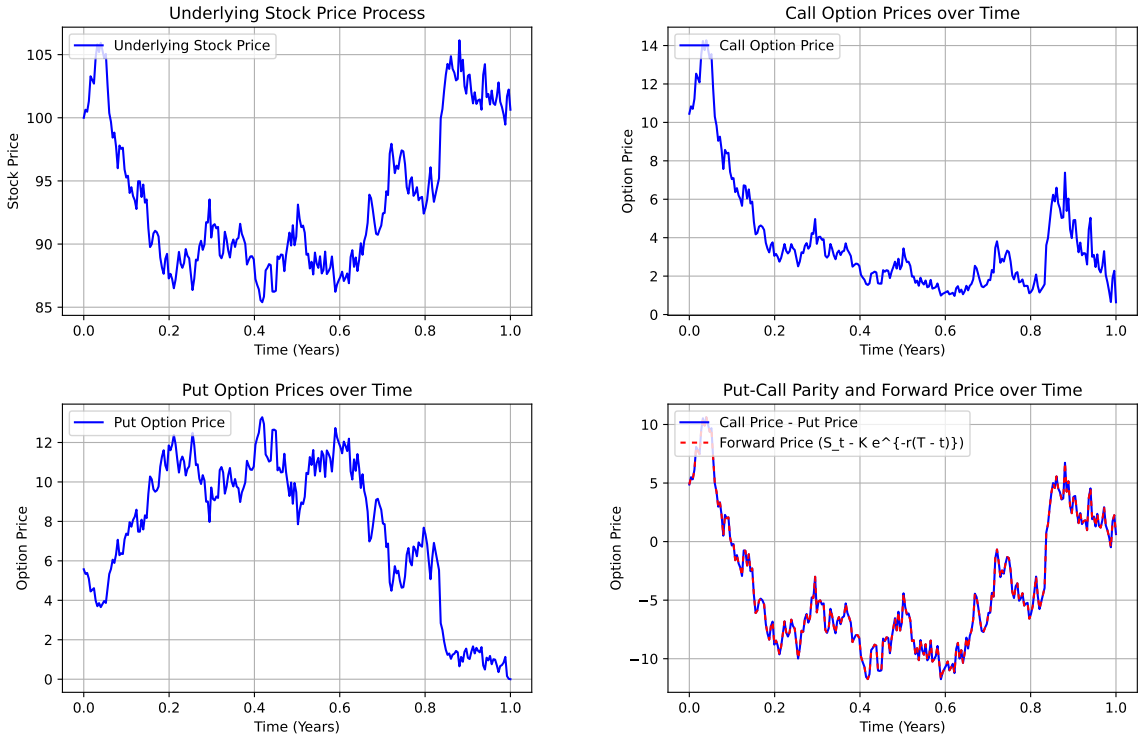


**Figure 2.2:** These figures represent the call option, put option and forward prices for an underlying process of 1-Dimension with initial value $S_0 = 100$, $K = 100$, $r = 5\%$, $\sigma = 0.2$ and $T = 1$. The figure on the bottom right verifies the put-call parity.

In the following sections, we focus on pricing portfolios consisting of baskets of correlated and uncorrelated European call options, European put options, and forward contracts, which can be represented as high-dimensional PDEs, with each dimension representing one asset, and developing a risk management framework for these portfolios using neural networks. We use the closed form solution and perform Monte Carlo simulations using the equations derived in this section to obtain the exact values to compare with the neural network's output.

# Chapter 3

# Neural Network

In this section we build on the work of Raissi [25] and Batuhan Guler, Alexis Laignelet and Panos Parpas [30]. We define the forward-backward stochastic differential equations and their relation to the Black-Scholes Barenblatt equation in high dimensions and formulate this problem for a neural network to solve. We also define the neural network architecture, training algorithm and the methodologies used to improve the initial results.

## 3.1 Initial Setup

In this section we aim to reproduce the results of the work of Raissi [25] and Batuhan Guler, Alexis Laignelet and Panos Parpas [30] based on their fully connected neural network setup.

### 3.1.1 Forward-Backward Stochastic Differential Equations

Let us consider the following set of forward-backward stochastic differential equations of the form

$$\begin{aligned} dX_t &= \mu(t, X_t, Y_t, Z_t)dt + \sigma(t, X_t, Y_t)dW_t, \qquad t \in [0, T], \\ X_0 &= \xi, \\ dY_t &= \phi(t, X_t, Y_t, Z_t)dt + Z_t'\sigma(t, X_t, Y_t)dW_t, \quad t \in [0, T]), \\ Y_T &= g(X_T), \end{aligned} \tag{3.1}$$

where $X_t$, $Y_t$ and $Z_t$ are different stochastic processes, $W_t$ is a wiener process, $\mu$ and $\phi$ represent drift functions and $\sigma$ is the diffusion function for the processes. From Pardoux et al. [19] we know that a solution to this set of equations can be linked to the solution of a partial differential equation of the form

$$\begin{aligned} u_t &= f(t, x, u, Du, D^2u) \\ &= \phi(t, x, u, Du) - \mu(t, x, u, Du)'Du - \frac{1}{2}\operatorname{Tr}[\sigma(t, x, u)\sigma(t, x, u)'D^2u] \end{aligned} \tag{3.2}$$

where $Du$ represents the gradient, $D^2u$ represents the hessian of u respectively with $Y_t = u(t, X_t)$ and $Z_t = Du(t, X_t)$ and the terminal condition of u is known that is,

$u(T, X_T) = g(X_T)$. We can approximate the function $u(t, X_t)$ using a neural network using the PyTorch library as introduced by Paszke et al. [39]. The terms $Du$ and $D^2u$ are calculated using the automatic differentiation capability of this library. This technique is superior to numerical and symbolic differentiation especially in machine learning as explained in Baydin et al. [40].

### 3.1.2 Euler-Maruyama Scheme

We now discretise the equations in 3.1 using the Euler-Maruyama scheme to get the following set of equations:

$$\begin{aligned} \Delta W_n &\sim \mathcal{N}(0, \Delta t_n), \\ X_{n+1} &\approx X_n + \mu(t_n, X_n, Y_n, Z_n)\Delta t_n + \sigma(t_n, X_n, Y_n)\Delta W_n, \\ Y_{n+1} &\approx Y_n + \phi(t_n, X_n, Y_n, Z_n)\Delta t_n + (Z_n)'\sigma(t_n, X_n, Y_n)\Delta W_n, \\ \Delta t_n &= t_{n+1} - t_n = \frac{T}{N}, \end{aligned} \tag{3.3}$$

for $n = 0, 1, 2, ..., N-1$, where $N$ is the number of time-steps.

### 3.1.3 Loss Function

The neural network predicts $Y_n$ for each input $(t_n, X_n)$. The loss function compares this prediction of the neural network with the actual value calculated by the discretisation step in equation 3.3 using a sum squared error loss for $N$ time-steps and $M$ realisations of the wiener process $W$ also known as batch size is given by:

$$\begin{aligned} \sum_{m=1}^{M}\sum_{n=0}^{N-1} \left|Y_{n+1}^m - Y_n^m - \Phi_n^m \Delta t_n - (Z_n^m)'\Sigma_n^m \Delta W_n^m\right|^2 \\ + \sum_{m=1}^{M} \left|Y_N^m - g(X_N^m)\right|^2 + \sum_{m=1}^{M} \left|Z_N^m - g'(X_N^m)\right|^2 \end{aligned} \tag{3.4}$$

where $\Phi_n^m = \phi(t_n, X_n^m, Y_n^m, Z_n^m)$ and $\Sigma_n^m = \sigma(t_n, X_n^m, Y_n^m)$.

From the equations in 3.3 we also have

$$\begin{aligned} X_{n+1}^m &= X_n^m + \mu(t_n^m, X_n^m, Y_n^m, Z_n^m)\Delta t_n + \sigma(t_n^m, X_n^m, Y_n^m)\Delta W_n^m, \\ Y_n^m &= u(t_n, X_n^m), \\ Z_n^m &= Du(t_n, X_n^m), \end{aligned} \tag{3.5}$$

with $X_0^m = \xi$ for all $m$.

### 3.1.4 Architecture

As mentioned in the previous section, the neural network predicts $Y_n$ for each batch of inputs $(t_n, X_n)$. At each time-step $n$, $X_n$ is calculated using the equation in 3.5

for all $M$ realisations of $X_n$. $Y_n$ is calculated using the neural network and $Z_n^m$ is obtained using the automatic differentiation functionality of the PyTorch library by taking the derivative of $Y_n$ with respect to $X_n$. The figure 3.1 represents the neural network which predicts $Y_n$ at each time-step $n$.

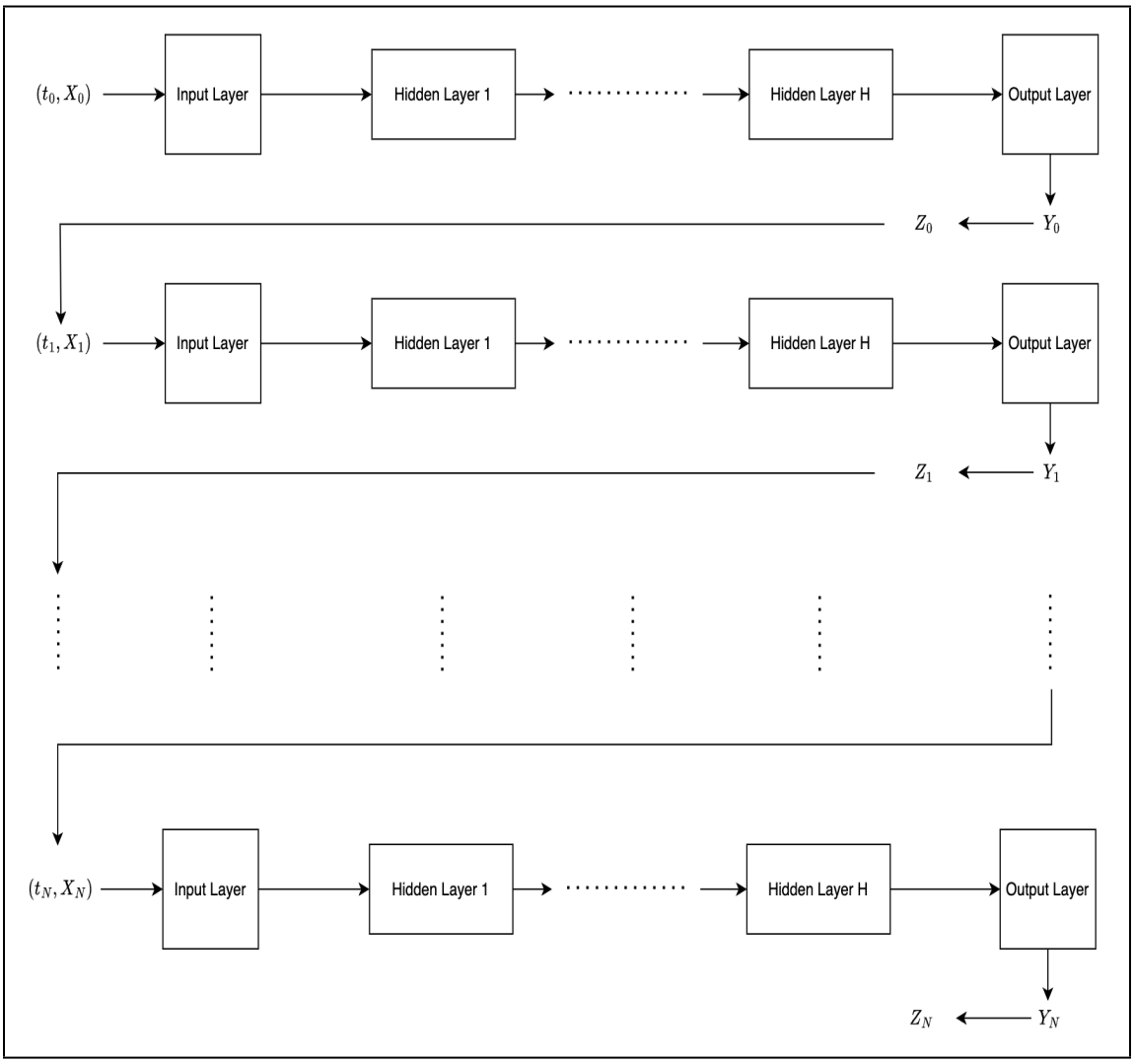


**Figure 3.1:** Collection of Forward-Backward Stochastic Differential Equation Neural Networks to approximate each $Y_n$ at each time step $n = 1...N$ for $m$ realisations of the underlying. We use the same neural network at each time step having $H$ hidden layers with the parameters learned using the algorithm defined in section 3.1.5.

.

### 3.1.5 Training Algorithm

The neural network is initialised with its attributes that include $\xi$ which is the initial value of the stochastic process $X_t$, $T$ which is the terminal time, $M$ which is the number of realisations of the Wiener process, $N$ which is the number of time-steps, $D$ which represents the number of dimensions of the PDE and $layers$ which represents the architecture of the neural network block. The neural network learns to produce the value $Y_t = u(t, X_t)$ with inputs $t$ and $X_t$ by minimising the loss function as in

equation 3.4. Furthermore, the value of $Z_t$ is calculated by using the automatic differentiation capabilities of the PyTorch library. The training algorithm is laid out in algorithm 1 below:

**Algorithm 1:** Training Algorithm for the Neural Network

initialise the model with its attributes - $\xi$, $T$, $M$, $N$, $D$ and $layers$;
**for** *number of iterations* **do**
  Generate $M$ realisations of the Wiener Process $W$ and time-steps $\Delta t$;
  initialise the loss to 0;
  Obtain the values for $M$ realisations of $Y_0$ and $Z_0$ by passing ($t_0$,$\xi$) through the model and automatic differentiation;
  **for** *number of time steps* $n = 1, ...., N-1$ **do**
    With $\Delta t_n$, $X_n$, $W_n$ $Y_n$ and $Z_n$ obtain $X_{n+1}$ and the true value $\widehat{Y}_{n+1}$ using equations 3.3;
    Obtain the model's predicted value $Y_{n+1}$ and $Z_{n+1}$ by passing $(t_n, X_n)$ as input and automatic differentiation;
    Add $\sum_{m=1}^{M} \left| Y_{n+1}^m - \widehat{Y}_{n+1}^m \right|^2$ to the loss;
  Add $\sum_{m=1}^{M} |Y_N^m - g(X_N^m)|^2 + \sum_{m=1}^{M} |Z_N^m - g'(X_N^m)|^2$ to the loss for the terminal time $T$;
  Perform back-propagation to compute the gradients across the neural network;
  Minimise the loss by using an appropriate optimiser;
  Update the weights;

### 3.1.6 Black-Scholes Barenblatt Equation

Consider the following set of forward backward stochastic differential equations based on the set of equations in 3.1

$$\begin{aligned} dX_t &= \sigma \text{diag}(X_t)\, dW_t, && t \in [0,T], \\ X_0 &= \xi, \\ dY_t &= r(Y_t - Z_t' X_t)\, dt + \sigma Z_t' \text{diag}(X_t)\, dW_t, && t \in [0,T], \\ Y_T &= g(X_T), \end{aligned} \tag{3.6}$$

Here, we have the stochastic process defined by $X_t$ representing the underlying asset and $Y_t$ represents the value of the contract. We assume the underlying has no drift i.e $\mu(t, X_t, Y_t, Z_t) = 0$ , the volatility of the underlying is assumed to be constant a constant $\sigma$ with $\sigma(t, X_t, Y_t) = \sigma \cdot \text{diag}(X_t)$ and the drift function for the contract value $\phi(t, X_t, Y_t, Z_t) = r(Y_t - Z_t' X_t)$. Furthermore, we assume that the price of the underlying assets are uncorrelated. Using the general solution for the set of equations in 3.1 given by the equation 3.2 we get the Black-Scholes Barenblatt partial differential equation:

$$u_t = -\frac{1}{2} \text{Tr}[\sigma^2 \,\text{diag}(X^2) D^2 u] + r(u - (Du)' x)$$

A solution to this equation is given by:

$$u(t,x) = e^{(r+\sigma^2)(T-t)}g(x) \tag{3.7}$$

### 3.1.7 Fully Connected Network Architecture

Based on the architecture described in Raissi [25], the neural network contains $H = 4$ hidden layers with each hidden layer having $L = 256$ neurons. The input layer has $D + 1$ neurons where $D$ is the number of dimensions of the $X_t$. The output layer contains the prediction $Y_t$. The neurons in each of the layers use sinusoidal activation. See figure 3.2 for the network architecture.

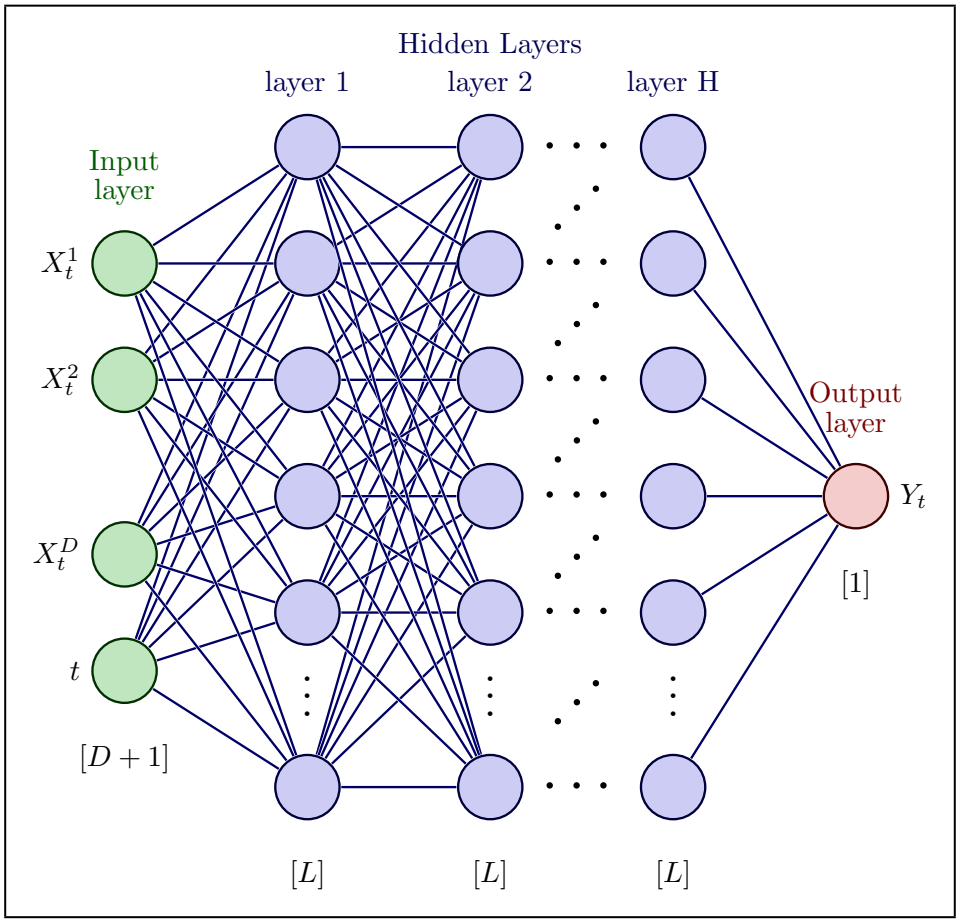


**Figure 3.2:** Fully Connected Neural Network Architecture

### 3.1.8 Experimental Results

Here, we conduct an experiment to train the model using the following parameters, the representations of these variables are explained in section 3.1.5, $\xi = [1.0, 0.5, 1.0, 0.5....]$, $D = 100$, $N = 50$, $M = 100$ and the architecture is described in section 3.1.7. Each of the neurons in the model have sinusoidal activation. We also assume that the risk-free rate $r = 0.05$ and the volatility of the underlying is

constant $\sigma = 0.4$. The payoff of the contract is given by $g(x) = ||x||^2$ which is also the terminal condition of the set of forward backward stochastic differential equations in equation 3.6. We use the Adam optimiser proposed by Kingma et al. [41] to minimise the loss function in equation 3.4. The model is trained for 100k iterations with a learning rate at $1e^{-3}$ for the first 20k iterations, $1e^{-4}$ for the next 30k iterations, $1e^{-5}$ for 30k iterations and $1e^{-5}$ for the last 20k iterations. We compare our results in figure 3.3 with that of Raissi [32] in figure 3.4 below:

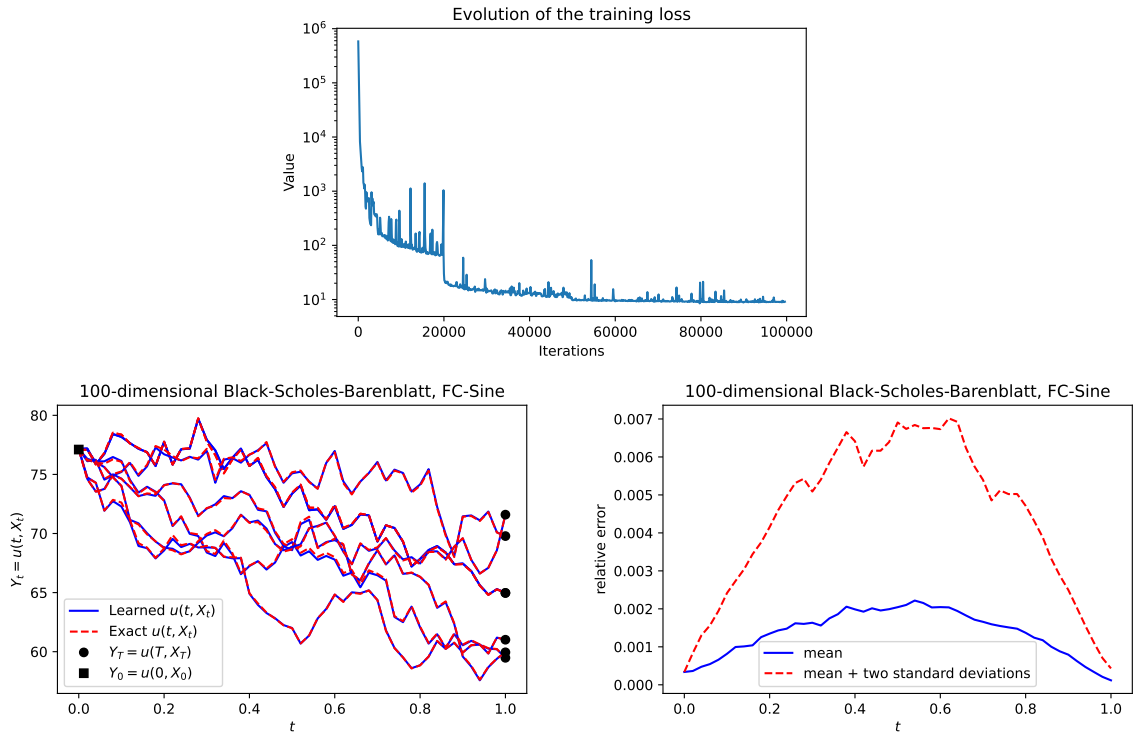


**Figure 3.3:** Training loss, predictions for different realisations of $X_t$ and relative error across predictions of 100 realisations for $X_t$ after training our model with parameters mentioned in section 3.1.8.

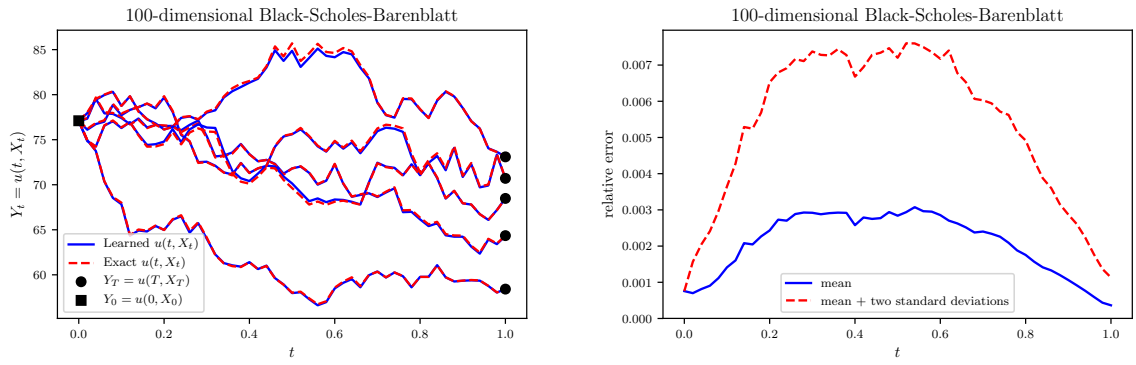


**Figure 3.4:** Predictions and relative error from Raissi [25] with the same parameters.

We see a difference in the predicted paths of the process $Y_t$ from figures 3.3 and 3.4 which is due to the randomness of the stochastic processes. The relative error metrics from these figures indicate that we have successfully been able to replicate

the performance as in Raissi [25]. The time taken to train our model is 13497s using the NVIDIA GeForce RTX 4080 GPU.

### 3.1.9 Analysis of Results and Comparison with Related Work

We choose the model as proposed by Raissi [25] over other state of the art deep learning methodologies to solve FBSDEs as proposed in Han et al. [26] [27], Sirignano et al. [28], and Beck et al. [29] due to certain drawbacks of these models which are overcome by this approach.

Firstly, the models in [26], [27], [28] and, [29] do not use a system of neural networks to approximate the entire function $u(t, x)$. Instead with the set of equations as described in equation 3.6, the algorithm uses a system of neural networks to only approximate $Z_n = Du(t_n, X_n)$ and the Euler-Maruyama scheme as in equation 3.3 is used to calculate the values of the stochastic processes $X_t$ and $Y_t$ at different time-steps. The loss function is given by $\sum_{m=1}^{M} |Y_n^m - g(X_n^m)|^2$ and does not contain a term for the control variate $Z_n$. The model only predicts the initial value $Y_0$ given by $u(t_0, X_0)$ by taking $X_0$ and $Z_0$ as parameters in the model. This suggests that the model may need to be retrained to calculate the value of $Y_t$ at subsequent times. Secondly, the complexity and the number of parameters of the neural network grow with an increase in the number of discretised time-steps as new networks with different parameters are initialised at each time-step. This leads to longer training times and memory usage for FBSDEs requiring higher accuracy through employing fine discretisation levels.

The model as proposed by Raissi [25] and explained in the sections above overcomes these issues. This model approximates $u(t, X)$ across the discretised time-steps from $n = 0...N$ employing the same neural network at each time-step, thereby reducing the number of learnable parameters. Furthermore, the models in [27], [26] and, [29] use a separate network to approximate $Z_n = Du(t_n, X_n)$ whereas our model uses automatic differentiation of $u(t, X)$ to calculate this value, further reducing the complexity of the neural network set-up. Also, with the loss function as defined in 3.4 including a term to minimise the error for the calculations of $Z_N$, we obtain an accurate approximation of $Z_N$ which is a useful value in option contract valuation as banks use it to hedge their positions in the underlying asset. In section 3.3 we show the benefits of including $Z_N$ in the loss.

From section 3.1.8 we can identify a few drawbacks with the model. The model uses a fully connected feed-forward architecture. With the development of deep learning to solve problems in computer vision, researchers have identified parallels in these approaches which can be used to solve optimal control problems in a reasonable amount of time. This is evident in Sirignano et al. [28] which uses recurrent neural networks in the form of LSTMs (Long Short Term Memory) to solve time dependent PDEs and the work by Batuhan Guler, Alexis Laignelet, and Panos Parpas [30] which suggest that using a more evolved Res-Net (Residual Neural Network) style neural

network architecture like the one proposed by Ciccone et al. [33]. Furthermore, they also suggest that discretisation techniques could potentially reduce the time taken to train the model to reach similar levels of loss as in the above experimental results. The multi-level Monte Carlo discretisation scheme proposed by Giles [42] during training could reduce the computational complexity while maintaining similar accuracy levels. We explore these techniques in the following sections by implementing them in our model.

## 3.2 Nais-Net Architecture

The Nais-Net (Non-Autonomous Input-Output Stable Network) architecture as proposed by Ciccone et al. [33] is defined as a general residual network (as proposed by He et al. [43]) where a block is the unrolling of a time invariant system and non-autonomy is implemented by having the external input applied to each of the unrolled processing stages in the block through skip connections (See figure 3.5). To further explain the Nais-Net architecture, we first define residual neural networks.

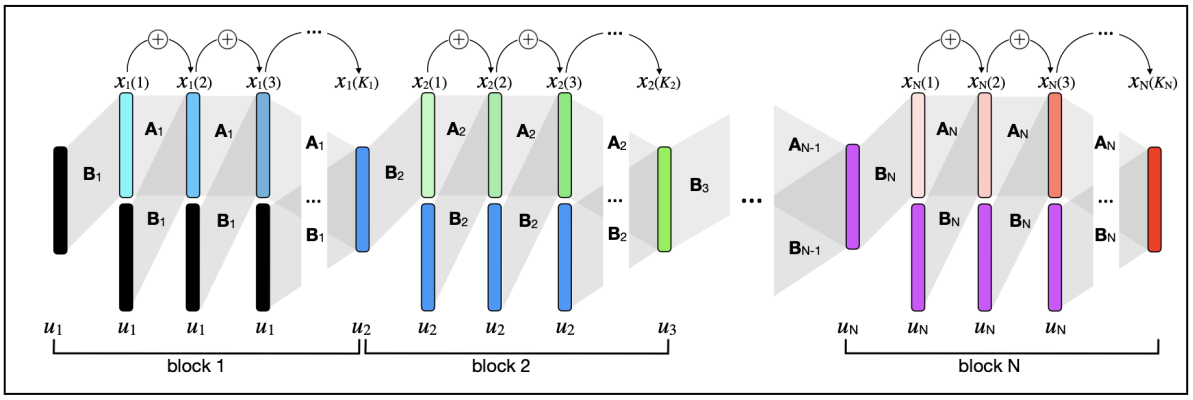


**Figure 3.5:** This figure represents the fully connected Nais-Net architecture from Ciccone et al. [33]. Each block represents a time-invariant iterative process as described in equation 3.10. The skip connections from $u_i$ to all layers in each block makes it non-autonomous. The blocks are chained together by passing the output from the final layer of one block to another.

### 3.2.1 Residual Networks

Consider a non-linear transformation $F$. Let $x(k)$ and $\theta(k)$ be the output and weights of layer $k$ respectively. Then a residual network is defined as follows:

$$x(k+1) = x(k) + F(x(k), \theta(k)) \tag{3.8}$$

Here, to compute the output representation for layer $k+1$ we add $x(k)$ to a linear transformation $F$ on $x(k)$ which depends on a set of parameters $\theta(k)$. A visual representation of the layers is given in the figure 3.6. These connections in residual neural networks help prevent the vanishing gradient problem faced by deep feed-forward neural networks.

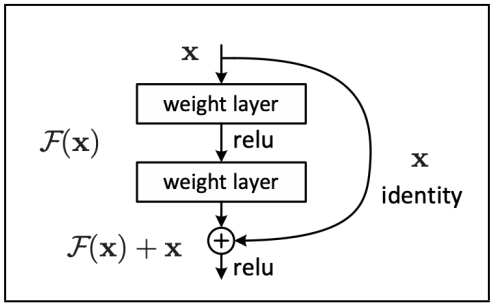


**Figure 3.6:** Residual Network block as proposed by He et al. [43]

The authors in [33] identify that the transformation defined in equation 3.8 is a forward Euler discretisation of the following ordinary differential equation:

$$\dot{x}(t) = f(x(t), \Theta) \quad \text{if} \quad \theta(k) = \Theta \quad \text{for all} \quad 1 \leq k \leq K \tag{3.9}$$

This leads to forward stability issues as described by Zhang et al. [44]. Nais-Net solves this issue.

### 3.2.2 Fully Connected Nais-Net

A fully connected Nais-Net layer is defined as follows:

$$x(k+1) = x(k) + h\sigma(Ax(k) + Bu + b) \tag{3.10}$$

where $x \in \mathbb{R}^n$ is the output of a layer , $u \in \mathbb{R}^m$ is the network input, $h > 0$, $A \in \mathbb{R}^{n \times n}$ is the state transfer matrix, $B \in \mathbb{R}^{n \times m}$ is the input transfer matrix, $b \in \mathbb{R}^n$ is the bias and $\sigma$ is the activation function for each layer. Furthermore, $A$ is restricted to be symmetric and negative definite by parameterising it as follows:

$$A = -R^T R - \epsilon I$$

where $I \in \mathbb{R}^{n \times n}$ is the Identity, $R \in \mathbb{R}^{n \times n}$ is learned by the model and $\epsilon$ is a hyper-parameter which is chosen such that $0 < \epsilon << 1$. Also, a bound on the Frobenius norm, $||R^T R||_F$ is enforced through algorithm 2 below to enforce stability and robustness by forcing the the weights to stay within the feasible set.

**Algorithm 2:** Fully Connected Re-projection

**Input:** $R \in \mathbb{R}^{n \times n}$, $\tilde{n} \leq n$, $\delta = 1 - 2\epsilon$, $\epsilon \in (0, 0.5)$.
**if** $\|R^T R\|_F > \delta$ **then**
  $\tilde{R} \leftarrow \sqrt{\delta} \frac{R}{\sqrt{\|R^T R\|_F}}$;
**else**
  $\tilde{R} \leftarrow R$;
**Output:** $\tilde{R}$

Batuhan Guler, Alexis Laignelet, and Panos Parpas [30] suggest that this architecture could yield better results than the experiments in section 3.1.8. Therefore, we implement this architecture in place of the fully connected feed-forward network.

### 3.2.3 Results

We use the same set-up as in section 3.1.8 but use the fully connected Nais-Net architecture with $\epsilon = 0.1$. We train the model using sinusoidal, tanh and ReLU activation as in [33] it has been shown to provide a stable and robust architecture using these activation functions. Figure 3.7 and 3.8 show that the model with sinusoidal activation as used originally by Raissi [25] is significantly better at approximating the function $u(t, X_t)$. From the training losses in figure 3.8 we see that the model converges to different minima for sinusoidal, tanh and relu activation functions.

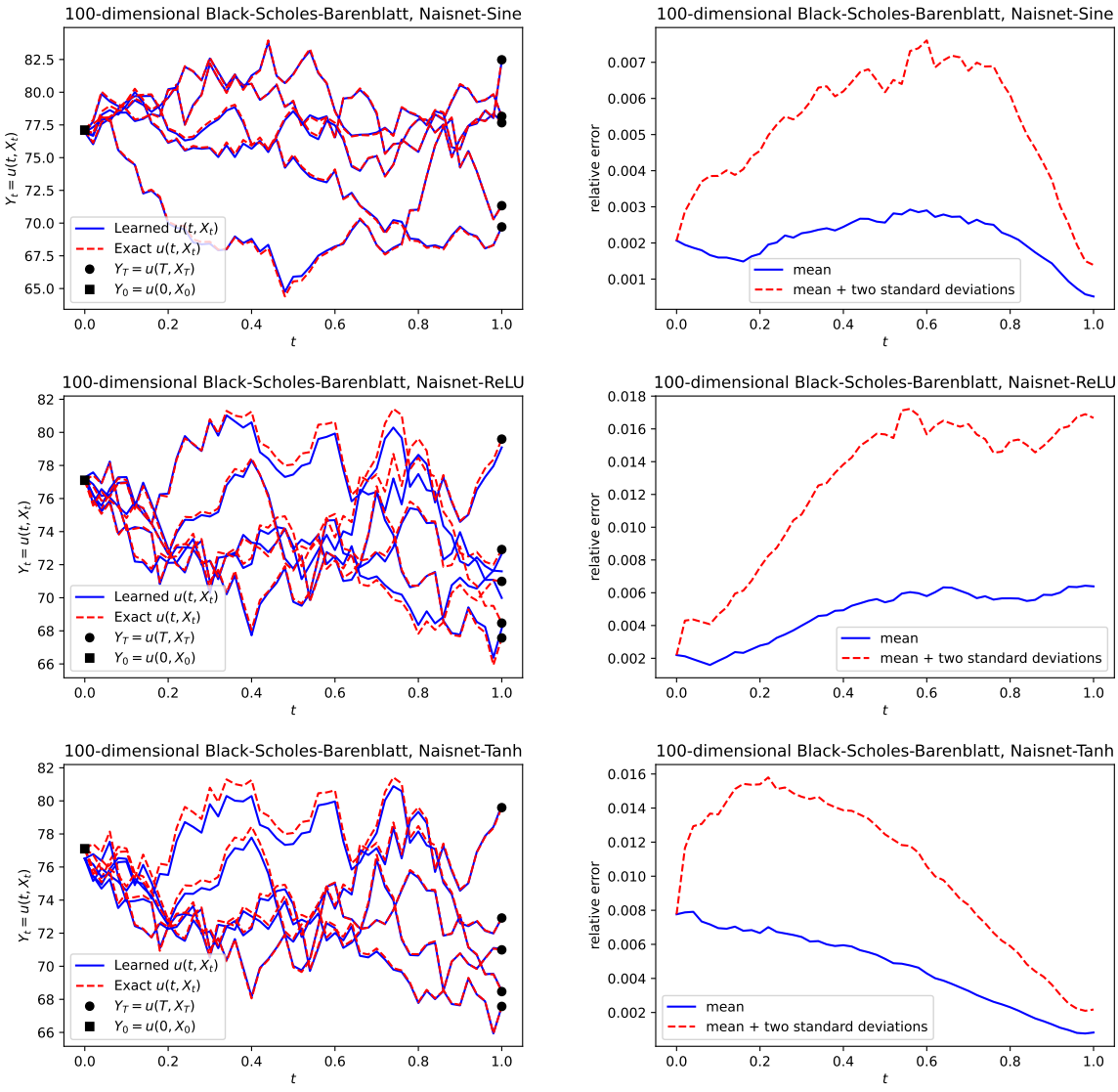


**Figure 3.7:** These figures from the top represent the predictions of the Nais-Net model for 5 realisations of the underlying process and the relative error of the predictions across 100 realisations for different activation functions after training the model for 25k iterations with a learning rate of $10^{-3}$ for the first 20k iterations and $10^{-5}$ for the last 5k iterations with the same parameters as in section 3.1.8.

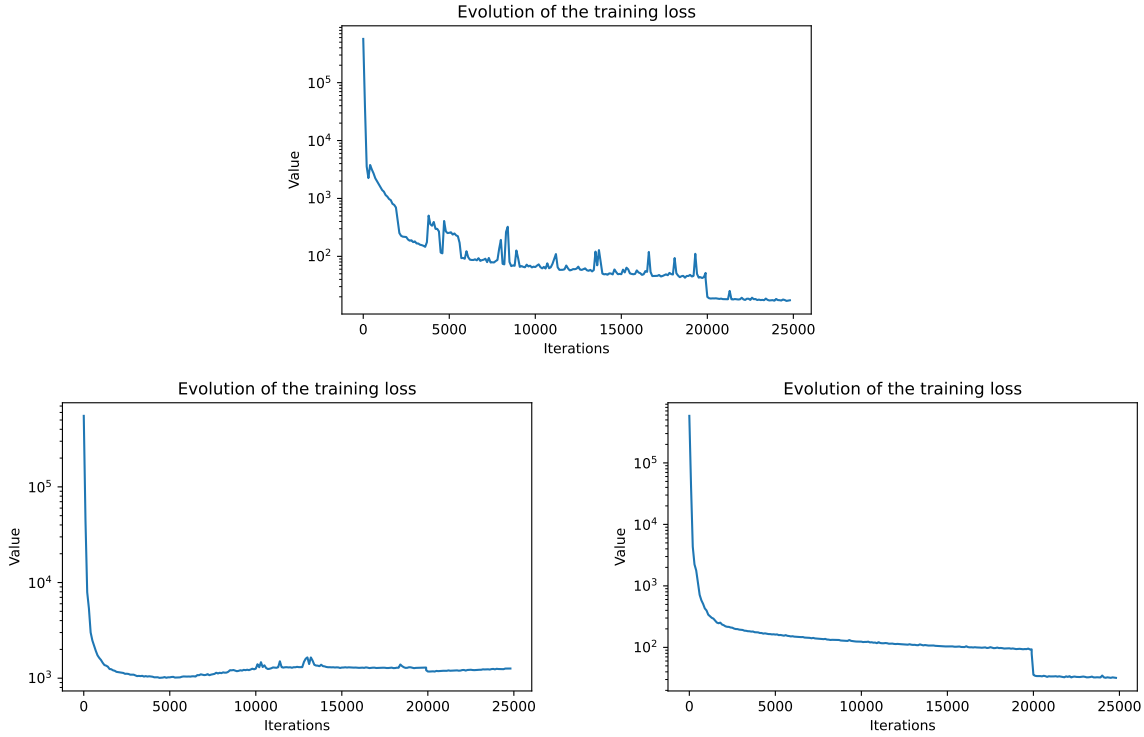


**Figure 3.8:** The top figure, the bottom left figure and the bottom right figure represent the training loss of the Nais-Net model using Sinusoidal activation, ReLU activation and Tanh activation respectively.

We observe that the Nais-Net model with sinusoidal activation produces similar levels of relative error across predictions after training for 25k iterations, with a training time of 4625s, compared to 13497s for the same model parameters as our experiments in section 3.1.8. This could be because of the weights being constrained in the re-projection step as explained in section 3.2.2.

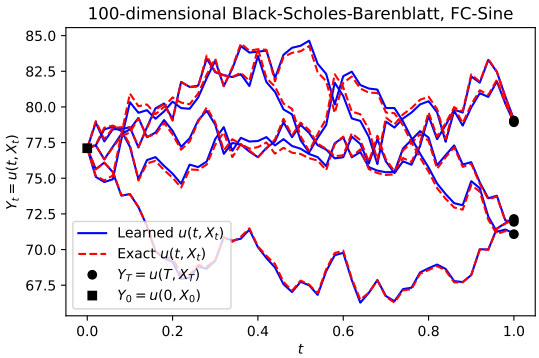


**Figure 3.9:** This figure represents the predictions for 5 realisations of the underlying process on training the Fully Connected Feed-Forward model in section 3.1.8 with the same training scheme as in section 3.2.3.

We also trained the fully connected feed-forward model with the same training

scheme across 25k iterations and obtained the results in figures 3.9 and 3.10. Although the fully connected model produces a lower error difference of magnitude $\approx 5e^{-4}$ for the initial time step, the error across all subsequent time steps is higher by a larger magnitude. The improvement in training time from the initial experiments in section 3.1.8 for the same level of accuracy, along with the improved performance compared to the feed-forward model with the same training scheme, and the fact that the Nais-Net model architecture has been tested to be stable and robust as proved in Ciccone et al. [33], suggests that this architecture is a better alternative to the feed-forward architecture used in our initial experiments.

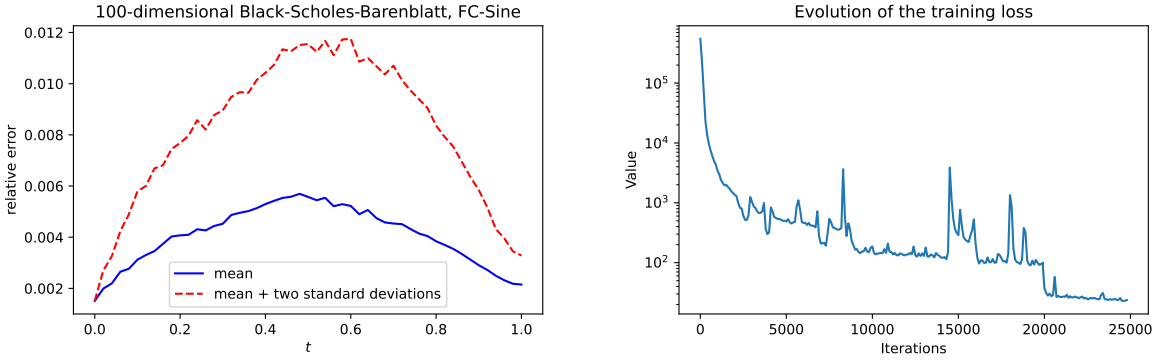


**Figure 3.10:** These figures from left to right represent the relative error across 100 realisations and the loss on training the Fully Connected Feed-Forward model in section 3.1.8 with the same training scheme as in section 3.2.3.

## 3.3 Loss Function

The models by Han et al. [26] [27], Sirignano et al. [28], and Beck et al. [29] do not use the control variate term in their loss function while training their different models. On training our model with a loss function given in 3.4 not including the control variate term $\sum_{m=1}^{M} |Z_N^m - g'(X_N^m)|^2$, we get the results in figure 3.11.

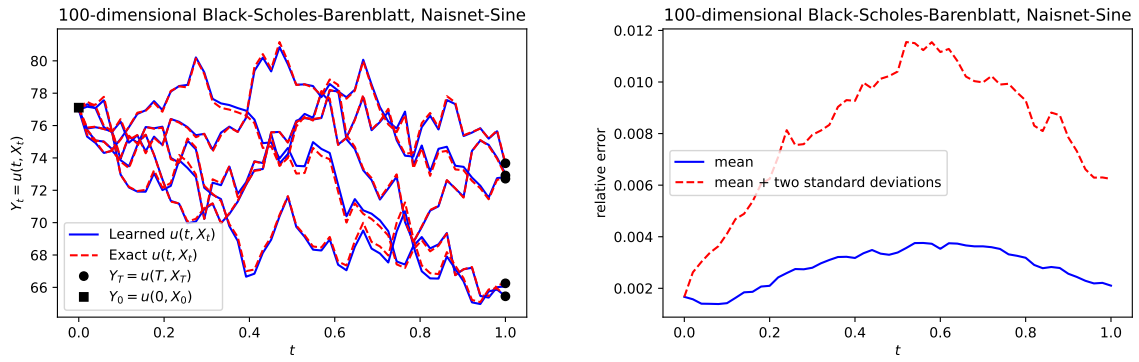


**Figure 3.11:** Predictions of different realisations of $X_t$ and relative error across 100 realisations of $X_t$ without including the control variate squared error term $\sum_{m=1}^{M} |Z_N^m - g'(X_N^m)|^2$ in the loss function.

The figure 3.11 shows that our model approximates $Y_0$ with the same relative error as the case we include the control variate in the loss, which can be seen in figure 3.7. We can infer that the minimisation of the control variate is not significant for our model to approximate $Y_0$, which is in line with the other models from the papers mentioned earlier that only approximate $u(0, X_0)$ or $Y_0$ and do not include the control variate error in their loss having a similar relative error as our model. From this, we can infer that the ability of our model to be able to predict $u(t, X_t)$ over time $t = 0...T$ does indeed depend on the minimisation of the error in the control variate $Z$.

From equation 3.2 we know that $Z$ is the first derivative of $Y_t$ with respect to the underlying asset price $X_t$. Also, from the Black Scholes differential equation defined in equation 2.10 we can see that the Taylor expansion of the stochastic process representing the price of the contract can be expanded to include higher derivatives of the price $Y_t$ with respect to the underlying $X_t$. For example, The second derivative is termed as gamma and is used by banks to hedge their exposure to the convexity of the price movements of the contract. Therefore, minimising this term in the loss function could potentially further improve the accuracy of our model's predictions.

## 3.4 Multi-Level Monte Carlo Technique

On logging the training time of 100 iterations by changing the dimensions and number of discretisation time steps for the stochastic processes in equation 3.3 in tables 3.1 and 3.2 respectively, we see that there is a significant increase in time for changes in the number of discretisation steps as compared to dimensions. This is because the number of neural networks used increases with an increase in the number of time steps, which leads to the observation that the time taken to train the model for 100 iterations scales geometrically with the number of discretisation steps, whereas since only the first layer of the model architecture (see figure 3.2) depends on the number of dimensions of the underlying stochastic process we only see a significant increase in training time when there is a large increase in the number of dimensions (as seen when the dimensions increase from 250 to 1000). Due to this observation, we first aim to identify methodologies to reduce the time due to the discretisation time steps in this section and in section 3.5 we explore the bias variance trade-off on changing the architecture. Furthermore, as results from Batuhan Guler, Alexis Laignelet, and Panos Parpas [30] suggest that there is scope to investigate numerical discretisation techniques, we explore the Multi-Level Monte Carlo discretisation approach as proposed by Giles [42][45].

### 3.4.1 Definition

The Multi-Level Monte Carlo (MLMC) was introduced by Giles [42][45] to reduce the computational complexity of estimating an expected value arising from a stochastic differential equation. In our case the expected value refers to the approximation of $u(t, x)$ and the discretised version of the stochastic differential equations refer to

| Dimensions | Time |
|---|---|
| 1 | 16.29s |
| 10 | 16.46s |
| 25 | 16.59s |
| 50 | 16.77s |
| 100 | 17.70s |
| 250 | 20.18s |
| 500 | 27.87s |
| 1000 | 58.62s |

**Table 3.1:** Training time of 100 iterations of Nais-Net model for different dimensions of the stochastic processes involved keeping $N$, the number of discretisation time-steps constant at 50 and other parameters the same as section 3.1.8.

| Time-Steps | Time |
|---|---|
| 1 | 0.76s |
| 10 | 3.87s |
| 25 | 8.98s |
| 50 | 17.70s |
| 100 | 35.42s |
| 250 | 90.57s |
| 500 | 181.00s |
| 1000 | 364.43s |

**Table 3.2:** Training time of 100 iterations of Nais-Net model for different time-step discretisation levels of the stochastic processes involved keeping $D$, the number of dimensions constant at 100 and other parameters the same as section 3.1.8.

the equations in 3.3. Consider the following stochastic differential equation:

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dW_t$$

with $\mu$ and $\sigma$ the drift and volatility terms respectively, $W$ a Wiener process. Also, suppose there exists a function $g$ which determines the terminal condition i.e, $g(X_T)$ for terminal time $T$. An Euler discretisation of the above SDE leads to:

$$X_{n+1} = X_n + \mu(t, X_t)h + \sigma(t, X_t)\Delta W_t$$

Giles [42][45] suggests for the problem involving reducing the expected mean squared error of the estimates of $g(X_T)$ across $N$ simulations the computational cost of achieving an accuracy in the order of $O(\epsilon)$ can be reduced from $O(\epsilon^{-3})$ to $O(\epsilon^{-2}(log(\epsilon))^2)$ by the use of a multi-level method that reduces the variance leaving unchanged the bias due to the Euler discretisation. This is done by using a sequence of different time-steps of the form:

$$h = (L)^{-1}T \text{ for integers } L \geq 2$$

In our case, the idea is to train the model with a small number of discretisation time-steps which has a lower overall accuracy but significantly lower computational cost as evident in table 3.2 and gradually increase this number at regular intervals to improve the accuracy during the process of training the model.

### 3.4.2 Geometric Multi Level Monte Carlo

The Geometric Multi Level Monte Carlo technique entails geometrically increasing the number of discrete time-steps in a series of simulations. That is,

$$h_l = (L)^{-l}T,$$

for any integer $L \geq 2$ for $l = 1, 2, 3...$ until $h_l = N$, where $N$ is the required number of time-steps. For example, if $L = 2$ and $N = 64$, we train the model for an equal number of iterations at discretisations levels such that the number of time intervals are 2, 4, 8, 16, 32 and 64 (See figure 3.12 for the results from this experiment).

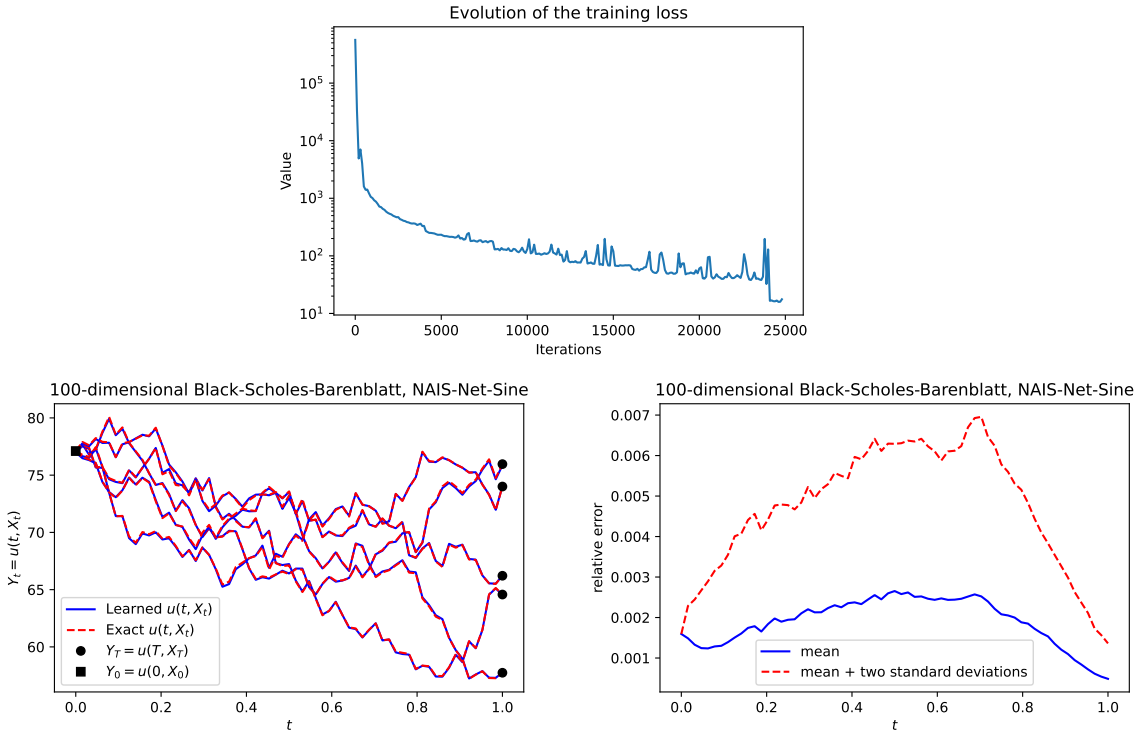


**Figure 3.12:** Training loss, predictions for different realisations of $X_t$ and relative error across predictions of 100 realisations for $X_t$ after training our model using a Geometric Multi-Level Monte Carlo by doubling the number of discretisation time-steps starting at $L = 2$ every 4k iterations which yields the sequence 2, 4, 8, 16, 32 and 64 using the Nais-Net architecture with sinusoidal activation for 24k iterations using learning rate $1e^{-3}$ and 1k iterations using learning rate $1e^{-5}$ and Adam optimisation until convergence with a total training time of 1948.67s.

### 3.4.3 Non-Geometric Multi Level Monte Carlo

The Non-Geometric Multi Level Monte Carlo Technique entails increasing the number of discrete time-steps in a series of simulations by an increasing sequence of numbers. That is,

$$h_l = (L)^{-1}T,$$

for any integer $L \geq 2$ for $l = 1, 2, 3...$ until $h_l = N$, where $N$ is the required number of time-steps. For example, we can start with $L = 3$ and $N = 50$, we train the model for an equal number of iterations at discretisations levels such that the number of time intervals are 3, 5, 11, 23, 50 (See figure 3.13 for the results from this experiment).

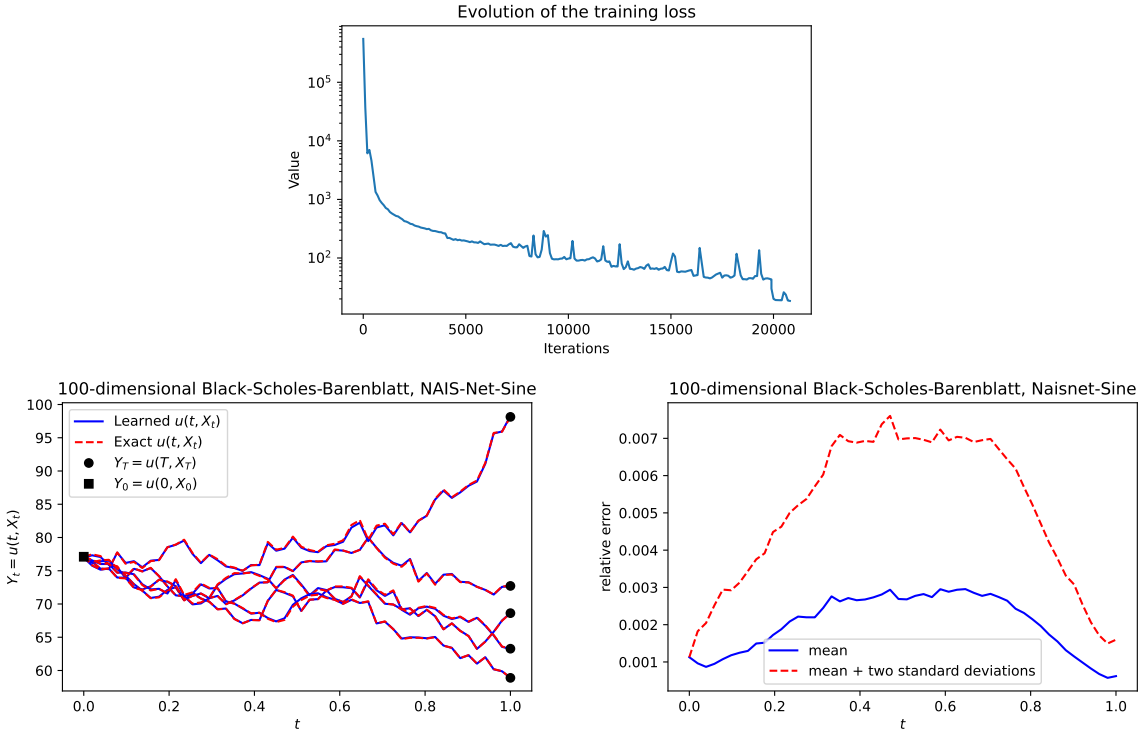


**Figure 3.13:** Training loss, predictions for different realisations of $X_t$ and relative error across predictions of 100 realisations for $X_t$ after training our model using a Non-Geometric Multi-Level Monte Carlo by changing the number of discretised time-steps using the formula $L = \lceil 50^{(\frac{iteration/4000+1}{5})} \rceil$, which yields the sequence 3, 5, 11, 23 and 50 using the Nais-Net architecture with sinusoidal activation for 20k iterations using learning rate $1e^{-3}$ and 1k iterations using learning rate $1e^{-5}$ and Adam optimisation until convergence with a total training time of 1534.82s

From our experiments we have presented the best performing combinations of discretisation steps in both the geometric and non-geometric cases. In comparison to the results in figures 3.7 and 3.8 for the single-level case we observe that the convergence of the loss function in the multi-level case is smoother and reaches a minima in less than half the time as the single-level case while achieving better relative error levels. On comparing the two multi-level approaches we see that the geometric multi-level approach results in a slightly better loss convergence which is likely due to the fact that the model is trained for longer with a larger number of discretisation time-steps. This difference however does not yield significant differences in the accuracy and relative error metrics which can be seen in figures 3.13 and 3.12. Given the faster training time in the non-geometric case, we use this method in the subsequent experiments in our report.

### 3.4.4 Generalisation at Finer Discretisation Levels

On testing the ability of the trained model to predict the approximation of $u(t, X_t)$ for an underlying process $X_t$, which is discretised to 1000 time-steps, we obtain the results as in figure 3.14. We observe that our model can approximate $u(t, X_t)$ for

a fine level of discretisation of 1000 time-steps with a similar level of relative error as it did for the highest level of discretisation of 50 time-steps that it was trained on. If we were to train our model at the 1000 time-step discretisation level, every 100 iterations would take $\approx 365$s. In contrast, with our implementation, we are now able to produce similar levels of relative error as the 50 time-step discretisation level (see figure 3.13) at the same level of computational cost by fully training the model in $\approx 1535$s. Also, from figure 3.14, we see that the Nais-Net model with multi-level discretisation (left) outperforms the initial feed-forward model (right) in approximating $u(t, X_t)$ with a lower mean error and standard deviation across timesteps t. Therefore this improvement in generalisation can be attributed to the Nais-Net architecture and the reduction in training time while maintaining this accuracy can be attributed to the MLMC scheme.

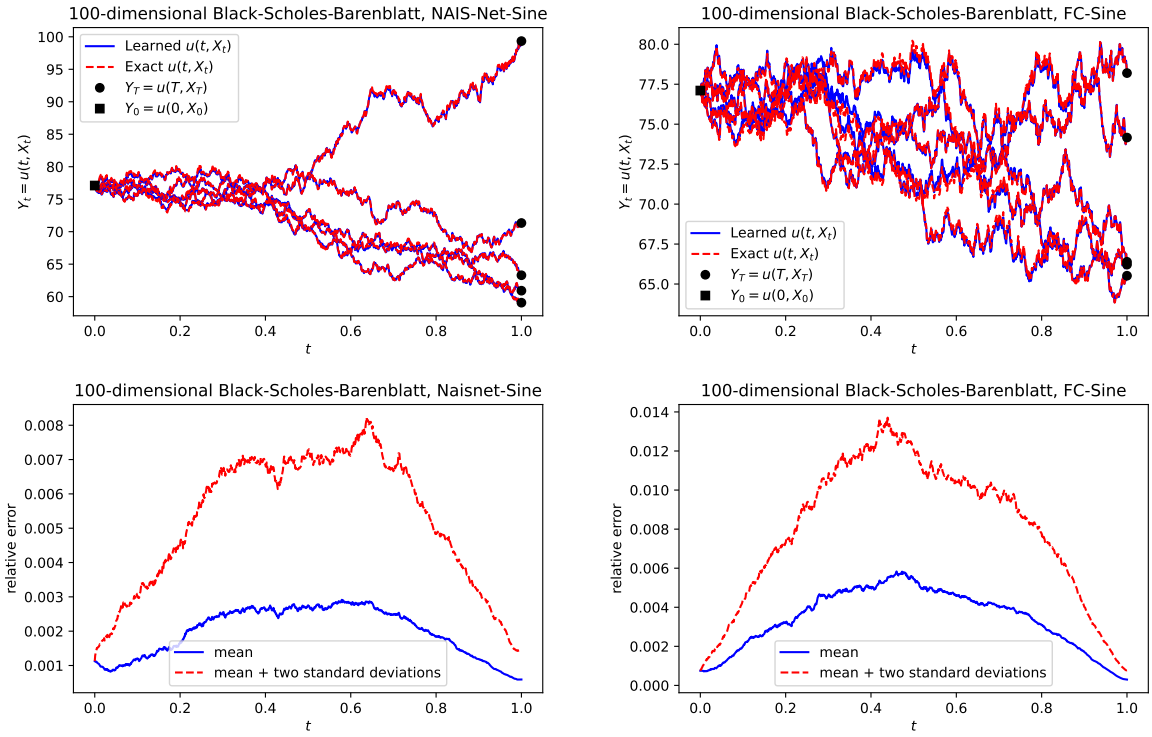


**Figure 3.14:** These figures on the left represent the predictions of the Nais-Net model trained using Non-Geometric Multi Level Monte Carlo above for 5 realisations of an underlying stochastic process with 1000 discretised time steps and the relative error of the predictions across 100 realisations whereas the right represents the same for the Fully-Connected Feed Forward model as in 3.1.8 trained for 100k iterations.

Furthermore, our model provides a more functional alternative as compared to other state of the art models as proposed Han et al. [27] [26] and Beck et al. [29] which do not generalise at finer discretisation levels due to which they have to be re-trained with a higher computational cost due to the increase in the number of unique neural networks with the discretisation steps whereas our model uses the same neural network at each time-step. Also, these models only predict the value of $Y_t$ at $t = 0$ and calculate the future values by time-marching using the Euler-Maruyama scheme which is computationally intensive at fine discretisation levels

whereas predicting $Y_t$ is instantaneous with our model once it is fully trained.

## 3.5 Bias Variance Trade-off

Now that we have a model which generalises well with a reasonable training time, we explore the bias-variance trade-off by changing the architecture and number of realisations used in each training step. We first train 20 models for the same equation and terminal condition as the previous sections using different configurations and the non-geometric MLMC scheme as described in section 3.4.3. We then obtain the prediction $Y_{t,i}$ for timesteps $t = 0, ...N$ with $N = 50$, and models $i = 1, ...N_m$ with $N_m = 20$ across $M = 10000$ realisations of the underlying process $X_t$. We also calculate the true values $\widehat{Y}_t$ across for the realisations of $X_t$.

| Number of Layers | Number of Neurons per Layer | Number of realisations | Bias$^2$ | Variance | Total Error |
|---|---|---|---|---|---|
| 1 | 256 | 100 | 0.5598 | 0.3856 | 0.9454 |
| 2 | 256 | 100 | 0.0079 | 0.0017 | 0.0096 |
| 2 | 100 | 100 | 0.0050 | 0.0048 | 0.0098 |
| 2 | 64 | 100 | 0.0164 | 0.0012 | 0.0176 |
| 2 | 256 | 50 | 0.0043 | 0.0042 | 0.0085 |
| 2 | 256 | 200 | 0.0032 | 0.0061 | 0.0093 |
| 2 | 256 | 500 | 0.0094 | 0.0011 | 0.0105 |
| 3 | 256 | 100 | 0.0029 | 0.0058 | 0.0087 |
| 3 | 100 | 100 | 0.0043 | 0.0043 | 0.0086 |
| 3 | 100 | 200 | 0.0060 | 0.0028 | 0.0088 |
| 3 | 100 | 50 | 0.0069 | 0.0025 | 0.0094 |
| 4 | 256 | 100 | $6.4\times10^{-4}$ | 0.0127 | 0.0133 |
| 4 | 256 | 1 | 0.1841 | 0.0911 | 0.2752 |
| 4 | 256 | 10 | 0.0441 | 0.0066 | 0.0507 |
| 4 | 256 | 50 | 0.0121 | $6.6\times10^{-4}$ | 0.0128 |
| 4 | 256 | 75 | 0.0059 | 0.0031 | 0.0090 |
| 4 | 256 | 150 | 0.0043 | 0.0046 | 0.0089 |
| 4 | 256 | 250 | 0.0032 | 0.0059 | 0.0092 |
| 4 | 256 | 500 | 0.0026 | 0.0070 | 0.0096 |
| 4 | 100 | 100 | 0.0049 | 0.0049 | 0.0098 |

**Table 3.3:** Bias-Variance Tradeoff Analysis for Different Models

We define the bias and variance of the i$^{\text{th}}$ model's predictions as follows:

$$\text{bias}_i^2 = \frac{1}{N+1}\sum_{s=0}^{N}(Y_{s,i} - \bar{\widehat{Y}}_s)^2 \quad \text{and} \quad \text{variance}_i = \frac{1}{N+1}\sum_{s=0}^{N}(Y_{s,i} - \bar{Y}_{s,i})^2 \tag{3.11}$$

where,

$$\bar{\widehat{Y}}_t = \frac{1}{M}\sum_{m=0}^{M}(\widehat{Y}_t^m) \quad \text{and} \quad \bar{Y}_{s,i} = \frac{1}{M}\sum_{m=0}^{M}(Y_{t,i}^m).$$

The analysis of the bias-variance trade-off (See table 3.3) across different model configurations indicates that increasing the number of layers, keeping all else constant, generally decreases bias but increases variance resulting in over-fitting, except in the 1 layer case where the model fails to generalise well. A higher number of neurons also results in a decrease in bias with an increase in variance. Adding more noise to the input by increasing the number of realisations consistently decreases bias and increases variance but not as much as the other parameters. Notably, the model with 2 layers, 256 neurons per layer, and 50 realisations achieved the lowest total error of 0.0085, suggesting an optimal balance between bias and variance. Thus, while the initial model (4 layers, 256 neurons, 100 realisations) performed well, fine-tuning the number of layers and increasing the realisations can further improve model accuracy. We therefore use this configuration for future experiments and the algorithms in chapter 4.

## 3.6 Correlated Underlying Processes

In a real world setting, underlying assets in a portfolio tend to have a quantifiable correlation with each other. We can incorporate the correlation of the underlying process $X_t$ by using a Cholesky decomposition of the correlation matrix of the assets.

### 3.6.1 Cholesky Decomposition

Consider two Wiener processes $w_1$ and $w_2$ with correlation $\rho$. The correlation matrix can be decomposed into a lower triangular and upper triangular part such that:

$$Corr(w_1, w_2) = (LL^T) = \underbrace{\begin{pmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 1 & \rho \\ 0 & \sqrt{1-\rho^2} \end{pmatrix}}_{L^T} = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$$

This is termed as a Cholesky decomposition. The correlation can be incorporated to the processes $w_1(t)$ and $w_2(t)$ using the Cholesky decomposition to generate correlated processes in $\widetilde{W_t}$ as follows:

$$\widetilde{W_t} = \begin{pmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{pmatrix} \begin{pmatrix} w_1(t) \\ w_2(t) \end{pmatrix} = \begin{pmatrix} w_1(t) \\ \rho w_1(t) + \sqrt{1-\rho^2} w_2(t) \end{pmatrix} \tag{3.12}$$

Since the process defined by $X_t$ is governed by a Wiener process, we can apply this concept to the equations in 3.6 if the assets represented in $X_t$ are correlated. Consider $D$ assets with the following correlation matrix such that $\rho_{ij}$ represents the correlation between asset $i$ and $j$:

$$Corr(x_1, x_2, ...x_D) = \begin{pmatrix} 1 & \rho_{12} & \rho_{13} & \cdots & \rho_{1,D-1} & \rho_{1D} \\ \rho_{12} & 1 & \rho_{23} & \cdots & \rho_{2,D-1} & \rho_{2D} \\ \rho_{13} & \rho_{23} & 1 & \cdots & \rho_{3,D-1} & \rho_{3D} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \rho_{1,D-1} & \rho_{2,D-1} & \rho_{3,D-1} & \cdots & 1 & \rho_{D-1,D} \\ \rho_{1D} & \rho_{2D} & \rho_{3D} & \cdots & \rho_{D-1,D} & 1 \end{pmatrix}$$

Let the Cholesky decomposition give $Corr(x_1, x_2, ...x_D) = (LL^T)$. Then the equations in 3.6 become:

$$\begin{aligned} dX_t &= \sigma \text{diag}(X_t)\, L dW_t, && t \in [0,T], \\ X_0 &= \xi, \\ dY_t &= r(Y_t - Z_t' X_t)\, dt + \sigma Z_t' \text{diag}(X_t)\, L dW_t, && t \in [0,T], \\ Y_T &= g(X_T). \end{aligned} \tag{3.13}$$

We discretise the above set of equations using the Euler Maruyama Scheme as in equation 3.3 and use our model to approximate $Y_t$ as we did for the uncorrelated case. Figures 3.15 and 3.16 show the evolution of the loss function and predictions of our model for correlated underlying asset prices. We observe that our model is even better at predicting $Y_t$ for high dimensional underlying data than low dimensional data.

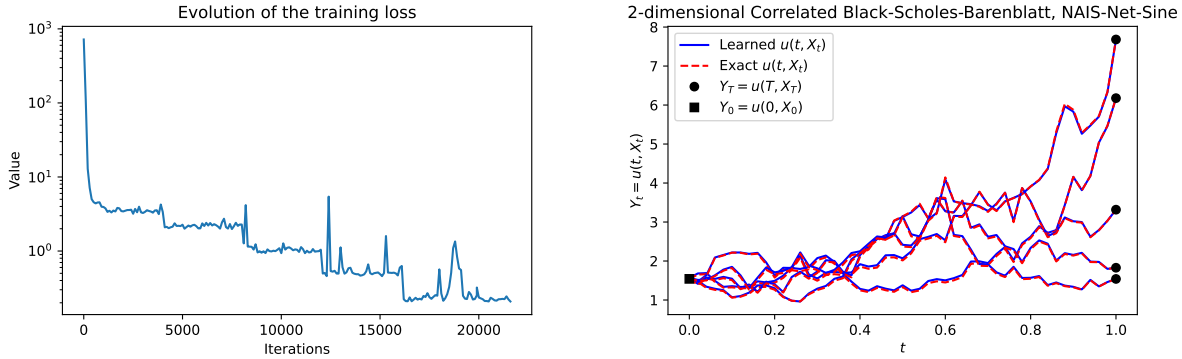


**Figure 3.15:** This figure represents the Loss (left) and the approximation of $u(t,x)$ across 5 realisations of $X_t$ on training the Nais-Net model with sinusoidal activation for 21k iterations using the non-geometric MLMC scheme as detailed in section 3.4.3 for a basket containing two correlated assets with correlation $\rho = 0.5$.

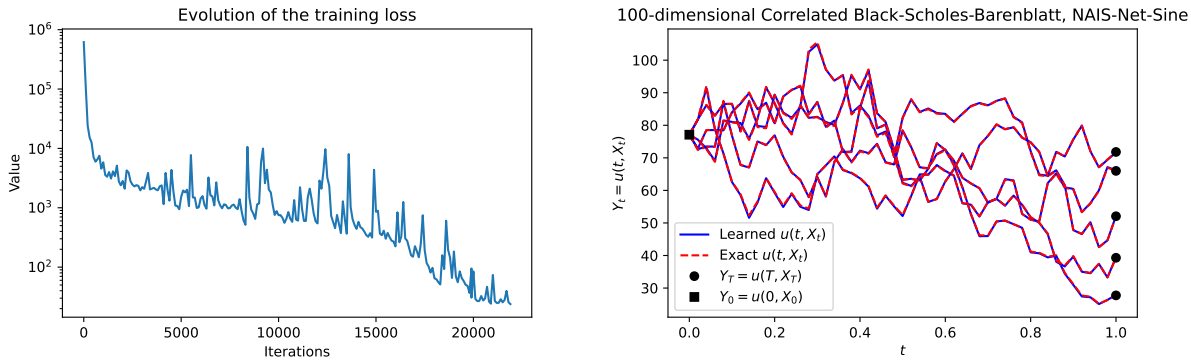


**Figure 3.16:** This figure represents the Loss (left) and the approximation of $u(t,x)$ across 5 realisations of $X_t$ on training the Nais-Net model with sinusoidal activation for 21k iterations using the non-geometric MLMC scheme as detailed in section 3.4.3 for a basket containing 100 correlated assets with correlation $\rho = 0.5$ between each asset.

# Chapter 4

# Risk Management Framework

In this chapter we introduce a methodology to create a risk management framework by using our model to simulate the dynamics of valuation adjustments (xVA) which include Credit Valuation Adjustments (CVA), Debt Valuation Adjustments (DVA) and Funding Valuation Adjustments (FVA) for portfolios consisting of European style Option and Forward Contracts. These calculations have become increasingly important for banks following the 2008 crisis which prompted them to take into account the default of all counterparties in the fair value pricing of their products. We use a similar framework and definition for the valuation adjustments based on the work by Pallavacini et al. [6] and extended by Brigo et al. [46] and Biagini et al. [8] to formulate this problem as a backward stochastic differential equation which we solve using our model. In the following sections we define the aforementioned xVA calculations and propose algorithms to simulate them. We then compare the performance of our model with that of Gnoatto et al. [34], as they also define the valuation adjustments similarly.

## 4.1 Market Setting

In this setup we consider two parties involved in all trading activities, i.e, the bank (B) and the counterparty (C), with the bank being the hedger and underwriter of all financial products and the counterparty being involved in the opposite side of these transactions. We define the clean value as the value of the financial contract without the valuation adjustments and the fair value as the value of the financial contract with the valuation adjustments.

### 4.1.1 Time of Default

Let $\tau^B$ and $\tau^C$ be the times of default of the bank and counterparty respectively. Default refers to the inability of any of these parties to fulfill an obligation. For example, in the bank's case, it may be the failure to make required payments or deliver underlying assets as specified in a financial derivative contract. This could include failing to pay the premium on an option contract, or not delivering the underlying asset on a forward contract. Whereas in the counterparty's case, default could occur

if they fail to provide the necessary collateral for a margin call, or neglect to pay for the received financial instruments, resulting in an inability to meet their financial commitments under the terms of the derivative contracts. The default times are assumed to be exponentially distributed random variables with time dependent intensity. We define the jump processes representing the defaults of the bank and the counterparty as follows:

$$\begin{aligned} H_t^B &= \mathbf{1}_{\{\tau^B \leq t\}} \\ H_t^C &= \mathbf{1}_{\{\tau^C \leq t\}} \end{aligned} \tag{4.1}$$

The cumulative intensity function $\Gamma^B$ and $\Gamma^C$ for the bank and the counterparty over the interval $[0, T]$, where $T$ is the terminal time such that $T < \infty$, is given by:

$$\begin{aligned} \Gamma_t^B &= \int_0^t \lambda_s^B \, ds, \quad t \in [0, T], \\ \Gamma_t^C &= \int_0^t \lambda_s^C \, ds, \quad t \in [0, T]. \end{aligned} \tag{4.2}$$

where $\lambda_t^B$ and $\lambda_t^C$ are non-negative adaptive stochastic processes related to the processes in equation 4.1 by a Poisson random measure $M_t$ as follows:

$$\begin{aligned} M_t^B &= H_t^B - \int_0^{t \wedge \tau^B} \lambda_s^B \, ds, \\ M_t^C &= H_t^C - \int_0^{t \wedge \tau^C} \lambda_s^C \, ds. \end{aligned} \tag{4.3}$$

### 4.1.2 Underlying Assets

Let us define the underlying assets on which the option and forward contracts are priced on. Consider the prices of the $D$ underlying assets $x_1(t), x_2(t), x_3(t)....x_D(t)$ on time $t \in [0, T]$ where $T$ is the terminal time is defined by the random process $X_t \in \mathbb{R}^D$ given by:

$$X_t = \begin{pmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_D(t) \end{pmatrix}$$

Consider a standard Wiener process $W \in \mathbb{R}^D$. We define the process $X_t$ denoting the evolution of the prices of the underlying assets by the following stochastic differential equation as in equation 3.1:

$$\begin{aligned} dX_t &= \mu(t, X_t)dt + \sigma(t, X_t)dW_t, \\ X_0 &= \xi. \end{aligned} \tag{4.4}$$

where $\mu : \mathbb{R}^+, \mathbb{R}^D \to \mathbb{R}^D$ is a function which defines the drift of the asset prices and $\sigma : \mathbb{R}^+, \mathbb{R}^D \to \mathbb{R}^{D \times D}$ is a function representing the volatility of the underlying asset prices. If the assets are correlated, we can use the Cholesky decomposition

of the correlation matrix of the asset prices as in section 3.6 to obtain the lower triangular matrix $L \to \mathbb{R}^{D\times D}$ of the correlation matrix to get the following stochastic differential equations for $X_t$:

$$\begin{aligned} dX_t &= \mu(t, X_t)dt + \sigma(t, X_t)LdW_t, \\ X_0 &= \xi. \end{aligned} \tag{4.5}$$

We assume that the underlying does not pay any dividends.

### 4.1.3 Cash Accounts

We assume that there exists cash accounts as per the agreements established between the counterparty and the bank to mitigate credit risk due to default. The cash account $B_t$ for $t \in [0, T]$ and $x \in \{B, C\}$ is assumed to have unitary value initially and is represented using a bounded stochastic process representing a rate of return $r_t$ as follows:

$$B_t = e^{\int_0^t r_s\, ds} \tag{4.6}$$

### 4.1.4 Risky Bonds

We define two bonds with the possibility of default with maturity $t < T$ representing the obligations of the bank and the counterparty. The dynamics of the bond values are given by the processes $P^B$ and $P^C$ for the bank and counterparty respectively as follows:

$$\begin{aligned} \mathrm{d}P_t^B &= r_t^B P_t^B\, \mathrm{d}t - P_{t-dt}^B\, \mathrm{d}M_t^B \\ \mathrm{d}P_t^C &= r_t^C P_t^C\, \mathrm{d}t - P_{t-dt}^C\, \mathrm{d}M_t^C \end{aligned} \tag{4.7}$$

where $r_t$ represents a rate of return as in equation 4.6 and $M_t$ is a compensated Poisson random measure as defined in equation 4.3.

### 4.1.5 Claims

We consider the payout of a financial contract as a contingent claim which is a stream of payments defined by the process $\bar{A}_t^d$ for each asset given by:

$$\bar{A}_t^d = \mathbf{1}_{\{t<\tau\}} A_t^d + \mathbf{1}_{\{t\geq\tau\}} A_T^d \tag{4.8}$$

where $A_t^d$ is the claim without taking into consideration the default of either counterparty and $A_T^d$ is the last payment before default for $d = 1....D$.

### 4.1.6 Collateral

Collateral is an asset that a borrower offers to a lender to secure a loan or other credit. The collateral serves as protection for the lender against the borrower's default. It is posted between counterparties to mitigate credit and default risk. We define a process $C_t = f(Y_t)$ that is a function of the value of the clean value or fair

value including the valuation adjustments as defined in section 4.2.1. We assume that that the collateral is posted in the form of cash $r^{c,l}$ representing the rate of interest received by the bank that posted collateral to a counterparty and $r^{b,l}$ representing the rate of interest received by the counterparty that posted collateral to a bank during a transaction.

## 4.2 xVA Framework

### 4.2.1 Clean Market Portfolio Dynamics

For all our calculations, we assume a clean market. Here, we divide the value of a financial contract into a clean value and a valuation adjustment. A clean value or price for a financial contract is the value accepted by the counterparties involved in the transaction, assuming perfect collateralisation and the transaction being default-free; however, a margin risk period, which may occur when a counterparty is on the verge of bankruptcy, results in the deviation of the value of the contract from the clean value, therefore creating a credit exposure which is quantified by valuation adjustments made to the clean value. This is consistent with current market practices for generating pricing for these financial contracts.

The clean values of financial contracts defined on a portfolio of underlying assets $X_t^d$ for $d = 1....D$ with maturities $T_d$ can be defined by the same approximations that our neural network produces in chapter 3. So these clean values $\widehat{Y}_t^d$ are given by the forward backward stochastic differential equations as follows:

$$\begin{aligned} \mathrm{d}\widehat{Y}_t^d &= r_t \widehat{Y}_t^d \mathrm{d}t + \sum_{k=1}^{d} \widehat{Z}_t^{d,k} \, \mathrm{d}W_t^k - \mathrm{d}A_t^d \\ \widehat{Y}_{T_d}^d &= 0. \end{aligned} \tag{4.9}$$

This can also be written as:

$$\widehat{Y}_t^d = \mathbb{E}\left[ B_t^r \int_{(t,T_d]} \frac{\mathrm{d}A_s^d}{B_s^r} \right] \tag{4.10}$$

with $r$ being the rate of interest used in a clean market setting and $t \in [0, T_d]$.

As the main contracts of interest are Eurpoean call options, European put options and forwards which have only one payoff at expiry given by the function $g_d$, we get:

$$A_t^d = 1_{\{t \geq T_d\}} g_d(X_{T_d})$$

Substituting the above in equation 4.9 we have:

$$\begin{aligned} \mathrm{d}\widehat{Y}_t^d &= r_t \widehat{Y}_t^d \, \mathrm{d}t + \sum_{k=1}^{d} \widehat{Z}_t^{d,k} \, \mathrm{d}W_t^k, \\ \widehat{Y}_{T_d}^d &= g_d(X_{T_d}), \\ \widehat{Z}_t^{d,k} &= \xi_t^d \sigma^{d,k}(t, X_t). \end{aligned} \tag{4.11}$$

with $\xi_t^d$ representing the position of the risky asset in the portfolio and $\sigma^{d,k}(t, X_t)$ representing the function used to obtain the covariance of the $d^{th}$ asset with that of the $k^{th}$ asset with $k = 1, ...., D$.

### 4.2.2 Fair Value Stochastic Differential Equation

We define the following processes required to define a process representing the fair value of a portfolio of financial contracts including valuation adjustments:

$$\begin{aligned} U_t^B &= -\xi_t^B P_{t-dt}^B, \\ U_t^C &= -\xi_t^C P_{t-dt}^C, \\ f(t, Y, C) &= -[(r_t^{f,l} - r_t)(Y_t - C_t)^+ - (r_t^{f,b} - r_t)(Y_t - C_t)^- \\ &\quad + (r_t^{c,l} - r_t)C_t^+ - (r_t^{c,b} - r_t)C_t^-], \\ g_\tau(\widehat{Y}, C) &= \widehat{Y}_\tau + H_{\tau^B}^C(1 - R^C)(\widehat{Y}_\tau - C_{\tau - d\tau})^- \\ &\quad - H_{\tau^C}^B(1 - R^B)(\widehat{Y}_\tau - C_{\tau - d\tau})^+ \end{aligned} \tag{4.12}$$

where,

- $\xi_t^B$ and $\xi_t^C$ represent the positions in the bonds posted by the bank and counterparty.
- $r_t^{f,b}$ and $r_t^{f,l}$ are lending and borrowing rates for non-collateral loans and, $r_t^{c,b}$ and $r_t^{c,l}$ are interest rates on collateral as described in section 4.1.6.
- $P_{t-dt}^B$ and $P_{t-dt}$ represent the dynamics of the bond values issued by the bank and counterparty as described in section 4.1.4.
- $C_t$ represents the collateral process as defined in section 4.1.6 with $C_t = C_t^+ + C_t^-$. Here, $C_t^+$ refers to the collateral posted by the bank to the counterparty and $C_t^-$ is the collateral posted by the counterparty to the bank.
- $Y_t$ is the fair value including the valuation adjustments and $\widehat{Y}_t$ is the clean value as described in section 4.2.1.
- Function $g(\widehat{Y}, C)$ represents the terminal condition of the fair value in terms of the clean value and the collateral process with $H_t^B$ and $H_t^C$ the jump processes representing defaults of the bank and counterparty as defined in section 4.1.1.
- $R^B$ and $R^C$ represent the recovery rates of the bank and counterparty respectively which is defined as the percentage of the collateral that each party can recover in the event of default.

Now, we define the stochastic differential equation to represent the fair value of a portfolio of financial contracts over time $Y_t$ as:

$$\begin{aligned} dY_t &= \sum_{k=1}^{d} Z_t^k \, \mathrm{d}W_t^{k,} - \sum_{d=1}^{D} \mathrm{d}\bar{A}_t^d - (f(t, Y, C) - r_t Y_t) \, \mathrm{d}t \\ &\quad + U_t^B \, \mathrm{d}M_t^B + U_t^C \, \mathrm{d}M_t^C, \\ Y_\tau &= g_\tau(\widehat{Y}, C). \end{aligned} \tag{4.13}$$

Biagini et al. [8] shows that there exists a unique solution to this set of equations given by:

$$Y_t = B_t^r \mathbb{E}\left[\sum_{d=1}^{D} \int_{(t,\tau\wedge T]} \frac{\mathrm{d}\bar{A}_s^d}{B_s^r} + \int_t^{\tau\wedge T} \frac{f(s,Y,C)}{B_s^r}\,\mathrm{d}s + \mathbf{1}_{\{\tau\leq T\}}\frac{g_\tau(\widehat{Y},C)}{B_\tau^r}\right] \tag{4.14}$$

### 4.2.3 xVA Backward Stochastic Differential Equation

Using the equations in 4.11 and 4.13 we define the valuation adjustments process $\text{XVA}_t$ as follows:

$$\begin{aligned} d\text{XVA}_t &= \sum_{k=1}^{d} Z_t^k\,\mathrm{d}W_t^k - a(t,\widehat{Y}_t,\text{XVA}_t)\,\mathrm{d}t, \\ \text{XVA}_T &= 0. \end{aligned} \tag{4.15}$$

where,

$$\begin{aligned} a(t,\widehat{Y}_t,\text{XVA}_t) = &-(1-R^C)(\widehat{Y}_t - C_t)^-\lambda_t^C \\ &+(1-R^B)(\widehat{Y}_t - C_t)^+\lambda_t^B \\ &+(r_t^{f,l}-r_t)(\widehat{Y}_t - \text{XVA}_t - C_t)^+ \\ &-(r_t^{f,b}-r_t)(\widehat{Y}_t - \text{XVA}_t - C_t)^- \\ &+(r_t^{c,l}-r_t)C_t^+ - (r_t^{c,b}-r_t)C_t^- \\ &-(r_t+\lambda_t^C+\lambda_t^B)\text{XVA}_t. \end{aligned} \tag{4.16}$$

The process $\text{XVA}_t$ can be related to equations 4.11 and 4.13 as follows:

$$\text{XVA}_t = \widehat{Y}_t - Y_t. \tag{4.17}$$

This equation admits the following solution as shown in Biagini et al. [8]:

$$\text{XVA}_t = -\text{CVA}_t + \text{DVA}_t + \text{FVA}_t + \text{ColVA}_t \tag{4.18}$$

where,

$$\begin{aligned} \text{CVA}_t &= B_t^r\mathbb{E}\left[(1-R^C)\int_t^T \frac{1}{B_s^r}(\widehat{Y}_s - C_s)^-\lambda_s^C\,\mathrm{d}s\right], \\ \text{DVA}_t &= B_t^r\mathbb{E}\left[(1-R^B)\int_t^T \frac{1}{B_s^r}(\widehat{Y}_s - C_s)^+\lambda_s^B\,\mathrm{d}s\right], \\ \text{FVA}_t &= B_t^r\mathbb{E}\left[\int_t^T \frac{(r_s^{f,l}-r_s)(\widehat{Y}_s - \text{XVA}_s - C_s)^+}{B_s^r}\,\mathrm{d}s\right] \\ &\quad - B_t^r\mathbb{E}\left[\int_t^T \frac{(r_s^{f,b}-r_s)(\widehat{Y}_s - \text{XVA}_s - C_s)^-}{B_s^r}\,\mathrm{d}s\right], \\ \text{ColVA}_t &= B_t^r\mathbb{E}\left[\int_t^T \frac{(r_s^{c,l}-r_s)C_s^+ - (r_s^{c,b}-r_s)C_s^-}{B_s^r}\,\mathrm{d}s\right]. \end{aligned} \tag{4.19}$$

We can define the valuation adjustments in equation 4.19 as follows:

- Credit Valuation Adjustment (CVA) is the adjustment to the fair value of a derivative to account for the counterparty credit risk. It represents the value of counterparty credit risk over the life of the derivative contract. It is calculated as the discounted expected loss due to the counterparty's default, taking into account the probability of default and the potential exposure at the time of default.

- Debt Valuation Adjustment (DVA) is the adjustment to the fair value of a derivative to account for the credit risk of the bank. It represents the market value of the bank's own credit risk. It is calculated as the discounted expected gain due to the entity's own default, considering the probability of default and the potential exposure at the time of default.

- Funding Valuation Adjustment (FVA) is the adjustment to the fair value of a derivative to account for the cost of funding the collateral required to enter into the derivative contract.It is calculated based on the cost of funding for the entity, which may vary over time, and the amount of collateral required throughout the life of the derivative.

- Collateral Valuation Adjustment (ColVA) is the adjustment to the fair value of a derivative to account for the cost or benefit associated with holding collateral. It considers the interest earned on the collateral posted and the funding cost of the collateral received. It is calculated by considering the difference between the interest rate on the collateral and the funding rate of the counterparty, applied to the amount of collateral held or posted over time.

These adjustments can be calculated for portfolios using the clean values computed using the neural network we have defined in chapter 3. From the solution for FVA calculations, as described in equation 4.19, we observe that the presence of a recursive structure which can be implemented by approximating this solution using a neural network, which is particularly useful when extending these calculations to portfolios with high dimensional data. In the following section, we define algorithms to calculate these metrics.

### 4.2.4 xVA Algorithms

We observe that from our definitions in equation 4.19 that the CVA and DVA are similar such that they admit a non recursive solution. So, these processes can be approximated at each time-step using a simulation with the clean values of the financial contracts like options and forwards at each time-step which we obtain from our neural network as explained in chapter 3. In line with the Euler-Maruyama discretisation as defined in equation 3.3, we can discretise the integral in the CVA and

DVA solutions across $N$ timesteps as follows:

$$\begin{aligned}\text{CVA}: \int_t^T \frac{1}{B_s^r}(\widehat{Y}_s - C_s)^- \lambda_s^C \, \mathrm{d}s &\approx \sum_{n=0}^{N} \frac{1}{B_n^r}(\widehat{Y}_n - C_n)^- \lambda_n^C \Delta t_n \\ \text{DVA}: \int_t^T \frac{1}{B_s^r}(\widehat{Y}_s - C_s)^+ \lambda_s^B \, \mathrm{d}s &\approx \sum_{n=0}^{N} \frac{1}{B_n^r}(\widehat{Y}_n - C_n)^+ \lambda_n^B \Delta t_n\end{aligned} \tag{4.20}$$

We can obtain the expectation of these terms by calculating the average across $M$ simulations of the clean values obtained from training algorithm 1. Therefore, we propose the algorithm 3 to predict these metrics.

**Algorithm 3:** CVA and DVA Algorithm

**Input:** Trained model to approximate the clean values $\widehat{Y}_t$ using algorithm 1 with the Nais-net architecture and Non-geometric MLMC;
**Output:** CVA and DVA terms at the initial time-step;
**begin**
  **for** *m = 1* **to** *M* **do**
    Generate approximation of the financial contract's clean values $\widehat{Y}_t^m$;
  Calculate the CVA and DVA terms as follows:

$$\text{CVA} = \frac{1}{M}\sum_{m=1}^{M}\sum_{n=0}^{N} \frac{1}{B_n^r}(\widehat{Y}_n^m - C_n)^- \lambda_n^C \Delta t_n^m$$

$$\text{DVA} = \frac{1}{M}\sum_{m=1}^{M}\sum_{n=0}^{N} \frac{1}{B_n^r}(\widehat{Y}_n^m - C_n)^+ \lambda_n^B \Delta t_n^m$$

The CVA and DVA terms at each time period can be calculated by applying the above algorithm at each time-step. The FVA term due to its recursive structure can be approximated using a neural network at each time-step similar to the set-up used in chapter 3 with the input in this case being the clean values $\widehat{Y}_t$ calculated by using a model trained by algorithm 1. The equation 4.18 can also be approximated using this neural network set-up. This provides banks with a methodology to approximate this PDE that overcomes the curse of dimensionality that traditional methods like least squares and finite differences suffer from, especially for valuation adjustment calculations involving large portfolios of multiple underlying assets. We define algorithm 4 to obtain the approximation for equation 4.18.

For algorithm 4 we need the discretised version of the equations in 4.15 which are defined as follows:

$$\begin{aligned}\text{xVA}_{n+1} &= \text{xVA}_n - a(t_n, \widehat{Y}_n, \text{xVA}_n)\Delta t + (Z_n)^\top \Delta W_n \\ g(\widehat{Y}_T) &= \text{XVA}_T = 0.\end{aligned} \tag{4.21}$$

With the function $a(t, Y, \text{xVA})$ having the same definition as in equation 4.16.

**Algorithm 4:** xVA algorithm

initialise and train a model to obtain the clean values using algorithm 1 with the Nais-net architecture and Non-geometric MLMC;
initialise the xVA model with its architecture and trained model;
**for** *number of iterations* **do**
  initialise the loss to 0;
  Generate $M$ predictions using the trained model to obtain the clean values $\widehat{Y}$, the Wiener process $W$ associated with it, the underlying process $X_t$, and the time-steps $\Delta t$;
  Obtain the values for $\text{xVA}_0$ and $Z_0$ by passing ($t_0$,$X_0$) through the xVA model and automatic differentiation of its output;
  **for** *number of time steps* $n = 1, ...., N-1$ **do**
    With $\Delta t_n$, $\widehat{Y}_n$, $W_n$, $\text{xVA}_n$ and $Z_n$ obtain the true value $\widehat{\text{xVA}}_{n+1}$ using the following equation as defined in 4.21:

$$\widehat{\text{xVA}}_{n+1} = \text{xVA}_n - a(t_n, \widehat{Y}_n, \text{xVA}_n)\Delta t + (Z_n)^\top \Delta W_n$$

    Obtain the model's predicted value $\text{xVA}_{n+1}$ and $Z_{n+1}$ by passing $(t_n, X_n)$ as input to the xVA model and automatic differentiation of its output;
    Add $\sum_{m=1}^{M} \left|\text{xVA}_{n+1}^m - \widehat{\text{xVA}}_{n+1}^m\right|^2$ to the loss;
  Add $\sum_{m=1}^{M} \left|\text{xVA}_N^m - g(\widehat{Y}_N^m)\right|^2 + \sum_{m=1}^{M} \left|Z_N^m - g'(\widehat{Y}_N^m)\right|^2$ to the loss for the terminal time $T$;
  Perform back-propagation to compute the gradients across the neural network;
  Minimise the loss by using a suitable optimiser;
  Update the weights;

Gnoatto et al. [34] uses a similar risk framework to define the the xVA calculations for forwards, call options, and basket call options and proposed an algorithm using the PDE solver by Han et al. [27] (refer to section 3.1.9 for the advantages of our PDE solver over that of Han et al.). Therefore, we conduct experiments in the following chapter to compare our outputs with that of Gnoatto et al. [34] and the exact values computed using Monte Carlo simulations.

# Chapter 5

# Numerical Results

In this chapter, we train our model defined in chapter 3 using the Nais-net architecture and a non-geometric multi-level Monte Carlo technique to obtain the valuation of European call options, forward contracts, baskets of forward contracts, and baskets of European call options. We use algorithms 3 and 4 to calculate the fair value of these contracts and portfolios by estimating the xVA equation as defined in chapter 4. We compare the results of our xVA calculations with the work by Gnoatto et al. [34], which uses a different neural network setup as proposed by Han et al. [27] and a different training algorithm to estimate these values. We also use Monte Carlo simulations to estimate the exact xVA values for comparison with our results. All the experiments detailed in this report are available in the GitHub repository: https://github.com/Aadhithya-06/Final-Year-Project/tree/master. For our experiments we use the NVIDIA GeForce RTX 4080 GPU.

## 5.1 Experimental Setup

Consider the process used to represent the price of an asset $X_t$ defined by the following stochastic differential equation:

$$\mathrm{d}X_t = rX_t\mathrm{d}t + \sigma X_t\mathrm{d}W_t, \tag{5.1}$$

where $r$ represents the risk free rate which is assumed to be constant, $\sigma$ represents the volatility of the asset price and $W_t$ is a Wiener process.

We define a European style contingent claim on the asset which may be a call option or forward contract payoff which we can define as the clean value of the contract follows:

$$\widehat{Y}_t = \mathbb{E}\left[e^{-r(T-t)}g(X_T)\right], \tag{5.2}$$

where $T$ is the terminal time and $g(X)$ is the terminal payoff. $Y_t$ can be expressed as the following backward stochastic differential equation:

$$\begin{aligned} \mathrm{d}\widehat{Y}_t &= r\widehat{Y}_t\mathrm{d}t + \widehat{Z}_t\mathrm{d}W_t, \\ \widehat{Y}_T &= g(X_T), \end{aligned} \tag{5.3}$$

where $\widehat{Z}_t$ is the derivative of $\widehat{Y}_t$ with respect to the underlying price $X_t$. This term is the first order sensitivity of the price of the underlying. For options contracts this is called delta ($\delta$) and is used by the bank as a hedging strategy wherein they open a position of $-1 \times \delta$ of the underlying when they sell an options contract to the client. This quantity is also approximated by our neural network via the automatic differentiation functionality in PyTorch.

We also define the terms Discounted Expected Positive Exposure (DEPE) and Discounted Expected Negative Exposure (DENE) of the European style contracts as follows:

$$\begin{aligned} \text{DEPE}(t) &= \mathbb{E}\left[e^{-r(t-T)}\left(\widehat{Y}_s\right)^+\right], \\ \text{DENE}(t) &= -\mathbb{E}\left[e^{-r(t-T)}\left(\widehat{Y}_s\right)^-\right]. \end{aligned} \tag{5.4}$$

We have defined it in the same way as Gnoatto et al. [34] so that it would aid us in our comparisons. Since this model only approximates $\widehat{Y}_0$ and the xVA measures at the initial time with the subsequent values obtained by time-marching, we compare our model's ability to approximate these values at $t = 0$ with this model. We also provide figures and relative error of our model's predictions with the exact solution at the initial time as the xVA PDE does not have an exact analytical solution in high dimensions in the subsequent sections.

## 5.2 Forward Contracts

We have derived the pricing for a European style Forward contract in section 2.4.5. We can define the terminal condition of this contract as follows:

$$g(X_T) = X_T - K, \tag{5.5}$$

where $K$ is the strike price of the forward contract and we set this to be the initial price i.e, $K = X_0 = \xi$. We have the exact clean value of this contract $\widehat{Y}_t$ as:

$$\widehat{Y}_t = X_t - Ke^{-r(T-t)} \tag{5.6}$$

In section 2.4.5 we defined the price of a forward contract in terms of a call option and a put option using the put-call parity. The expected positive exposure and the expected negative exposure are represented by the payoffs of the call option and the put option with strike $K$ on the underlying. Therefore we have

$$\begin{aligned} \text{DEPE}(t) &= X_t N(d_1) - Ke^{-r(T-t)}N(d_2), \\ \text{DENE}(t) &= X_t N(-d_1) - Ke^{-r(T-t)}N(-d_2), \end{aligned} \tag{5.7}$$

with

$$\begin{aligned} N(x) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{z^2}{2}}dz, \\ d_1 &= \frac{ln(X_t/K) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}, \\ d_2 &= \frac{ln(X_t/K) + (r - \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}. \end{aligned} \tag{5.8}$$

### 5.2.1 Clean Values

For our first experiment, we use the algorithm 1 to train a model with the Nais-Net architecture and non-geometric multi-level Monte Carlo technique to obtain the clean price valuation of a single forward contract. We use the risk-free rate $r = 0$, the volatility $\sigma = 0.25$ and a strike $K = 100$. We also set the initial price of the underlying asset to be $X_0 = \xi = 100$. We set the terminal time to be $T = 1$. We train the model for a maximum of 50 discretisation steps and use the ability of our model to generalise at finer discretisation levels to predict the solution for 200 discretisation time-steps after training. See figure 5.1 for the loss and predictions of our model and figure 5.2 for a comparison of the Expected positive and negative exposures between our model and that of Gnoatto et al. for this case. The L2 error across the 200 discretised time-steps is $4.4 \times 10^{-4}$.

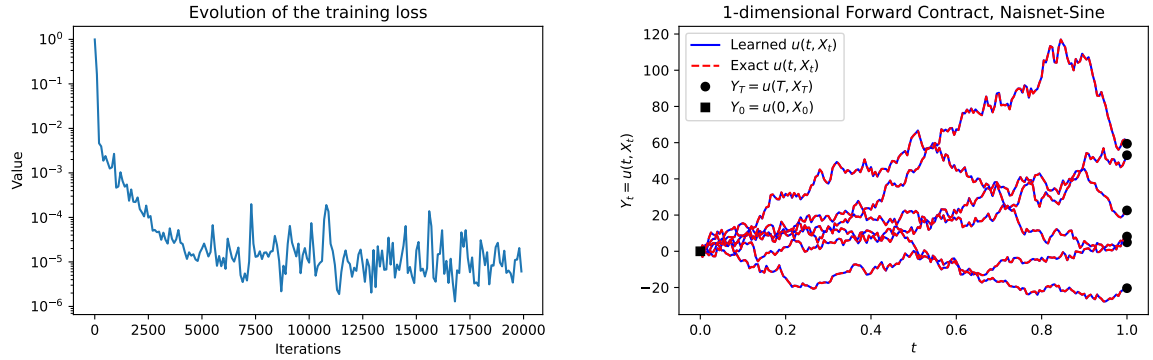


**Figure 5.1:** This figure represents the Loss (left) and the approximation by a neural network with 2 hidden layers and 21 neurons in each layer of $u(t, x)$ across 5 realisations of $X_t$ over 200 time-steps on training the Nais-Net model with sinusoidal activation in 50 time-steps for 20k iterations with a training time of 403.76s using the non-geometric MLMC scheme as detailed in section 3.4.3 for a forward contract.

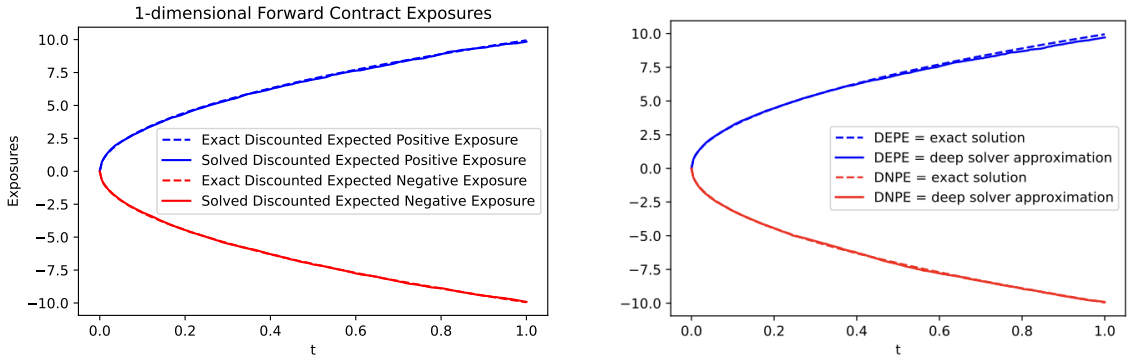


**Figure 5.2:** This figure represents the DEPE and DENE predicted by our model (left) and that of Gnoatto et al. [34] (right).

From figure 5.2 we notice that our model approximates the exposures slightly better than that of Gnoatto et al., especially the DEPE towards the terminal time. Our model has a maximum absolute error of 0.072 for the DEPE and DENE with the exact values and a standard deviation of 0.0176 whereas the model by Gnoatto et

al. has a maximum absolute error of 0.12 for the DEPE and DENE with a standard deviation of 0.3647.

### 5.2.2 FVA calculations for Forward Contract Portfolios

In this section, we use algorithm 4 to compare the initial FVA values. For the purposes of this experiment we use $r = 0.02$. We assume that there is no collateral, $C = 0$ and there is no default risk $\lambda^C = \lambda^B = 0$. We also assume the recovery rates $R^C = R^B = 1$. This allows us to obtain only the FVA value from equation 4.18. Suppose we have a rate $r^{f,l} = r^{f,b} = r^f$, then the exact FVA value at time $t$ is the difference between the exact value at time $t$ calculated using the discounting rate $r$ and the exact value calculated using the discounting rate $r^f$. The results of our algorithm and comparison with the exact FVA values and the model provided by Gnoatto et al. for 1D Forward contract with $r = 0.02$ and $r^f = 0.04,\ 0.08$ and $0.12$ is provided in table 5.1.

| $r^f$ | 0.04 | 0.08 | 0.12 |
|---|---|---|---|
| Gnoatto et al. | 0.03950 | 0.11550 | 0.18970 |
| Exact Value | 0.03920 | 0.11531 | 0.18843 |
| Proposed Model | 0.03923 | 0.11542 | 0.18815 |

**Table 5.1:** Comparison of our Proposed model, model by Gnoatto et al. and Exact values of $\text{FVA}_0$ for a Forward contract with strike $K = 100$ and initial underlying price $X_0 = 100$ across different $r^f$ rates. We train our Nais-Net model with Sine activation, 2 hidden layers and 256 neurons each for 3000 iterations using algorithm 4. We pass a trained model to compute the clean values of the forward contract with $r = 0.02$ at each time step.

We also validate the performance of our algorithm for high dimensional data by considering portfolios of forward contracts with each dimension of the input representing one forward contract. The results and comparisons are provided in table 5.2.

| $d$ | Exact Value | Proposed Model | Error (%) | Gnoatto et al. | Error (%) | Training Time (s) |
|---|---|---|---|---|---|---|
| 1 | 0.03920 | 0.03923 | 0.0765 | 0.03950 | 0.7651 | 435 |
| 10 | 0.39209 | 0.39205 | 0.0102 | 0.39199 | 0.0255 | 590 |
| 25 | 0.98023 | 0.98047 | 0.0244 | 0.97568 | 0.4692 | 736 |
| 50 | 1.9605 | 1.9607 | 0.0102 | 1.9439 | 0.8467 | 948 |
| 100 | 3.9209 | 3.9204 | 0.0128 | 3.8976 | 0.5942 | 1362 |
| 150 | 5.8813 | 5.8801 | 0.0204 | 5.8603 | 0.3570 | 1521 |
| 200 | 7.8418 | 7.8371 | 0.0599 | 7.8159 | 0.3290 | 1783 |

**Table 5.2:** Comparison of our Proposed model, model by Gnoatto et al. and Exact values of $\text{FVA}_0$ for Forward Contracts with $K = 100$ and $X_0 = 100$ across different portfolios.

We increase the complexity of the model architecture with an increase in the number of dimensions to calculate the clean and the FVA values, with 2 hidden layers of size

21, 30, 45, 70, 256 and 256 to calculate the clean values for dimensions 1, 10, 25, 50, 100 and 150 respectively and 4 hidden layers of size 256 for the 200 dimension case. For the FVA calculations we use the same architecture as the 1D case for dimensions 1, 10, 25 and 50 respectively and 4 hidden layers with 256 neurons for the 100, 150 and 200 dimensional cases. From table 5.2 we see that our model generalises well to obtain an error of less than 0.1% on training for between 2000 and 5000 iterations with learning rates between $1e^{-5}$ and $1e^{-3}$ for the portfolios of forward contracts in comparison with the model proposed by Gnoatto et al. where the error is around the 1% level.

### 5.2.3 FVA calculations with Collateral

For our calculations we propose a collateral arrangement such that collateral is exchanged between each part involved in the transaction at every point in time when the collateral is above or below a certain threshold. We define a collateral account that follows the following process for the fully collateralised case:

$$C_t = \left[(\widehat{Y}_t - c)^+ - (\widehat{Y}_t + c)^-\right] \tag{5.9}$$

where, $c$ is the threshold. We perform experiments with full collateralisation across portfolios of forward contracts to obtain the results in table 5.3.

| $d$ | Full Collateralisation | No Collateralisation |
|---|---|---|
| 1 | 4.5449 $\times 10^{-5}$ | 0.03923 |
| 10 | -4.3958 $\times 10^{-5}$ | 0.39205 |
| 25 | 5.2526 $\times 10^{-5}$ | 0.98047 |
| 50 | -1.6764 $\times 10^{-5}$ | 1.9607 |
| 100 | -1.0244 $\times 10^{-5}$ | 3.9204 |
| 150 | -1.0710 $\times 10^{-6}$ | 5.8813 |

**Table 5.3:** FVA values with collateral threshold $c = 10$.

From table 5.3 we see that the FVA value is negligible when the transaction is fully collateralised at each time-step. This is in line with our expectation as the presence of collateral leads to a decrease in credit risk since the collateral can be used to recover losses from the transaction in the event of default of either party involved. Furthermore, funding costs decrease as due to posting collateral, the parties can borrow at a lower rate to hedge their exposure.

## 5.3 European Call Option

We have derived the pricing for European Call Options in section 2.4.3. Here, we define the terminal condition as follows:

$$g(X_T) = (X_T - K)^+ = \max(X_T - K, 0) \tag{5.10}$$

where we set the strike price K to be the initial price of the underlying asset such that $K = X_0 = \xi$. The exact clean values of the European call option contract as defined in section 2.4.3 is:

$$\widehat{Y}_t = X_t N(d_1) - K e^{-r(T-t)} N(d_2) \tag{5.11}$$

with $d1$, $d2$ and $N$ having the same definitions as in equation 5.8. The expected positive and negative exposures are therefore:

$$\begin{aligned} \text{DEPE}(t) &= \mathbb{E}\left[e^{-r(t-T)}\left(\widehat{Y}_t\right)^+\right] \\ &= \mathbb{E}\left[e^{-r(t-T)}\left(X_s - K\right)^+\right] = \widehat{Y}_t \\ \text{DENE}(t) &= -\mathbb{E}\left[e^{-r(t-T)}\left(\widehat{Y}_s\right)^-\right] = 0 \end{aligned} \tag{5.12}$$

### 5.3.1 Clean Values

We use the algorithm 1 to train a model with the Nais-Net architecture and non-geometric multi-level Monte Carlo technique with 4 hidden layers of size 256 each to obtain the clean price valuation of a call option . We use the risk-free rate $r = 0.01$, the volatility $\sigma = 0.25$ and a strike $K = 100$. We also set the initial price of the underlying asset to be $X_0 = \xi = 100$. We set the terminal time to be $T = 1$. We train the model for a maximum of 50 discretisation steps and use the ability of our model to generalise at finer discretisation levels to predict the solution for 200 discretisation time-steps after training. See figure 5.3 for the loss and predictions of our model and figure 5.4 for a comparison of the expected positive and negative exposures between our model and that of Gnoatto et al. for this case.

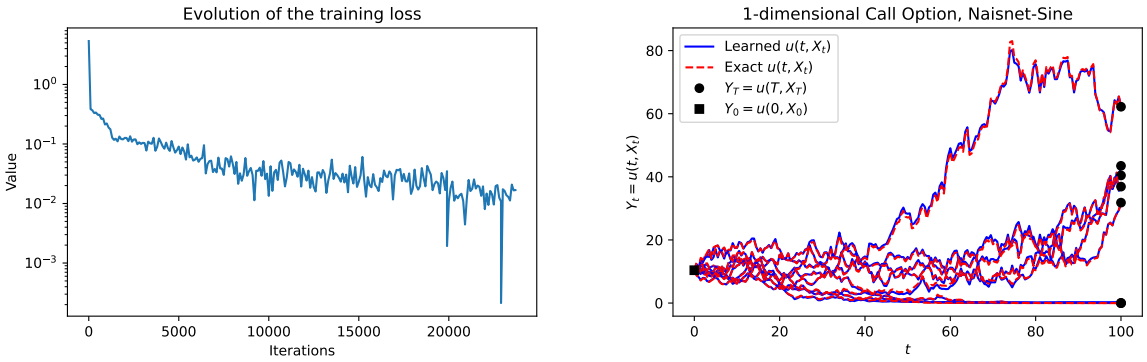


**Figure 5.3:** This figure represents the Loss (left) and the approximation by a neural network with 4 hidden layers and 256 neurons in each layer of $u(t, x)$ across 7 realisations of $X_t$ over 200 time-steps on training the Nais-Net model with sinusoidal activation in 50 time-steps for 24k iterations using the non-geometric MLMC scheme as detailed in section 3.4.3 for a call option. The average L2 error across all predictions is $5.33 \times 10^{-3}$.

From figure 5.4 we notice that our model approximates the exposures slightly better

than that of Gnoatto et al., especially the DENE towards the terminal time. Our model has a maximum absolute error of 0.1147 for the DEPE and the DENE with the exact values and a standard deviation of 0.0328 for the DEPE whereas the model by Gnoatto et al. has a maximum absolute error of 0.17 for the DEPE and DENE. Furthermore our model perfectly predicts the DENE case with an error of 0.

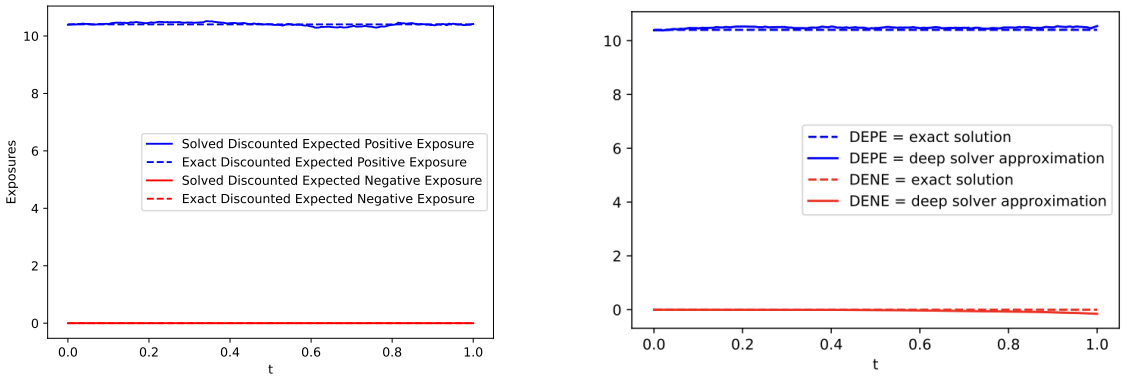


**Figure 5.4:** This figure represents the DEPE and DENE predicted by our model (left) and that of Gnoatto et al. [34] (right).

## 5.4 European Call Option Basket

We define a European Call Option Basket with $D$ underlying assets such that $X_t$ as defined in equation 5.1 is now of the form $X_t^d$ where $d$ refers to the asset corresponding to the $d^{th}$ with $d = 1...D$. We define the terminal condition as follows:

$$g(X_T) = (\sum_{d=1}^{D} X_T^d - d \cdot K)^+ = \max(\sum_{d=1}^{D} X_T^d - d \cdot K, 0) \tag{5.13}$$

where we set the strike price $K$ to be the initial price of the underlying asset such that $K = X_0^d = \xi^d$. We set this value to be 100 for our experiments. The expected positive and negative exposures are given by:

$$\begin{aligned} \text{DEPE}(t) &= \mathbb{E}\left[e^{-r(t-T)}\left(\widehat{Y}_t\right)^+\right] \\ &= \mathbb{E}\left[e^{-r(t-T)}\left(\sum_{d=1}^{D} X_T^d - d \cdot K\right)^+\right] = \widehat{Y}_t \\ \text{DENE}(t) &= -\mathbb{E}\left[e^{-r(t-T)}\left(\widehat{Y}_s\right)^-\right] = 0 \end{aligned} \tag{5.14}$$

### 5.4.1 Clean Values

We use the algorithm 1 to train a model with the Nais-Net architecture and non-geometric multi-level Monte Carlo technique with 4 hidden layers of size 256 each

to obtain the clean price valuation of a 100 dimensional call option basket . We use the risk-free rate $r = 0.01$, the volatility $\sigma = 0.25$ and a strike $K = 100$. We also set the initial price of the underlying asset to be $X_0 = \xi = 100$. We set the terminal time to be $T = 1$. We train the model for a maximum of 50 discretisation steps and use the ability of our model to generalise at finer discretisation levels to predict the solution for 200 discretisation time-steps after training. See figure 5.5 for the loss and predictions of our model and figure 5.6 for a comparison of the expected positive and negative exposures between our model and that of Gnoatto et al. for this case.

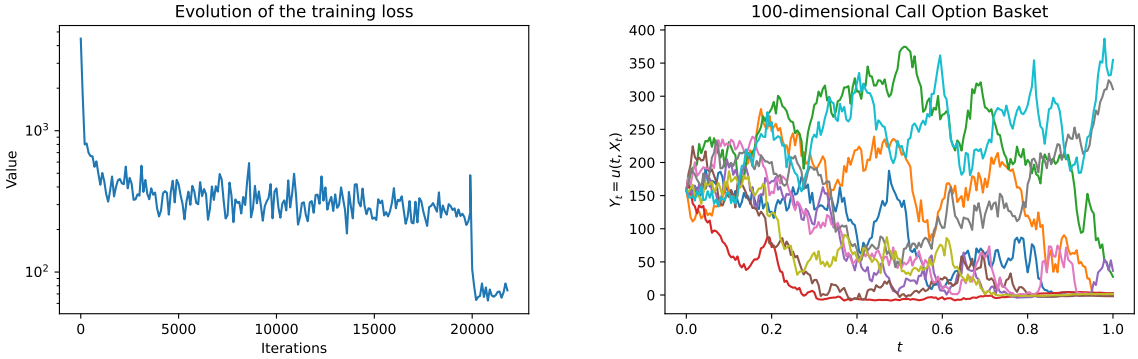


**Figure 5.5:** This figure represents the Loss (left) and the approximation by a neural network with 4 hidden layers and 256 neurons in each layer of $u(t, x)$ across 10 realisations of $X_t$ over 200 time-steps on training the Nais-Net model with sinusoidal activation in 50 time-steps for 21k iterations with a training time of 1547.35s using the non-geometric MLMC scheme as detailed in section 3.4.3 for a 100 asset Call Option Basket.

From figure 5.6 we notice that our model approximates the exposures slightly better than that of Gnoatto et al., especially the DEPE and DENE towards the terminal time similar to the case of forward contracts and call options in a low dimensional setting as in sections 5.2 and 5.4 respectively. Our model has a maximum absolute error of 1.372 for the DEPE and 0.410 for the DENE with the exact values and a standard deviation of 0.29 and 0.14 for the DEPE and DENE cases respectively whereas the model by Gnoatto et al. has a maximum absolute error of 3.25 and 2.98 for the DEPE and DENE respectively. Therefore our model has an error $\approx 0.8\%$ whereas the model proposed by Gnoatto et al. has an error of $\approx 2\%$.

### 5.4.2 CVA and DVA Calculations

We validate our algorithms 3 and 4 for a 100 dimensional call basket portfolio to calculate the CVA and DVA metrics by setting a unique stream of funding with a fixed rate $r = 0.01$ and no collaterlisation. This ensures that the FVA and ColVA terms are 0. We set the intensities of default of the bank $\lambda^B = 0.01$ and the counterparty $\lambda^C = 0.10$ to be constant over time. We also define the recovery rates of the bank $R^B = 0.4$ and the counterparty $R^C = 0.3$. We first compute the clean values as defined in section 5.4.1. Using these values, we use algorithm 3 to obtain the value of $\text{XVA}_0 = \text{DVA}_0 - \text{CVA}_0$ to obtain the value 0.8995 by averaging across 8192

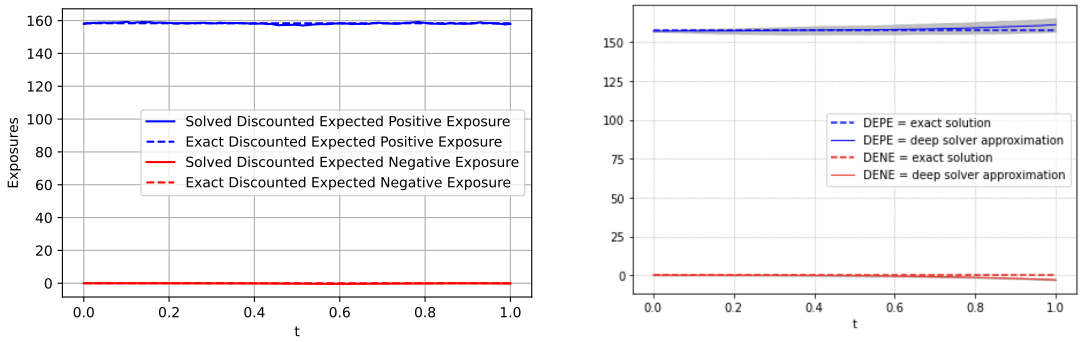


**Figure 5.6:** This figure represents the DEPE and DENE predicted by our model (left) and that of Gnoatto et al. [34] (right) for a Basket of Call Options.

simulations. The exact value is obtained by a monte carlo simulation of portfolio values over 100k simulations to obtain the value of 0.8999 for $\text{DVA}_0 - \text{CVA}_0$. So our algorithm produces an absolute error of $0.05\%$. We also validate the performance of algorithm 4 using the parameters mentioned above as we can obtain not only the initial value but also subsequent values of the xVA across time. So we train a Nais-Net model with 4 hidden layers of size 256 across 3000 iterations to obtain the initial xVA value of 0.8984. The model proposed by Gnoatto et al. [34] trained with the same parameters yields an initial value of 0.8952 indicating an error of $\approx 0.52\%$ whereas our model has an error of $\approx 0.20\%$. Therefore, our model using algorithm 4 has a similar error level as the model proposed by Gnoatto et al. whereas algorithm 3 allows a more accurate pricing for the initial value for the CVA and DVA terms.

### 5.4.3 FVA Calculations

We use a similar set-up as in section 5.2.2 to validate our algorithm 4 by approximating the FVA value for a 100 dimensional call option basket. We set the risk free rate $r = 0.01$ and assume that there is no collateral, $C = 0$ and there is no default risk $\lambda^C = \lambda^B = 0$. We also assume the recovery rates $R^C = R^B = 1$. This allows us to obtain only the FVA value from equation 4.18. Suppose we have a rate $r^{f,l} = r^{f,b} = r^f = 0.04$, then the exact FVA value at time $t$ is the difference between the exact value at time $t$ calculated using the discounting rate $r$ and the exact value calculated using the discounting rate $r^f$. So we train a Nais-Net model with 4 hidden layers of size 256 across 4000 iterations to obtain the initial FVA value of 4.6698 whereas the exact value calculated is 4.6693. Therefore our model has an error of $\approx 0.01\%$. We also trained a new instance of this model 10 times using the same parameters and noticed that there was a standard deviation of $\approx 0.05\%$. Therefore, the error is approximately at the $0.1\%$ level similar to the FVA calculations for the portfolios of forward contracts.

# Chapter 6

# Conclusion

## 6.1 Achievements

The first goal of the project was to replicate the work of Raissi [25] and Batuhan Guler, Alexis Laignelet, and Panos Parpas [30]. We successfully replicated their work and conducted further experiments on the Black-Scholes equation with a quadratic payoff using the ResNet-based Nais-Net architecture. We identified that the projection step of the Nais-Net architecture, which ensured the stability of the results by constraining the weights of the neural network, also enabled faster convergence to a similar level of loss in 25k iterations that the feed-forward model achieved in 100k iterations of training due to this feature, thereby achieving the same error levels as Raissi in a lesser number of training iterations. Our next aim was to further reduce computational complexity while maintaining the same error levels. Subsequently, we analysed methods to decrease the training time. This analysis led us to explore discretisation techniques such as geometric and non-geometric Multi-Level Monte Carlo methods, which significantly reduced training time—by approximately 85%. As a result, we could produce a model that approximates the solution with the same level of accuracy in 25k iterations, compared to the initial model which required 100k iterations. We also conducted experiments to understand the model's ability to generalise to finer discretisation levels in section 3.4.4 and explored the bias-variance trade-off to identify the optimal architecture and the number of realisations needed for training, the results of which are detailed in section 3.5. Since the price movements of most assets in portfolios used to price financial contracts are correlated with each other, we introduced a methodology to incorporate correlations to the underlying assets and achieved a similar level of accuracy as in the uncorrelated case in section 3.6.

Then we laid out a framework to represent the xVA equations as a backward stochastic differential equation and proposed a recursive algorithm using our neural network architecture, which uses the values predicted by the model defined in chapter 3 to approximate a solution to this equation. We compared the results of our approach to calculate the Funding Valuation, Debt Valuation, and Credit Valuation adjustments for portfolios of forward contracts and call options, achieving an error of approximately 0.1% compared to another state-of-the-art approach by Gnoatto et al.

[34], which achieves an error of approximately 1% with respect to the exact initial values calculated using a Monte Carlo simulation. Furthermore, we also simulated Funding Valuation adjustments in the presence of collateral to validate our expectation that the inclusion of collateral reduces credit risk, thereby making the funding valuation adjustment negligible. Therefore, we have developed a framework to calculate an accurate risk-adjusted valuation of low and high-dimensional portfolios of forward contracts and call options in a reasonable amount of time.

## 6.2 Future Work

In this project, we focused on the valuation and xVA calculations of financial contracts with European-style payoffs that occur only at the terminal time. Therefore, a natural extension would be to develop a similar framework for the valuation of financial contracts with American payoffs that can be exercised at any time before expiry. We have successfully formulated an optimisation problem for the xVA and Black-Scholes backward stochastic differential equation by stochastic differential equation discretisation and path sampling using the control variate of this process. This control variate ($Z_t$ as described in equation 3.1) in the Black-Scholes equation represents the hedging ratio, which is important for the bank to hedge their exposures to the financial contracts that they sell to clients. Currently, we only minimise the error of the control variate generated by automatic differentiation of the solution of our model at the terminal time in the loss function, as in equation 3.4. Another extension could be to modify our algorithm and also test the approximation of this value at each time step and compare it to the exact values calculated by other techniques. Additionally, we can follow the conclusions drawn from our analysis in section 3.3 and include higher derivative terms to obtain not only the hedge ratio but also other sensitivities, which are important for banks to hedge their exposure. Furthermore, alternative optimisation techniques could also be explored to improve the model's performance. Before banks can use the proposed algorithm, it also needs to be tested in a real-world setting. Therefore, an analysis of our algorithm's performance by training our models on real price and risk metrics data gathered from banks would further validate our approach.

Approximating the solution to forward-backward stochastic differential equations in high dimensions using neural networks is also a powerful tool in industries outside finance. For instance, in the field of energy, neural networks are used to solve the Hamilton-Jacobi-Bellman equations to optimise the operation and management of energy storage systems, balancing supply and demand efficiently. In environmental science, these techniques help model complex systems like climate dynamics by approximating solutions to the Stochastic Navier-Stokes equations. Additionally, in the realm of neuroscience, neural networks assist in solving the Hodgkin-Huxley model, which describes the electrical characteristics of neurons, aiding in the study of brain activity and the development of neurological treatments. Therefore, there is potential for our project to solve these problems impacting various industries if future work is done to test our approach on these equations.

# Bibliography


[1] Darrell Duffie and Ming Huang. Swap rates and credit quality. *Journal of Finance*, 51(3):921–49, 1996. pages 1

[2] T. Bielecki and M. Rutkowski. *Credit Risk: Modeling, Valuation and Hedging*. Springer Finance Berlin, 2003. pages 1

[3] Damiano Brigo and Agostino Capponi. Bilateral counterparty risk valuation with stochastic dynamical models and application to credit default swaps, 2009. pages 1

[4] DAMIANO BRIGO, ANDREA PALLAVICINI, and VASILEIOS PAPATHEODOROU. Arbitrage-free valuation of bilateral counterparty risk for interest-rate products: Impact of volatilities and correlations. *International Journal of Theoretical and Applied Finance*, 14(06):773–802, 2011. pages 1

[5] Damiano Brigo and Andrea Pallavicini. Ccp cleared or bilateral csa trades with initial/variation margins under credit, funding and wrong-way risks: A unified valuation approach, 2014. pages 1

[6] Andrea Pallavicini, Daniele Perini, and Damiano Brigo. Funding valuation adjustment: a consistent framework including cva, dva, collateral,netting rules and re-hypothecation, 2011. pages 1, 36

[7] Tomasz R. Bielecki and Marek Rutkowski. Valuation and hedging of contracts with funding costs and collateralization, 2014. pages 1

[8] Francesca Biagini, Alessandro Gnoatto, and Immacolata Oliva. A unified approach to xva with csa discounting and initial margin, 2021. pages 1, 36, 41

[9] Maxim Bichuch, Agostino Capponi, and Stephan Sturm. Robust xva, 2020. pages 1

[10] Maxim Bichuch, Agostino Capponi, and Stephan Sturm. Arbitrage-free xva. *Mathematical Finance*, 28(2):582–620, April 2017. pages 1

[11] https://www.bis.org/publ/bcbs189.pdf. pages 1

[12] https://www.icmagroup.org/market-practice-and-regulatory-policy/secondary-markets/secondary-markets-regulation/fundamental-review-of-the-trading-book-frtb/. pages 1

[13] MARK JOSHI and OH KANG KWON. Least squares monte carlo credit value adjustment with small and unidirectional bias. *International Journal of Theoretical and Applied Finance*, 19(08):1650048, 2016. pages 1

[14] Shashi Jain Patrik Karlsson and Cornelis W. Oosterlee. Counterparty credit exposures for interest rate derivatives using the stochastic grid bundling method. *Applied Mathematical Finance*, 23(3):175–196, 2016. pages 1

[15] Claudio Albanese, Simone Caenazzo, and Stephane Crepey. Credit, funding, margin, and capital valuation adjustments for bilateral portfolios. *Probability, Uncertainty and Quantitative Risk*, 2(0):7, 2017. pages 1

[16] Jean-Michel Bismut. Conjugate convex functions in optimal stochastic control. *Journal of Mathematical Analysis and Applications*, 44(2):384–404, 1973. pages 2

[17] E. Pardoux and S.G. Peng. Adapted solution of a backward stochastic differential equation. *Systems and Control Letters*, 14(1):55–61, 1990. pages 2

[18] N. El Karoui, S. Peng, and M. C. Quenez. Backward stochastic differential equations in finance. *Mathematical Finance,* 7(1):1–71, 1997. pages 2

[19] Étienne Pardoux and Shanjian Tang. Forward-backward stochastic differential equations and quasilinear parabolic pdes. *Probability Theory and Related Fields*, 114:123–150, 1999. pages 2, 16

[20] https://medium.com/free-code-camp/the-curse-of-dimensionality-how-we-can-save-big-data-from-itself-d9fa0f872335. pages 2

[21] Emmanuel Gobet, Jean-Philippe Lemor, and Xavier Warin. A regression-based monte carlo method to solve backward stochastic differential equations. *The Annals of Applied Probability*, 15(3), August 2005. pages 2

[22] Emmanuel Gobet and Plamen Turkedjiev. Approximation of backward stochastic differential equations using malliavin weights and least-squares regression. *Bernoulli*, 22(1), February 2016. pages 2

[23] Bruno Bouchard and Nizar Touzi. Discrete-time approximation and monte-carlo simulation of backward stochastic differential equations. *Stochastic Processes and their Applications*, 111(2):175–206, 2004. pages 2, 7

[24] T. Uchiyama and N. Sonehara. Solving inverse problems in nonlinear pdes by recurrent neural networks. pages 99–102 vol.1, 1993. pages 2

[25] Maziar Raissi. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations, 2018. pages 2, 3, 8, 16, 20, 21, 22, 25, 54

[26] Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, November 2017. pages 2, 22, 27, 32

[27] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018. pages 2, 22, 27, 32, 44, 45

[28] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, December 2018. pages 2, 22, 27

[29] Christian Beck, Sebastian Becker, Philipp Grohs, Nor Jaafari, and Arnulf Jentzen. Solving the kolmogorov pde by means of deep learning. *Journal of Scientific Computing*, 88(3), July 2021. pages 2, 22, 27, 32

[30] Batuhan Güler, Alexis Laignelet, and Panos Parpas. Towards robust and stable deep learning algorithms for forward backward stochastic differential equations, 2019. pages 2, 3, 8, 16, 22, 24, 28, 54

[31] Batuhan Güler, Alexis Laignelet, and Panos Parpas. https://github.com/batuhanguler/deep-bsde-solver. pages 2

[32] Maziar Raissi. https://github.com/maziarraissi/fbsnns. pages 2, 21

[33] Marco Ciccone, Marco Gallieri, Jonathan Masci, Christian Osendorfer, and Faustino Gomez. Nais-net: Stable deep networks from non-autonomous differential equations, 2021. pages 3, 23, 24, 25, 27

[34] Alessandro Gnoatto, Athena Picarelli, and Christoph Reisinger. Deep xva solver – a neural network based counterparty credit risk management framework, 2022. pages 3, 36, 44, 45, 46, 47, 51, 53, 55

[35] Oliver Ibe. *Fundamentals of Applied Probability and Random Processes*. CA: Academic Press., 2014. pages 4

[36] Emmanuel Gobet. *Monte-Carlo Methods and Stochastic Processes: From Linear to Non-Linear*. Chapman and Hall/CRC,, 2016. pages 8

[37] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973. pages 9

[38] John C Hull. *Options, Futures, and Other Derivatives*. Pearson, 2018. pages 9

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. pages 17

[40] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey, 2018. pages 17

[41] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. pages 21

[42] Michael B. Giles. Multilevel monte carlo path simulation. *Operations Research*, 56(3):607–617, 2008. pages 23, 28, 29

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. pages 23, 24

[44] Linan Zhang and Hayden Schaeffer. Forward stability of resnet and its variants, 2018. pages 24

[45] Michael B. Giles. Multilevel monte carlo methods, 2018. pages 28, 29

[46] Damiano Brigo, Marco Francischello, and Andrea Pallavicini. Nonlinear valuation under credit, funding, and margins: Existence, uniqueness, invariance, and disentanglement. *European Journal of Operational Research*, 274(2):788–805, 2019. pages 36