A Comparison of Numerical Methods for Solving Backward Stochastic Differential Equations

A Comparison of Numerical Methods for Solving Backward Stochastic Differential Equations

Andrew Duquette

Acknowledgements

Thanks to Dr. Traian Pirvu for being supportive throughout my Master's, especially considering the difficult time we were operating during. He was always helpful when we talked and able to point me to where I needed to go. I'd also like to thank Dr. Shui Feng both for his mentorship during my undergraduate career and serving on the committee. He always managed to show me that there was more that I knew that I didn't even realize. In addition thanks to Dr. Bartek Proyas who was willing to step up to be on my supervisory committee on short notice and help me see this through to the finish.

I would also like to thank my family for the support over the course of writing this thesis. They always gave me support and cheer when they could. I couldn't have done this without you all.

Abstract

We study several different numerical methods for solving Backwards Stochastic Differential Equations and Partial Differential Equations. The main methods reviewed are a least-squares Monte-Carlo method, and a method utilizing Artificial Neural Networks: the Deep BSDE method. We implement both algorithms and compare their performance solving a BSDE to find the fair price of a financial option in an incomplete market. We find that the Deep BSDE method can provide similar approximation efficiency to the Monte-Carlo algorithm with a limited number of partition points, but outperforms it as the number of partition points increases.

In addition, we examine another technique that utilizes artificial neural networks to solve Partial Differential Equations directly, the Deep PDE Method. This method provides solution dynamics across the entire domain of the problem, as opposed to the Deep BSDE method which provides solution dynamics only for a small subset of points in the domain. This method is computationally intensive to train, and also requires bounded domains and boundary conditions and bounded domains, unlike the Deep BSDE algorithm where the domain is unbounded. It remains a future research question to see whether the Deep BSDE method could be modified to work with bounded domains or if the Deep PDE Method could somehow be expanded to work on unbounded domains.

Contents

Introduction

In the area of mathematical finance and stochastic control, it is desirable to understand the movement of a particular process which has both deterministic elements as well as stochastic noise. An example of this is how to model the value of a stock. Given there are no large shocks to the market, we may consider the price to be increasing in a deterministic fashion, but also containing random noise. In a general form, we can use the following equation to model this sort of process:

$$dX_t = b(t, X_t)dt + \sigma(t, X_t)dB_t$$

where B_t is taken to be a standard Brownian motion over the given probability space. The solution $\{X_t\}_{t\in[0,T]}$ to this equation is a stochastic process which takes values in $\mathbb{R}^d$. Stochastic equations are used to model many different phenomena where the underlying process is inherently uncertain. A backward stochastic differential equation (BSDE) is defined using an underlying stochastic process X_t as above. A simple BSDE may be of the form

$$dY_t = g(t, Y_t)dt + Z_t dB_t$$
$$Y_T = h(X_T),$$

for given functions g and h. The solution to the above BSDE for given functions is given by a unique tuple (Y, Z) of processes adapted to filtration generated by the Brownian motion B. The process Z is required as part of the solution to force the solution to be a process *adapted* to the underlying filtration generated by the Brownian motion B_t.[1]

BSDEs are similarly used to model financial options and derivatives in mathematical finance, but also have an interesting application in finding solutions to certain types of PDEs. In particular, there is a strong relationship between solutions of SDEs and BSDEs and various classes of Partial Differential Equations (PDEs). This relationship has been exploited for many years to find numerical solutions to PDEs by

[1] These definitions can be found in the appendix if required.

first solving an associated BSDE (given explicitly by the Feynman-Kac relationship), and then using that solution to find a solution to the corresponding PDE. Generally, this probabilistic approach was used to solve PDEs, and the extension from the traditional Feynman-Kac methods to a larger class of PDEs involved extending the original link, which only utilised SDEs to represent solutions to elliptic PDEs, to classes of parabolic PDEs of both quasi-linear and semi-linear types, the solutions of which could be represented using solutions to BSDEs of appropriate form.

For many years, BSDEs were solved numerically using various Monte Carlo numerical methods, such as those given in [4] [5] [11] among many others. Although these methods were capable of finding the solution to a given BSDE they were constrained by the amount of computation time required to find a solution, especially as the dimensionality of a given problem increased. In particular, these methods are affected by the "curse of dimensionality": as the dimension of the underlying problem increases the computation complexity to find an approximate solution to the problem increases exponentially in time. Thus these methods become infeasible to use for approximating solutions once a certain number of dimensions in the underlying problem is reached, and a significant amount of study began to find methods to mitigate these issues.

Recent advances in numerically approximating BSDEs utilize artificial neural networks (ANNs) in order to aide in the approximation. This algorithm utilizes a merged PDE and BSDE formulation in order to simulate the BSDE *forward* in time. By doing so, the BSDE itself can be simulated at every step along with the SDE. A neural network is used to simulate the Z stochastic process of the solution of a BSDE and in the process of training this neural network, backpropagation is used to refine an initial guess of the answer Y_0 of the BSDE to the actual solution. Since this simulation manages to simulate both the SDE and the Z process of the BSDE in the same loop, it avoids the exponential computational costs associated with Monte-Carlo algorithms, which required the simulation of the entire underlying SDE prior to the simulation of the associated BSDE.

In this thesis we will provide a historical overview of BSDEs, some information on what factors allow for there to be solutions to these equations, and explore several of the previously mentioned techniques for utilizing numerical solutions to these equations. In addition, we will explore alternative methods for solving non-linear PDEs utilizing numerical methods. The first chapter will introduce BSDEs proper as well as

certain variations of the simpler BSDE above. The second chapter will discuss numerical approximations of the solutions of these equations using traditional Monte-Carlo methods. The third chapter will give an overview of neural networks. The fourth chapter will discuss the use of neural networks to solve BSDEs of various forms. The final two chapters will be dedicated to future research steps and conclusions.

Notation

The usual n-dimensional Euclidean space is denoted by $\mathbb{R}^n$ throughout the sequel, with $|\cdot|$ and $\langle\cdot,\cdot\rangle$ the usual Euclidean norm and inner product respectively. We again consider a filtered probability space $(\Omega, \mathcal{F}, P)$ and denote by $L^2_{\mathcal{F}}(\Omega, \mathcal{R}^m)$ to be the set of all $\mathcal{F}$-measurable $\mathbb{R}^m$-valued square integrable random variables.

Similarly, $L^2_{\mathcal{F}}(0,T;\mathbb{R}^m)$ is taken to be the set of all $\mathcal{F}$-progressively measurable processes X which take values in $\mathbb{R}^n$ such that $\mathbb{E}\left[\int_0^T |X(t)|^2 dt\right] < \infty$. We also denote $L^2(\mathcal{F}_t)$ as the space of all $\mathcal{F}_t$ measurable processes from $[0,T]\times\mathbb{R}^m \to \mathbb{R}$.

$L^2_{\mathcal{F}_T}(\Omega, W^{(1,\infty)}(\mathbb{R}^n;\mathbb{R}^m)$ is the set of all function $f : \mathbb{R}^n \times \Omega \to \mathbb{R}^m$ where $\omega \in \Omega \mapsto f(x,\omega)$ is uniformly Lipschitz for every $x \in \mathbb{R}^n$ as well as $\mathcal{F}_T$-measurable for all $x \in \mathbb{R}^n$. In addition, $f(0,\omega) \in L^2_{\mathcal{F}}(\Omega;\mathbb{R}^m)$.

∇f is the gradient of the function, for some function $f(x_1,\ldots,x_n)$, $\nabla f = \left(\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n}\right)$
$\Delta_x f$ for some function f is the *Laplace operator* on that function, that is the square of the gradient. $\Delta_x f = \sum_{i=1}^n \frac{\partial^2 f}{\partial x_i^2}$

Chapter 1

Backward Stochastic Differential Equations

1.1 Overview

Backwards Stochastic Differential Equations (BSDEs) are Stochastic Differential Equations where the terminal condition is given. In order to discuss BSDEs we first need to understand some of the underlying theory. A basic knowledge of probability theory is assumed. [1] Every BSDE is built on top of a probability space equipped with a *Filtration*. Formally we define a filtration in the following way:

Definition 1.1.1 (Filtration) *Given a probability space $(X, \mathcal{F}, P)$, a sequence of σ-algebras given by $(\mathcal{F}_t)_{t\in[0,T]}$ where $\mathcal{F}_t \subseteq \mathcal{F}$ for all $t \in [0, T]$ is a filtration if it satisfies $\mathcal{F}_s \subseteq \mathcal{F}_t$ for any pair $s, t \in [0, T]$ such that $s \leq t$.*

A filtration

Definition 1.1.2 (Brownian Filtration) *A Brownian Filtration is defined to be, for some Brownian motion W_t on the measurable space $(\mathbb{R}^d, \Sigma)$,*

$$\mathcal{F}_t = \sigma\left\{W_s^{-1}(a)|0 \leq s \leq t, a \in \Sigma\right\}$$

[1] For a review, check Appendix A.

where $\sigma(A)$ is the smallest sigma-algebra generated by the set A.

The underlying filtration for any given BSDE is generally assumed to be a filtration generated by a Brownian motion process denoted by W_t. A filtration is meant to represent the *information* available at a given time t. For a given stochastic equation, we want that at any time t we can find the value of a given stochastic process only from the information that we have experience in the past. For some time t, any process that is $\mathcal{F}_t$-measurable would have this property. If a process has this property, that process is called *adapted*. More formally,

Definition 1.1.3 (Adapted Process) *A process X_t is adapted to the filtration $\mathcal{F} = \{\mathcal{F}_t\}_{t\in[0,T]}$ when, for every time $s \in [0,T]$, X_s is $\mathcal{F}_s$ measurable.*

In general we want the solutions to our BSDEs to be adapted to the underlying filtration. We now have what we need to properly define BSDEs and their solutions:

Definition 1.1.4 (Backward Stochastic Differential Equation) *A Backward Stochastic Differential Equation is a Stochastic Differential Equation defined over a probability space $(\Omega, P, \mathcal{F})$ with an associated Brownian filtration. The equation takes the form*

$$Y_t = Y_T + \int_t^T g(s, Y_s, Z_s)ds - \int_t^T Z_s dW_s \tag{1.1}$$

$$Y_T = \xi \tag{1.2}$$

Where ξ is a given random variable, W_s a Brownian motion. A solution to a BSDE is a tuple of the form (Y_t, Z_t) where Y_t and Z_t are adapted random variables over a Brownian filtration $\{\mathcal{F}_t\}_{t\in[0,T]}$.

Generally we call the given deterministic function $g(t, y, z)$ the *driver* of the BSDE. There are several categories of BSDE, each of which can be associated to larger classes of PDEs.

1.1.1 BSDEs with Lipschitz Drivers

Linear BSDEs are the simplest form of BSDE. Here, the driver of the BSDE $g(t, y)$ is uniformally Lipschitz in y. This class of BSDEs were the first studied and give rise to solutions of linear parabolic PDEs. Here, a solution is required only to be a tuple (Y, Z) of stochastic processes. The process Z is necessary in order to ensure that the solution is adapted. The process Z is essentially the gradient of Y. Note that here there is not an underlying process X and this is essentially an SDE where instead of starting with an initial value, we start with a terminal value. Unlike PDEs, where this is a simple change, for SDEs one need to worry about whether the process is adapted to the filtration - that is, we need a process that does not depend on the values at a future time to find its value at the current time. It is for this reason that the process Z is introduced in the solution, as it allows us to create an adapted process where this otherwise would not be possible.

Existence of Solutions

The existence of solutions to BSDEs under various constraints has been studied extensively. The first results along these lines were given in [30] by Pardoux and Peng. It was first shown that BSDEs where the driver g is a Lipschitz function have unique solutions. If $g(t, y, z)$ is Lipschitz and such that $h(x)$ is an $\mathcal{F}_T$-measurable random variable with $||x||^2 < \infty$, then $(g, h(x))$ are said to be *standard parameters* of a given BSDE according to El Karoui et al. With standard parameters, we can assert the following theorem from [30]:

Theorem 1.1.1 (Existence of Solutions for BSDEs with Lipschitz drivers) *Consider a BSDE of the form 1.1. If the pair (g, Y_T) are standard parameters, than there exists a pair of stochastic processes (Y, Z) such that (Y, Z) is a solution of the BSDE 1.1.*

This was the initial result given for solutions to BSDEs. There is a proof available in [30], as well as a somewhat shorter proof in [10].

1.2 Other Types of BSDEs

1.2.1 Fully-Coupled Forward-Backward SDEs

Forward-Backwards SDEs, or FBSDEs are BSDEs where there is an underlying process X. This form of BSDE has been used extensively in financial applications where the price of some financial option is given by Y and the underlying process represents the stock or other asset that the financial option is based on. For our purposes we are dealing mainly with the use of FBSDEs and their relationship to quasi-linear PDEs. For this reason, we reproduce here a proof from [22].

There are several subclasses of FBSDEs which depend on the relationship between the BSDE Y and the underlying SDE X. In the case that the value of X depends upon the value of Y, in addition to Y relying on the value of X then the FBSDE is known as a *Fully-Coupled FBSDE*. In the case where this is not true, then the FBSDE can also be described as *partially coupled*. For instance, the Black-Scholes Model with price impact can be simulated through the use of a Fully-Coupled FBSDE, where the price of the put or call option is simulated by Y and the price of the underlying asset is simulated by X.

1.2.2 Analytical Solutions to BSDEs

The most obvious way to find a solution for a given BSDE is to assume that Y is some kind of function of X and t. This makes sense in the case where Y is a derivative of some underlying asset X

An analytic method for solving BSDEs is the four step scheme described in [23]. Again we consider a general BSDE of the form (1.1).

Assumptions for the Four-Step Method

The following assumptions are made in order to guarantee a solution to the following fully-coupled BSDE:

(A1) The fuctions b, $\hat{b}$, σ, $\hat{\sigma}$ and Y are smooth functions taking values in their respective domains with first order derivatives in x, y, and z. These derivatives are bounded by a constant $L > 0$.

(A2) We have that

$$\sigma(t, X_t, Y_t)\sigma(t, X_t, Y_t)^T \geq \nu(|Y_t|)I \forall (t, X_t, Y_t) \in [0, T] \times \mathbb{R}^n \times \mathbb{R}^m$$

for some positive continuous function $\nu(\cdot)$.

(A3) For any tuple $(t, x, y, z) \in [0, T] \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m \times n}$ the linear map $\hat{\sigma}_z(t, x, y, z) \in \mathcal{L}(\mathbb{R}^{m \times n})$ is invertible with inverse $\hat{\sigma}_z(t, x, y, z)^{-1}$ satisfying

$$||\hat{\sigma}(t, x, y, z)^{-1}||_{\mathcal{L}(\mathbb{R}^{m \times n})} \leq \lambda(|y|)$$

and for any $(t, x, y) \in [0, T] \times \mathbb{R}^n \times \mathbb{R}^m$ the set

$$\{\hat{\sigma}(t, x, y, z) | z \in \mathbb{R}^{m \times n}\} = \mathbb{R}^{m \times n}$$

and there exists a positive continuous function $K(\cdot)$ such that

$$\sup\{|z||\hat{\sigma}(t, x, y, z)|\hat{\sigma}(t, x, y, z) = 0\} \leq K(|y|)$$

(A4) There is a positive function μ and constants $C > 0$ and $\alpha \in (0, 1)$ such that g is bounded in $C^{2+\alpha}(\mathbb{R}^m)$ and for all $(t, x, y, z \in [0, T] \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m \times n}$ we have that

$$|\sigma(t, x, y)| \leq \mu(|y|), |b(t, x, y, 0)| \leq \mu(|y|), |\hat{b}(t, x, 0, z)| \leq C$$

Four Step Scheme

Given we have a BSDE that satisfies the above assumptions, we have the following way to attain a solution to the BSDE:

Step 1: Find a function of the form $z : [0, T] \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m \times n}$, where

$$z(t, x, y, p) = p\sigma(t, x, y, z(t, x, y, p))$$

for any 4-tuple $(t, x, y, p) \in [0, T] \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m \times n}$.

Step 2: The above function should be used to solve the parabolic system for the function $\phi : [0, T] \times \mathbb{R}^n$

$$\begin{cases} \frac{\partial \phi^k}{\partial t} + \frac{1}{2}\text{tr}\left[\frac{\partial^2 \phi^k}{\partial x^2}(\sigma\sigma^T)(t, x, \phi, z(t, x, \phi, \frac{\partial \phi}{\partial x}))\right] + \langle b(t, x, \phi, z(t, x, \phi, \frac{\partial \phi}{\partial x})), \frac{\partial \phi^k}{\partial x}\rangle \\ -h^k(t, x, \phi, z(t, x, \phi, \frac{\partial \phi}{\partial x})) = 0, \qquad (t, x) \in [0, T) \times \mathbb{R}^n, 1 \leq k \leq m, \\ \phi(T, x) = g(x), x \in \mathbb{R}^n. \end{cases} \tag{1.3}$$

Step 3: Use the above two functions to find a solution to the SDE

$$\begin{cases} dX_t = \hat{b}(t, X_t)dt + \hat{\sigma}(t, X_t)dW_t, t \in [0, T], \\ X_0 = x, \end{cases} \tag{1.4}$$

where the functions $\hat{b} = b(t, x, \phi, z(t, x, \phi, \frac{\partial \phi}{\partial x}))$ and $\hat{\sigma} = \sigma(t, x, \phi, z(t, x, \phi, \frac{\partial \phi}{\partial x}))$

Step 4: We can now set

$$Y_t = \phi(t, X_t), Z_t = z(t, x, \phi, \frac{\partial \phi}{\partial x}).$$

The four-step method makes some assumptions on the generator of the BSDE. In particular it requires that the functions in the BSDE are Lipschitz continuous. Furthermore it is not always the case that we can find analytic solutions to these problems. Other analytical solutions to BSDEs require similar assumptions on the generator of the BSDE, and therefore become impractical for BSDEs outside of the scope of these assumptions. Therefore for such BSDEs, numerical methods and significant efforts have been made to design and improve methods for solving these equations numerically, given how often such exceptions occur in fields such as mathematical finance.

1.2.3 2BSDEs

A 2BSDE is a specific BSDE which has a driver of the form $g(t, X_t, Y_t, Z_t, \Gamma_t)$ where here the term Γ_t refers to the second order derivatives of the BSDE Y. This is equivalent to the gradient of the process Z, and so for a 2BSDE, we have the following structure for the stochastic processes:

$$\begin{aligned} dY_t &= g(t, X_t, Y_t, Z_t, \Gamma_t)dt + Z_t dX_t \\ dZ_t &= A_t dt + \Gamma_t dX_t \\ Y_T &= h(X_T) \end{aligned}$$

BSDEs have many applications, especially in mathematical finance where they can be used to price certain financial instruments (see e.g. [19]). They also find use in theoretical economics, optimal stochastic control among other areas. One of the main attractions to them for mathematicians is their connection to PDEs through the Feynman-Kac formula.

1.2.4 BSDEs with Quadratic Drivers

For quite a while the existence of solutions to BSDEs with only these standard parameters were proven. However, in 2000 the paper [20] proved the existence of solutions for BSDEs such that the solution has quadratic growth in the process Z. This has interesting implications for the resultant solutions of the BSDEs, and in particular the domains that they are defined on compared to "standard" BSDEs of the form discussed above.

In addition Kobylanski showed that there exists unique solutions for BSDEs of the form 1.1 where the driver $g(t, Y_t, Z_t)$ satisfies the inequality

$$|g(t, Y_t, Z_t)| \leq b + c|Z_t|^2 \text{ a.s.} \tag{1.5}$$

1.3 Feynman-Kac Connection for BSDEs

Consider the FBSDE given by

$$\begin{aligned} dX_s &= b(s, X_s)ds + \sigma(s, X_s)dW_s, \qquad t \leq s \leq T, \\ X_t &= x, \\ -dY_s &= f(s, X_s, Y_s, Z_s)ds - Z_s dW_s, \qquad t \leq s \leq T, \\ Y_T &= g(X_T), \end{aligned}$$

and the following PDE

$$0 = -\frac{\partial u}{\partial t} - \frac{1}{2}\sum_{i,j=1}^{n}(\sigma\sigma^T)_{ij}(t,x)\frac{\partial^2 u}{\partial x_i \partial x_j} - \sum_{i,j=1}^{n} b_i(t,x)\frac{\partial u}{\partial x_i} + f(t, x, u, \nabla u_x),$$

$$u(T, x) = g(x)$$

Where $u(t, x) : [0, T] \times \mathbb{R}^d \to \mathbb{R}$ is the solution to the PDE where we assume u is at least C^2 and the following assumptions hold for b and σ,

1. $b(t, x)$ and $\sigma(t, x)$ are Lipschitz in x. That is, $|b(t, x_1) - b(t, x_2)| + |\sigma(t, x_1) - \sigma(t, x_2)| \leq K|x_i - x_2||$.
2. We have $|b(t, x)|^2 + |\sigma(t, x)|^2 \leq K^2(1 + |x|^2)$.
3. $\sigma\sigma^T$ is a positive definite $d \times d$ matrix.

It follows from Itô's formula applied to $u(s, X_s)$ that

$$\begin{aligned} Y_s &= u(s, X_s) \\ Z_s &= (\sigma^T \nabla u)(s, X_s) \end{aligned}$$

which gives us the solution processes of the FBSDE system in terms of a solution u of a PDE so long as $u \in C^2(\mathbb{R} \times \mathbb{R}^n)$.

Furthermore, it is shown in for example [20] that a viscosity solution to the PDE is given by

$$u(t, x) := Y^{t,x}(t).$$

It is however not possible to use a BSDE to find a classical solution to a given PDE. For more on the relationship between non-linear PDEs and solutions to BSDEs, see [40][20] or Appendix C for more information on this relationship.

Chapter 2

Monte-Carlo Numerical approximations to FBSDEs

We consider the FBSDE given in the following form:

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

$$Y_t = h(X_T) + \int_t^T g(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s dW_s.$$

Here $X = (X_t : 0 \leq t \leq T)$ is a d-dimensional stochastic process, represented in the form of a forward stochastic differential equation (SDE) and $Y = (Y_t : 0 \leq t \leq T)$ is a stochastic process represented as a BSDE. As usual, W_t denotes an n-dimensional Weiner Process, and h is a deterministic functional which determines the terminal conditions of the BSDE. In the sequel, $\Phi(X)$ is approximated by $\Phi^N(P^N_{t_N})$ where $(P^N_{t_k})_{0\leq k\leq N}$ is a Markov chain where the first components are given by the components of $(X^N_{t_k})_{0\leq k\leq N}$. Here, the P_{t_k} are in fact projections onto subspaces of $L^2(\mathbb{R}^d)$.

2.1 Assumptions

In general, the following assumptions are made to ensure the existence and uniqueness of a solution (Y, Z) to the above BSDE,

(A1) The functions $(t, x) \mapsto f(t, x)$ and $(t, x) \mapsto \sigma(t, x)$ are uniformly Lipschitz continuous with respect to $(t, x) \in [0, T] \times \mathbb{R}^d$.

(A2) $|g(t_2, x_2, y_2, z_2) - g(t_1, x_1, y_1, z_1)| \leq C(|t_2 - t_1|^{\frac{1}{2}} + |x_2 - x_1| + |y_2 - y_1| + |z_2 - z_1|)$ for any such pair $(t_i, x_i, y_i, z_i) \in [0, T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^n$, $i = 1, 2$.

(A3) for any continuous function s^1 and s^2,

$$|\Phi(s^1) - \Phi(s^2)| \leq C \sup_{t \in [0,T]} |s^1 - s^2|.$$

An additional assumption is needed on the function Φ in [11].

(A4) $\Phi^N(\cdot)$ is Lipschitz continuous, such that $\sup_N |\Phi^N(0)| < \infty$. Furthermore, $E[|P^{N,k_0,x}_{t_N} - P^{N,k_0,x'}_{t_N}|^2] + E[P^{N,k_0,x}_{t_{k_0+1}} - P^{N,k_0,x'}_{t_{k_0+1}}|^2] \leq C|x - x'|^2$ uniformly continuous in k_0 and N.

2.2 Euler Scheme for SDE approximation

In order to solve any given BSDE, there is an always an expectation to solve for the underlying SDE. Thus it is first necessary to estimate that SDE, although this is generally easier to do when the BSDE is decoupled. The underlying SDE for a BSDE is discretized using a traditional log-Euler scheme. Here the interval $[0, T]$ is partitioned by the sequence of times $\{t_i\}_{i=0}^N$. Thus there are $N + 1$ different timesteps to approximate the SDE on. The initial condition gives t_0, and the remaining timesteps are found using the following discretized SDE:

$$d\hat{X}_{t_i} = X_{t_{i-1}} + b(t, X_{t-1})(t_i - t_{i-1}) + \sigma(t, X_{t-1})(W_{t_i} - W_{t_{i-1}})$$

where as usual dW represents the expected Brownian motion associated with the SDE. It is well known that this scheme gives an accurate approximation of the SDE as N tends to infinity.

Monte-Carlo Method for BSDE Estimation

With a simple method of the approximation of an SDE now available to us, it is possible to use several methods to attempt to find a solution for the BSDE. The discretization of $[0, T]$ is kept when approximating the BSDE Y. The following gives an adequate representation of the BSDE under this discretization.

$$Y^N_{t_{k+1}} \approx Y^N_{t_k} + g(t_k, X^N_{t_k}, Y^N_{t_k}, Z^N_{t_k})\Delta t_k + Z^N_{t_k}\Delta W_{t_k}$$

and therefore

$$Y^N_{t_k} \approx Y^N_{t_{k+1}} - g(t_k, X^N_{t_k}, Y^N_{t_k}, Z^N_{t_k})\Delta t_k - Z^N_{t_k}\Delta W_{t_k} \tag{2.1}$$

Here $\Delta t_k = t_{k+1} - t_k$ and $\Delta W_{t_k} = W_{i,t_{k+1}} - W_{i,t_k}$ for $1 \leq i \leq q$ the i'th component of the process.

Using the following:

$$0 = E[\Delta W_{i,t_k}(Y_{t_k} + g(t_k, X_{t_k}, Y_{t_k}, Z_{t_k})\Delta t_k|\mathcal{F}_{t_k}] \approx E[\Delta W_{i,t_k} Y_{t_{k+1}}|\mathcal{F}_{t_k}] - Z_{i,t_k}\Delta t_k$$

We can then define a discretization for Z using the above equation:

$$Z^N_{i,t_k} = \frac{1}{\Delta t} E[Y^N_{t_{k+1}} \Delta W_{i,k}|\mathcal{F}_{t_k}]$$

and by applying expected values to 2.1 we get the following approximation for $Y_{t_k}^N$:

$$Y_{t_k}^N = E[Y_{t_{k+1}}^N | \mathcal{F}_{t_k}] + g(t_k, S_{t_k}^N, Y_{t_k}^N, Z_{t_k}^N)\Delta t$$

The following theorem gives a bound on the squared error of the resulting approximation based on the number of partitions N.

Theorem 2.2.1 *Under the above assumptions, there is a constant C such that for any N and $|\pi|$ sufficiently small, the following inequality holds.*

$$\sup_{0\leq t\leq T} E[|Y_t - Y_t^N|^2] + \int_0^T E[|Z_t - Z_t^N|^2] \leq C\left(|\pi| + E[|\xi - \xi^\pi|^2] + \left(\frac{1}{2} + C|\pi|\right)^N\right)$$

This theorem is from [3], we reproduce a proof of this theorem is reproduced in the Appendix B.

The main step for finding a numerical approximation using the least-squares Monte-Carlo method utilizes an approximation of the above conditional expectations, by taking their projection onto a set of bases functions for $L_2(\Omega, \mathcal{F}, P)$. The function bases, denoted $p_{\ell,k}$ for $\ell \in \{t_i\}_{i=1}^n$, $1 \leq k \leq d$ can vary, potential functions suggested in [11] include:

i) *Hypercubes*: Each $p_{\ell,k}$ is chosen to be the indicator function of some hypercube in a domain $D \subseteq \mathbb{R}^d$ centered on the point P_0^N. Therefore, $D = \prod_{i=1}^{d'}(P_{0,i}^N - R, P_{0,i}^N + R]$ is partitioned into hypercubes with edge length of size $\delta > 0$ for some given δ, where the indicator functions are given for each of the resulting hypercubes. (i.e. each $p_{\ell,k}$ is associated to the indicator function of a hypercube $D' \subseteq D$ where $D' = (P_{0,1}^N + m_1\delta, P_{0,1}^N + (m_1+1)\delta \times \cdots \times (P_{0,d}^N + m_d\delta, P_{0,d}^N + (1+m_d)\delta]$ where $m_1, \ldots, m_d$ are chosen scalars).

ii) *Voronoi Partition*

The basis functions $p_{\ell,k}$ are taken to be indicator functions of a Voronoi partition of the domain, the centers of each partition are simulations of the Markov

Chain P^N. Specifically, each partition is given by $C_{k,i} = \{x : |x - P_{t_k}^{N,M+i}| < \inf_{j\neq i}|x - P_{t_k}^{N,M+j}|\}$.

iii) *Global Polynomials*

The basis vectors are given as polynomials of d' variables of degree less than d_y and $p_{\ell,k}$ is taken as the polynomial basis of degree less than d_z.

Where in the above, d' denotes the dimension of the state space of $(P_{t_k}^N)_k$.

The choice of function basis allows for more or less specificity of the projection error in the approximation of the conditional expectations related to the BSDE. In particular, the following theorem in [11] specifies the potential projection error when taking repeated Picard iterations to estimate $(Y_{t_k}^N, Z_{\ell,t_k}^N)$ via the projection estimates $Y_{t_k}^{N,I,I}$ and $Z_{t_k}^{N,I,I}$. Here N stands for the current Picard iteration, I is the number of partition points, and the notation $Y^{N,I,I}$ refers to the backwards estimation of Y using a projection onto a subspace of $L^2(\mathbb{R})$ using a basis of one of the above forms that is dense in $L^2(\mathbb{R})$, from the terminal time index N, where I is the number of Picard iterations.

Theorem 2.2.2 *Assume* $(A1) - (A3)$. *For small* Δt,

$$\begin{aligned}
&\max_{0\leq k\leq N} E|Y_{t_k}^{N,I,I} - Y_{t_k}^N|^2 + \Delta t \sum_{k=0}^{N-1} E|Z_{t_k}^{N,I,I} - Z_{t_k}^N|^2 \\
&\leq C(\Delta t)^{2(t-1)}[1 + |S_0|^2 + E|\Phi^N(P_{t_N}^N)|^2] \\
&\qquad + C\sum_{k=0}^{N-1} E|\mathcal{R}_{p_{0,k}}(Y_{t_k}^N)|^2 + C\Delta t \sum_{k=0}^{N-1}\sum_{i=1}^{q} E|\mathcal{R}_{p_{\ell,k}}(Z_{\ell,t_k}^N)|^2
\end{aligned}$$

Where $\mathcal{R}_{p_{\ell,k}}(\cdot)$ specifies the projection error. The choice of basis affects the above error, as well as the speed of computations.

In [5] it is suggested to take a basis that acts as a martingale. They give some examples similar to above, for instance showing that indicator functions of hypercubes and monomials can be used to give this effect, since the expectations can be explicitly

computed. For instance, for hypercubes it can be calculated, where $\eta_{a,b} = \mathbf{1}_{[a,b]} = \mathbf{1}_[a_1, b_1] \times \cdots \times [a_D, b_D]$:

$$E[\eta_{a,b}(X_T)|X_{t_k} = x] = \prod_{d=1}^{D} E[\mathbf{1}_{[a_d,b_d}(X_d, T)|X_{d,t_k} = x_d] = \prod_{d=1}^{D} \mathcal{N}(\bar{b}_d) - \mathcal{N}(\bar{a}_d).$$

where $\mathcal{N}$ is the CDF for a standard normal, $\bar{a}_d, \bar{b}_d$ are given by

$$\bar{y}_d = \frac{\log(y_d/x_d) - (\mu - \frac{\sigma^2}{2})(T - t_k)}{\sigma\sqrt{T - t_k}}.$$

This allows them to assume the following conditional expectation can also be show in closed form:

$$E[\Delta W_{i,t_k} Y^N_{t_{k+1}} | \mathcal{F}_{t_k}]$$

and thus only a single conditional expectation needs to be numerically calculated to approximate solutions to the given BSDE.

2.3 Numerical Solutions to BSDEs with Quadratic Growth

Gobet et al. and Bender et al. provided an example of a least-squares Monte-Carlo method for approximating BSDEs with Lipschitz continuous drivers, however methods for approximating the solution to BSDEs with drivers with quadratic growth were not created for several years, despite knowledge of the existence to the solutions of these formulas thanks to Kobylanski (2000).

Methods were eventually developed, for instance [18] approximates solutions by using Cole-Hopf exponential transformation, however this only applies for a restricted set of generators with quadratic growth. Other methods utilized distorted time horizons, as in for instance [12], where approximations of quadratic BSDEs are studied in the context of Dynamic Programming equations. In general, when working with quadratic

BSDEs unless specific assumptions are made on the form of the generator, it is the case that exponential terms creep into the error terms of the algorithms used to solve them. Thus approximating quadratic BSDEs come with its own challenges.

2.4 Other Methods for Approximating Solutions for BSDEs

While the above solution to the problem of finding a solution does work with a reasonable error rate, convergence can be slow. More recently, alternatives to using the Monte Carlo method for finding solutions to BSDEs have begun to be developed, foremost among these the Deep BSDE method. These methods take advantage of a class of functions, well known as neural networks, or artificial neural networks (ANN for short).

2.4.1 Branching Diffusion processes

Recent work exemplified by [7] uses branching diffusion processes to solve BSDEs. Non-linearities are dealt with by approximating a non-linear driver $g(X)$ with a function

$$c(X)\mathbf{1}_{[g(X)+\varepsilon \geq e^r Y]}$$

and letting epsilon vanish to 0. Here $c(x)$ is assumed to be a continuous function with polynomial growth. Monte Carlo estimations are then done by simulating the underlying SDE of a weakly coupled SDE. An additional method was also used using traditional local polynomial approximation of q and associated Monte-Carlo estimation utilizing a Picard iteration scheme similar to Gobet et al. and Bender et al.

The approximation method utilizing local polynomial approximation was reported as particularly unstable, however the approximation utilizing noise to smooth the driver $g(t, x, x')$ was significantly interesting and thus avoided the use of a Picard iteration scheme as in the previous approximation technique. However, this algorithm utilized a grid space-time layout and was similarly susceptible to the curse of dimensionality that the Deep BSDE and Deep PDE Method seek to avoid in their approximation

methods.

2.4.2 Polynomial Chaos Expansion

BSDEs can be approximated through the use of a technique known as Wiener Chaos Expansion (for more details, see [8]). In this method, given the following Picard scheme for approximating a solution, such as

$$Y_{t_k}^N = E\left[Y_T + \int_0^T g(s, Y_s^{N-1}, Z_s^{N-1})ds|\mathcal{F}_{t_k}\right] - \int_0^T g(s, Y_s^{N-1}, Z_s^{N-1})ds \tag{2.2}$$

It is required that we find an estimation for the random variable

$$F^N = Y_T + \int_{t_k}^T g(s, Y_s^{N-1}, Z_s^{N-1}).$$

Wiener Chaos Expansion proposes to use is used to approximate F^N in the following manner.

First, note that F^N can be decomposed into the following

$$F^N = E[F^N] + \sum_{k\geq 1}\sum_{|n|=k} d_k^n \prod_{i\geq 1} K_{n_i}\left(\int_0^T h_i(s)dW_s\right). \tag{2.3}$$

Here K_{n_i} is the $n_i' th$ Hermite polynomial (given by $H_n(x) = (-1)^n \exp\{\frac{x^2}{2}\}\frac{d^n}{dx^n}\exp\{-\frac{x^2}{2}\}$.), which are used as they form a dense orthogonal basis for $L^2(\mathbb{R})$.

By taking a finite sequence $n_1, \ldots, n_q$ of integers and a finite sum of terms p for the sum on the LHS, we can approximate the above random variable by

$$F^N \approx E[F^N] + \sum_{1\leq k\leq p}\sum_{|n|=k} d_k^n \prod_{1\leq i\leq q} K_{n_i}\left(\int_0^T h_i(s)dW_s\right).$$

Which can thus be used to approximate the solution Y^N by taking the conditional expectation above. In addition, one needs to find Z^N from taking the Malliavan derivative of that expected value, details of which are beyond the scope of this work, for details see [8].

2.4.3 Approximation using Neural Networks

While Monte-Carlo methods have been well researched, they suffer from the so-called "curse of dimensionality" - that is, BSDEs become more difficult to compute as the dimension of the given BSDE increases. Several proofs have been published showing that certain neural network architectures successfully avoid this problem (see e.g. [1][14]). Neural networks have been investigated as potential solutions to partial differential equations since at least 1999 (see [34] for some additional details on these methods) and for ODEs even earlier [21][24], however their use for finding solutions to BSDEs is relatively new. In the next chapter, we begin with a brief overview of neural network structures so as to better understand their use when solving these equations.

Chapter 3

Neural Networks, the Deep BSDE Method, and Deep PDE Methods

A multi-layer neural network is a set of equations that have multiple layers $k = 1, \ldots, m$ where m is the total number of layers of the neural network. Let $\mathbf{v}^0$ be the input vector. Each subsequent layer is of the form

$$\sigma(\mathbf{b}^k + W^k \mathbf{v}^{k-1})$$

where here $\mathbf{b}^k$ is a vector offset, known as the bias of the given layer, and W^k is a matrix of weights for the layer. The function σ is known as the activation function, which is a continuous function that takes values in $[0, 1]$. The choice of σ can greatly affect the performance of the neural network. These weights and biases are in general random at the initialization of the neural network, and are updating through training methods. These training methods allow for the neural network to approximate functions with varying degrees of success, based upon the complexity of the functions and the training sample used.

There are several different architectures that can be used when creating deep learning neural networks, along with mixed networks. One type of network that has been applied to find solutions to BSDEs is the affine network model. The details for their solutions are found in [2]. Below we detail the network architecture, and cover its application to the problem in Chapter 4.

3.1 Overview of Artificial Neural Networks

3.1.1 History

Neural networks are relatively simple functions that can manage to approximate a very large class of functions, in many different domains of applications. Artificial Neural Networks are seeing applications in many areas, such as artificial intelligence, computer vision and many other fields. How did we figure out what we could do with these particular setups and how did it come to be used to find solutions to BSDEs?

Artificial Neural Networks are mathematically constructed to attempt to process input data in a similar way to how it is thought that the brain processes data. In 1943, a model on the brain on the basis of neurons was proposed by Warren McCulloch and Walkter Pitts, where neurons were connected and considered to be in a binary state, either on or off [27]. It was quickly seen however that neurons were not binary as described in this early model, but in fact had many characteristics that suggested non-linearity. In 1949, Donald Hebb published the book "The Organization of Behaviour" which theorized that neuron connections are formed when particular neurons tend to fire together. If no connection is established between the two neurons, then such a connection betweens to be established or is made stronger when they fire at the same time [16]. This idea came to be known as Hebbian Learning.

The perceptron was an idea introduced in the 1958 by Frank Rosenblatt, which was the predecessor of the modern neural network, in the sense that although perceptrons had binary activations (unlike today's neural networks where activation of units takes on a continuum of values), but the weights could be learned from successive inputs. Although the perceptron and other methods using linear functions began to find success, by 1969 it was known that there were many downsides to the use of perceptrons.

Work in the field largely stagnated until the introduction of the backpropagation method for neural network learning, which was not introduced until 1986, in a work by Rumelhart, Hinton and Williams [36]. By the 90s, it had become well established as a useful technique for training artificial neural networks, and their was significant progress developing ANNs trained using backpropagation techniques and variants of

it.

The structure of a neural network greatly affects how easily it can approximate functions of significant computational complexity. Prior to 2006, the main form of neural networks used was a 2-layer model (i.e. one with a single hidden input layer), as prior to this point any higher depth neural network was difficult to train [6]. It was eventually discovered that applying unsupervised learning techniques to each layer individually results in significantly lower after supervised learning to properly optimize these networks, and thus deeper neural networks became viable for tasks.

3.1.2 Structure

Central to the idea of a neural network is that of an activation function, which is a non-linear function that allows for the universal approximation powers of a neural network. We begin by defining some useful concepts for which functions allow for the univeral approximation.

Definition 3.1.1 (Activation Function) *An activation function is a continuous function $\sigma : \mathbb{R}^m \to \mathbb{R}^m$ for some m.*

Definition 3.1.2 (Artificial Neural Network) *Let $d, m, \ell_1, \ldots, \ell_n \in \mathbb{N}$. An artificial neural network is a function $f : \mathbb{R}^d \to \mathbb{R}^m$ such that it can be decomposed into a composite sequence of functions of the form*

$$a_n = \sigma_n(g_{n-1}(a_{n-1}))$$

where n is the number of layers, $g_i : \mathbb{R}^{\ell}_{i-1} \to \mathbb{R}^{\ell_i}$ are each linear functions, and each σ_i is an activation function.

An artificial neural network is generally used as a function approximator - data is fed in and the network is "trained" through various methods. However, the activation function plays an important role in *which* functions an ANN is capable of approximating. In particular, with certain activation functions, ANNs act as **universal approximators** in the following sense

Definition 3.1.3 *A neural network is a universal approximator over a space Ω if, for a given activation function f, the set $\Sigma(f) = span\ \{f(y \cdot x + \theta) | y \in \mathbb{R}^n, \theta \in \mathbb{R}\}$ is dense in $C(\Omega)$.*

Which activation functions allowed for these universal approximation capabilities was intensely studied during the 1980's. In works such as [9], for instance, it is discussed how a certain subset of functions allow for universal approximation.

Definition 3.1.4 (Sigmoidal) *A function is a* sigmoidal *function if it is continuous and*

$$f(x) \to \begin{cases} 1 & as\ x \to +\infty \\ 0 & as\ x \to -\infty \end{cases}$$

In particular as the study of activation functions progressed, new concepts such as the following were introduced:

Definition 3.1.5 (n-Discriminatory Function) *A function $f : \mathbb{R} \to \mathbb{R}$ is discriminatory for some $n \in \mathbb{N}$ if the constraint*

$$\int f(y \cdot x + \theta) d\mu(x) = 0 \text{ for all } y \in \mathbb{R}^n, \text{ and } \theta \in \mathbb{R}$$

where μ is a signed Borel measure, holds only for $\mu = 0$.

Any function which is n-discriminatory for every n is considered to be *discriminatory*. It turns out that it is precisely the discriminatory property that gives the universal approximation capability to a neural network, at least when the activation function is consistently used for all nodes in the network, as given by the following result from [9]

In the Deep BSDE method, the ReLU function is given as an appropriate activation function. The details of the ReLU function are discussed below.

There are many different types of neural networks. For our purposes, we need only look at feedforward neural networks which are the basic kind. Here each g_i above would take the form $g_i(x) = W_i x + b_i$ where $W_i \in \mathbb{R}^{\ell_i \times \ell_{i-1}}$ and $b_i \in \mathbb{R}^{\ell_i}$.

3.1.3 Activation functions

For neural networks, there has been a great deal of research into what forms activation functions should take. Examples include the more modern rectifier function,

$$\sigma(x) = \max\{0, x\},$$

which is used for instance in [2] and has achieved significantly popularity. Another activation function is the *sigmoid*,

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

which provides a softer increase from 0 to 1 than the rectifier.

ReLU

The ReLU activation function is the function given by

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

The ReLU function is used in the Deep BSDE algorithm that we will be discussing later. It's advantage is that it makes a clear distinction between activated and unactivated nodes and thus can make training easier. However, it is not differentiable at all points, which can introduce inaccuracies.

Tanh

The hyperbolic tan function is a sigmoidal function which is differentiable at all points. A neural network that uses exclusively tanh will thus produce a containuous approximation. This function is used as the activation function for the layers in the Deep PDE algorithm that we will review later.

3.2 Training of Neural Networks

Let a^L denote the output of layer L. Let W^k denote the weight matrix of the k'th layer, and b^k denote the bias vector for the layer k. In order to train a neural network, we must find W^k and b^k for each layer, such that the input $\mathbf{x}$ produces an output $\bar{y}$ which is a reasonable approximation to the function that we wish to approximate. One way to do this is through the method of *backpropagation*.

Let $C(W^k, b^k)$ be a cost function. We wish to use backpropagation in order to minimize this cost function with respect to the parameters of the neural network, by choosing the weights and biases for each node to do so. We let (x, y) denote a training sample and denote by $\mathcal{S}$ the set of all such training samples. Then we can think of the cost function in terms of $C(W^k, b^k) = \sum_{(x,y)\in\mathcal{S}} \mathcal{L}(y, \bar{y}) = \mathcal{L}(y, \bar{y})$, which gives that this cost function is the sum of the losses gained over a batch of the training samples.

3.2.1 Universal Approximation

It is well documented that neural networks using discriminatory functions are capable of approximating any continuous and bounded function with enough training data. This is shown through the *Universal Approximation Theorem*. Details of this theorem can be found in for instance [33]. It is proved there that so long as a given activation is non-polynomial when used with a feedforward neural network architecture, the resulting functions that can be produced by the neural network is dense in $C(\mathbb{R}^n)$. This feature of neural networks is exploited in the Deep BSDE method as well as the Deep PDE method which we will review in a later chapter to find approximations to solutions of BSDEs and PDEs respectively.

3.2.2 Structures

In general, the most basic neural network is a feedforward neural network - one in which the layers of the network are laid out in a linear fashion. Each layer takes the output from the previous layer and provides that output to the next layer.

This is a rather simple layout but it has some problems. In particular, as the number of layers grows it becomes more difficult to train them effectively via backpropagation. This is due to a process known as *gradient explosion* whereby the gradients being tracked during backpropagation either increase substantially or become negligible after a few layers. For this reason and others, other ANN architectures have been developed that help to avoid this problem, generally by keeping a persistent memory throughout the layers that can be referenced. We will discuss two of the most relevant ones below, as they will be useful in understanding one of the algorithms we will be examining in a later chapter.

Recurrent Neural Networks

A recurrent neural network is a specific kind of neural network, often with a "loop" of some sort in its structure. That is, data from the previous time step is fed back in to the neural network at the current time step. In this way the neural network is able to remember what has previously happened and change it's behaviour accordingly.

For instance, consider the following layer design,

Long Short-Term Memory Networks

Long Short-Term Memory Networks (LSTM) are a specific form of RNN such that there is a persistent form of memory that can be accessed and potentially modified during each timestep. This is separate from the output of the layers itself, as is given at each timestep. There are a wide variety of LSTM, but they all share in common a persistent memory, that can be accessed through gates. There are several "gates" in an LSTM Neural Network, for modifying the memory. Virtually every form of LSTM will feature a *forget gate* which does unit

3.3 Deep BSDE Algorithm

Neural networks can be applied to find approximate solutions to BSDEs, (see for instance [39][13][14]). This allows solutions to systems of semi-linear and quasi-linear

and fully non-linear parabolic PDEs to be approximated using neural networks, due to a generalization of the Feynman-Kac duality between solutions of BSDEs and solutions of BSDEs.

As an example of these techniques consider the following FBSDE,

$$\begin{aligned} dX_s &= b(s, X_s)ds + \sigma(s, X_s)dW_s, \qquad t \leq s \leq T, \\ X_t &= x, \\ -dY_s &= f(s, X_s, Y_s, Z_s)dds - Z_s dW_s, \qquad t \leq s \leq T, \\ Y_T &= g(X_T), \end{aligned}$$

[1] Recall the relationship between BSDEs and PDEs. The elements are defined according to the solution to a PDE of the form

$$0 = -\frac{\partial u}{\partial t} - \frac{1}{2}\sum_{i,j=1}^{n}(\sigma\sigma^T)_{ij}(t,x)\frac{\partial^2 u}{\partial x_i \partial x_j} - \sum_{i,j=1}^{n} b_i(t,x)\frac{\partial u}{\partial x_i} + f(t, x, u, \nabla u_x),$$

$$u(T, x) = g(x)$$

where b and σ are assumed to be continuous functions in terms of both t and x, and f is assumed to be Lipschitz.

In particular, each of these are related to the PDE in the following way.

$$\begin{aligned} Y_t &= u(t, X_t) \\ Z_t &= (\sigma\nabla_x u)(t, X_t) \end{aligned}$$

where $Y_t \in \mathbb{R}$, Z_t, where u is the solution to the above PDE.

[1] In actuality the algorithm is defined more generally for 2BSDEs where Z is of the form $Z_t = Z_0 + \int_0^t A_s ds + \int_0^t \Gamma_s dW_s$ and ANNs are used to approximate both A and Γ, however one can also just approximate Z for BSDEs that do not deal with second derivatives.

In order to approximate a solution to the PDE, we use a forward time discretization of the BSDE by the following, for a partition π of $[0,T]$ with mesh size $\sup_{0\leq k\leq N}(t_{k+1}-t_k)$ sufficiently small. Then

$$X_0 = \xi, \tag{3.1}$$
$$X_{t_{n+1}} \approx X_{t_n} + b(t_n, X_{t_n})(t_{n+1}-t_n) + \sigma(t_n, X_{t_n})(W_{t_{n+1}} - W_{t_n}) = X_t^N, \tag{3.2}$$
$$Y_{t_{n+1}} \approx Y_{t_n} + f(t_n, X_{t_n}, Y_{t_n}, Z_{t_n})(t_{n+1}-t_n) + Z_{t_n}(W_{t_{n+1}} - W_{t_n}) = Y_t^N, \tag{3.3}$$
$$\tag{3.4}$$

We approximate Z_t in the following manner

$$Z_{t_{n+1}} \approx Z_{t_n} + A(X_{t_n})(t_{n+1}-t_n) = Z_t^N$$

Here $A(x)$ is an artificial neural network with 2 hidden layers, input dimension d, d nodes per hidden layer and output dimension d.

This neural networks has a set of parameters than can be labelled θ which are trained using backpropagation until it minimizes the loss function given by $|Y_T^N - g(X_T^N)|^2$ (recalling that in the definition of the BSDE, $Y_T = g(X_T)$) . The Deep BSDE method works then by thinking of the first of these parameters, θ_1 as the initial value Y_0 of the BSDE given above. Therefore, by training these neural networks we end up finding a solution to the initial condition of the BSDE Y_0 by simply using backpropagation to optimize the initial estimate given by a randomly selected value.

The Deep BSDE method uses the following algorithm for computing Y_0:

Initialize number of partition points N, dimensions d, time T, maxsteps, batch size, learning rate
Initialize $X_0 = \xi$
Initialize dW a random normal variable with standard deviation $\sqrt{T/N}$
Initialize Y_0 a uniform random variable from $[-1, 1]^d$ as an initial weight θ_*
Initialize Z_0, A_0 as uniform random variables from $[-0.1, 0.1]^d$
Initialize $i = 0$ **while** *step* $<$ *max_step* **do**
  for $i <$ *N - 1* **do**
    Set Y = Y + f(i*T/N,X,Y,Z_{i-1})*(T/N) + Z_{i-1}*dW
    Parameterize the neural network Z_i with weights $(W_i, b_i) = \theta_i$
    Update $X = X + b(t, X)$(T/N)+$\sigma(X)$*dW
    Pick a new value from standard normal for dW
    Set $i = i + 1$
  end
  Set Y = Y + f(i*T/N,X,Y,Z_{i-1})*(T/N) + Z_{i-1}*dW
  Update $X = X + b(t, X)$*(T/N)+$\sigma(X)$*dW
  Calculate loss $|Y - g(X)|^2$ Update weights using Adam optimization.
end
Return θ_*

This algorithm manages to avoid the curse of dimensionality that the traditional Monte Carlo fall into, by avoiding the need to compute an expected value for Z using the results of previous values of Y as in the Monte Carlo algorithm. Instead by taking an estimate of Z using a neural network the SDE X can be simulated along with the BSDE Y, with the results from the simulation of X being used to immediately update Y. The Monte Carlo algorithm required the complete simulation of SDEs before even beginning to simulate the BSDE portion, and it is this restriction that causes the exponential computation time as the dimensionality of the problem increases.

3.3.1 Deep BSDE Extension

Naito and Yamada - A priori estimate using Weak- Approximation

In their paper [26], Naito and Yamada improve the speed of convergence and reduce the error rate of the Deep BSDE algorithm. This is achieved through the use of a priori knowledge.

Naita and Yamada consider a BSDE of the form

$$X_s = x_0 + \int_0^s V_0(X_r)dr + \sum_{i=1}^{d} \int_0^s V_i(X_r)dW_r^i \tag{3.5}$$

$$Y_s = g(X_T) + \int_s^T f(X_r, Y_r, Z_r)dr - \sum_{i=1}^{d} \int_s^T Z_r^i dW_r^i. \tag{3.6}$$

Naito and Yamada suggest using a "weak approximation" of the BSDE Y by applying Malliavan calculus and Stochastic Taylor expansion to find an a priori approximation of the BSDE. This approximation does not take into account any non-linearity of the BSDE, and the Deep BSDE method is then used to complete the non-linearity once the initial guess of the value Y_0 has been done. These improvements are relatively incremental, but nonetheless useful. By reducing the amount of information that the neural network needs to approximate by using approximations of Malliavan calculus, the neural network takes less time to train and produces more accurate results than otherwise would be the case. More details on weak-approximations of BSDEs and their derivation can be found in [25].

Recall that in the Deep BSDE solver, a forward discretization is used for the BSDE. In particular, a *guess* was made about the initial value Y_0. In Naito and Yamada's scheme, Y_0 is approximated using a priori knowledge, by taking the estimation

$$Y_0^{(0)} \approx E[g(\hat{X}_T^{x_0})\pi_T^{x_0}].$$

Here, $\hat{X}_T^x$ is a Gaussian approximation,

$$\hat{X}_T^x = x + V_0(x)t + \sum_{i=1}^{d} + \sum_{i=1}^{d} V_i(x)W_t^i.$$

and $\pi_T^{x_0}$ is a relatively complex Malliavan weight that provides a weak-approximation of the BSDE, and is given by

$$\begin{aligned}
\pi_t^x =& 1 + \sum_{\ell,j=1}^{N} \sum_{i_1,i_2=0}^{d} \sum_{i_3=1}^{d} \frac{1}{2t} L_{i_1} V_{i_2}^{\ell}(x)(A^{-1})^{\ell j}(x) V_{i_3}^{j}(x) \\
&\times \left(W_t^{i_1} W_t^{i_2} W_t^{i_3} - t W_t^{i_3} \mathbf{1}_{\{i_1=i_2\neq 0\}} - t W_t^{i_1} \mathbf{1}_{\{i_2=i_3\neq 0\}} - t W_t^{i_2} \mathbf{1}_{\{i_1=i_3\neq 0\}} \right) \\
&+ \sum_{\ell_1,\ell_2,u_1,u_2=1}^{N} \sum_{i_1,i_2=1}^{d} \sum_{k_1,k_2=1}^{d} \frac{1}{4} L_{i_1} V_{i_2}^{\ell_1}(x) L_{i_1} V_{i_2}^{\ell_2}(x) \\
&\times (A^{-1})^{\ell_1 u_1}(x) V_{k_1}^{u_1}(x) (A^{-1})^{\ell_2 u_2}(x) V_{k_2}^{u_2}(x) \left(W_t^{k_1} W_t^{k_2} - t \mathbf{1}_{\{k_1=k_2\neq 0\}} \right),
\end{aligned}$$

In the above weight, the matrix components for A are given by $A^{ij}(x) = \sum_{k=1}^{d} V_k^i(x) V_k^j(x)$, and the L_i's given by

$$L_i f(x) = \sum_{k=1}^{N} V_i^k(x) \frac{\partial f}{\partial x_k}(x) \tag{3.7}$$

$$L_0 f(x) = \sum_{k=1}^{N} V_0^k(x) \frac{\partial f}{\partial x_k}(x) + \frac{1}{2} \sum_{k,l=1}^{N} \sum_{j=1}^{d} V_j^k(x) V_j^l(x) \frac{\partial^2 f}{\partial x_k \partial x_l}(x), \tag{3.8}$$

Under the assumption that the functions V_i are smooth, and each V_i is a function from $\mathbb{R}^N \to \mathbb{R}^N$.

A further improvement in the algorithm is found by an a priori approximation of the component $Z_s = \Psi(s, X_s)$. We then decompose $\Psi(s, X_s)$ into two parts,

$$\Psi(s, X_s) = \Psi^{AS}(s, X_s) + \hat{\Psi}(s, X_s)$$

where here Ψ^{AS} is an approximation of Ψ that is found asymptotically using results

from Malliavan calculus, and $\hat{\Psi}$ is found by means of a neural network. Here,

$$\Psi_i^{AS}(t,x) = E\left[g(\bar{X}^x_{T-t})\frac{W^i_T - W^i_t}{T-t}\right]$$

Once again, the cost function minimized here is identical to the one used by Jentzen et al. [39].

In contrast, [17] provides two separate algorithms for using neural networks to solve an FBSDE system more traditionally, starting from the terminal time $t = T$ and finding an approximate solution for the BSDE at $t = 0$.

3.4 Neural Networks for Solving PDEs

In recent years, work has been done to find numerical solutions to PDEs through the use of Artificial Neural Networks (ANNs) in a similar manner to how BSDEs can be solved using the Deep BSDE method. Several methods have been developed to solve PDEs using artificial neural networks. This section reviews the Deep PDE Method and provides an example comparing the use of Deep PDE for solving a PDE to the Deep BSDE algorithm. In general, the Deep BSDE algorithm is significantly less computationally intensive but is limited in being able to solve for only one time value, where as the Deep PDE method trains a single ANN in order to approximate the solution for a given PDE

Consider a parabolic PDE of the form

$$\partial_t u(t,x) + \mathcal{L}u(t,x) = 0 \tag{3.9}$$
$$u(0,x) = u_0(x) \qquad \text{for some } x \in \Omega, \tag{3.10}$$
$$u(t,x) = g(t,x) \qquad \text{for some } x \in [0,T] \times \partial\Omega. \tag{3.11}$$

Here $\mathcal{L}$ is a second-order parabolic operator that is not necessaily linear.

The Deep PDE Method is a method for solving PDEs that limits the problem of the

curse of dimensionality by using a mesh free method along with an artificial neural network in order to approximate the solution to a given PDE or systems of PDEs. We summarize the algorithm for the method and briefly discuss its effectiveness.

Algorithm

The Deep PDE Method uses a neural network in order to approximate a solution u to the given PDE. While several other methods using deep learning techniques to solve PDEs exists, the Deep PDE method attempts to reduce the effect of an increase on the dimension of the underlying space, by avoiding the use of a mesh setup. Instead, points are chosen from Ω based on a probability distribution, which are then fed into the forward propagated into a given neural network. The neural network is updated using stochastic gradient descent in an attempt to minimize the following objective function:

$$J(f) = ||\partial_t f + \mathcal{L}f||^2_{L^2([0,T]\times\Omega)} + ||f - g||^2_{L^2([0,T]\times\partial\Omega)} + ||f(0,\cdot) - u_0||_{L^2(\Omega)} \tag{3.12}$$

By minimizing the above function, the difference between the time derivative and the quasi-linear portion of the PDE is reduced, and in addition the boundary conditions and starting conditions are approximated well by the neural network. There are two questions to deal with: does the use of stochastic gradient descent converge to an approximate solution given this objective function, and to what extent can this process be improved if so? We discuss convergence in Theorem 3.4.1. Improvements to the optimization method could take the use of the Adam optimizer, as suggested by Jentzen et al. for training the Deep BSDE method, or the use of other optimizers. However, we review the structure suggested in [38] here.

The general structure of the neural network used by Sirignano et al. is given by the following set of tensors. At each la yer, $\ell = 1, \ldots, \ell$ the following tensors are calculated.

$$S^1 = \sigma(W^1\mathbf{x} + \mathbf{b}) \tag{3.13}$$
$$Z^\ell = \sigma(U^{z,\ell}\mathbf{x} + W^{z,\ell}S^\ell + \mathbf{b}^{z,\ell}) \tag{3.14}$$
$$G^\ell = \sigma(U^{g,\ell}\mathbf{x} + W^{g,\ell}S^1 + \mathbf{b}^{g,\ell}) \tag{3.15}$$
$$R^\ell = \sigma(U^{r,\ell}\mathbf{x} + W^{r,\ell}S^\ell + \mathbf{b}^{r,\ell}) \tag{3.16}$$
$$H^\ell = \sigma(U^{h,\ell}\mathbf{x} + W^{h,\ell}(S^\ell \odot R^\ell) + \mathbf{b}^{h,\ell}) \tag{3.17}$$
$$\tag{3.18}$$

These tensors then allow the calculation of the following tensor, which begins the next layer.

$$S^{\ell+1} = (1 - G^\ell) \odot H^\ell + Z^\ell \odot S^\ell$$

Note that $U^{\cdot,\ell}$ and $W^{\cdot,\ell}$ are weight matrices, which differ per tensor. $\mathbf{x}$ is the input vector and $\mathbf{b}^{\cdot,\ell}$ is a bias vector.

It is assumed that the terminal or initial condition $g(x)$ of the PDE is continues and that its first derivative is bounded in the domain, the domain Ω is a bounded open subset of $\mathbb{R}^d$ with the boundary $\partial\Omega \in C^2$ and that the neural network f^n for each learning step n is continuous, and $(f^n)_{n\in\mathbb{N}} \in L^2(\Omega_T)$. The theorem below shows that for appropriate conditions on the PDE, there are appropriate parameters for the above neural network which minimizes the objective function J.

Theorem 3.4.1 *Define*

$$\mathcal{C}^n(\psi) = \left\{ \zeta(t,x) : \mathbb{R}^{1+d} \mapsto \mathbb{R} : \zeta(t,x) = \sum_{i=1}^{n} \beta_i \psi\left(\alpha_{1,i}t + \sum_{j=1}^{d} \alpha_{j,i}x_j + c_j\right) \right\}.$$

where $\theta = (\beta_1, \ldots, \beta_n, \alpha_{1,1}, \ldots, \alpha_{d,n}, c_1, \ldots, c_n) \in \mathbb{R}^{2n+n(1+d)}$ *are the possible parameters. Assume* $\psi \in C^2(\mathbb{R}^d)$ *is a bounded, non-constant function. Furthermore define* $\mathcal{C}(\psi) = \bigcup_{n\geq 1} \mathcal{C}^n(\psi)$. *Assume the underlying space* Ω_T *is compact and the probability measures* ν_1, ν_2, ν_3 *have supports contained in* Ω_T, Ω *and* $\partial\Omega_T$ *respectively.*

Assume that $f(t,x,u,\nabla_x u)$ is locally Lipschitz and that the PDE (3.5)-(3.7) has a unique classical solution $u(t,x)$ such that

$$u(t,x) \in \mathcal{C}(\Omega_t) \bigcup \mathcal{C}^{1+\eta/2,2+\eta(\Omega_T)}$$

where $\eta \in (0,1)$ and $\sup_{(t,x)\in\Omega_T} \sum_{k=1}^{2} |\nabla_x^{(k)} u(t,x)| < \infty$.

Then there exists for every $\varepsilon > 0$ a positive $K > 0$ (which can depend on u) such that there is an $h \in \mathcal{C}(\psi)$ which satisfies $J(f) \leq K\varepsilon$.

This theorem that we can satisfy the loss function for the method. In section 4.3 we implement a modified objective function which uses only part of this given objective function. This theorem extends to our chosen objective function, as any function f that satisifies this theorem will also satisfy our objective function.

Theorem 3.4.2 *Assume the previous assumption on the PDE holds, then the PDE (3.5)-(3.7) has a unique bounded solution. In addition, h^n converges to u, the unique solution to (3.5)-(3.7) strong in $L^p(\Omega_T)$ for every $p < 2$. If the sequence $\{h^n(t,x)\}_{n\in\mathbb{N}}$ is uniformly bounded in n and equicontinuous, then the convergence to the solution u is uniform in the domain Ω_T.*

The above theorem allows us to know that with enough training steps the Deep PDE method converges to a solution of the (3.5)-(3.7). The proofs of these theorems are relatively long, and are available in Appendix A of [38].

There are several other ways of applying neural networks to solve PDEs, but also to learn the PDE itself. Work in this area of machine learning to approximate PDEs is relatively new. An example of recent work in this area is in [37]. This paper discusses the implementation of a Deep Learning PDE Method (DPM), This method splits PDEs into known a priori components and unknown components, specifically the class represented by the following equation:

$$\frac{\partial u}{\partial t}(t,x) = f_\nu(u(t,x), u_x(t,x), u_{xx}(t,x)) + h_\theta(u(t,x), u_x(t,x)) + h_\theta(u(t,x), u_x(t,x), u_{xx}(t,x)),$$

where here we are using the general notation for partial derivatives in space under the study of PDEs. Specifically, $u(t,x) : \Omega \to \mathbb{R}^d$ is the solution to the PDE, where Ω is some arbitrary subset of $\mathbb{R}^q$, and u_x, u_{xx} represent its first and second spatial derivatives. In addition, the terms ν and θ here are considered to be parameters that vary based on the restrictions required for the PDE.

Now, this representation splits the PDE into two separate function: One represents the known equations for which we have a priori representations, denoted by f_ν and the second, h_θ represents the unknown component of the background physics that needs to be found. A deep learning algorithm is used to find an approximation for h_θ which allows an overall approximation of the solution. Thus, h_θ takes the form

$$h_\theta : \mathbb{R}^{(2q+1)} \to \mathbb{R}^d$$

and θ will describe the neural network parameters such as number of hidden layers, number of nodes, and weight matrices for each layer.

The example used throughout the paper is the Navier-Stokes equations that are used to simulate the physics of water flows. The training of the neural network is again done by using stochastic gradient descent to find a minimum to a cost function (referred to in [37] as an objective function). Letting $V(t,x;\nu)$ represent trusted data that can be used to model the underlying physics, the objective function here is given by

$$L(\theta) = \sum_{m=1}^{M} \sum_{n=1}^{N_t} \int_\Omega ||u(t_n,x;\nu_m) - V(t_n,x;\nu_m)||dx,$$

recalling that the solution of the PDE is dependent upon the known parameters ν. This method therefore is an attempt to learn PDEs based on known trusted observations as opposed to simulated data.

Chapter 4

Comparison of Methods

Given that we now have several methods of numerically solving PDEs, either directly or via finding numerical solutions to their corresponding BSDEs, it would be interesting to see what methods are more effective than others. This comparison is somewhat difficult however. For instance, although the nonlinear Feynman-Kac formula gives us the ability to approximate solutions to PDEs using the Deep BSDE method, it is important to note that this method only gives a result for a single value in $[0, T] \times \mathbb{R}^d$, where as the Deep PDE method uses a neural network to approximate a complete function which solves a given PDE. That is, the neural network in the case of the Deep PDE Method takes as input points in $[0, T] \times \mathbb{R}^d$ and provides as output the value of the solution function at that point, whereas the Deep BSDE method trains a neural network to find a solution to a BSDE, which happens to provide the value of the solution of the corresponding PDE at a certain point. Therefore, while the Deep BSDE method is significantly less computationally expensive, it also provides significantly less information. To what extent does this matter in applications where we want to find various values of the corresponding solution function for a given PDE? This chapter attempts to look into this question and provide an answer. For this, we need to first look at implementations of both methods and compare their accuracy for a single point, and also see at what point, if any, the use of one method over the other is more practical. In the first subsection we compare finding nunmerical solutions to BSDEs between using Monte-Carlo algorithms and using the Deep BSDE algorithm. In the second section we give a brief overview of differences between using the Deep

BSDE algorithm to find solutions to their corresponding PDEs and using a neural network to find a numerical solution directly using the Deep PDE method.[1]

4.1 Monte-Carlo numerical solutions when compared with Deep BSDE

Earlier we reviewed much of the literature of techniques to solve BSDEs. The earlier techniques, characterized by works such as [11] [3] [5] utilized Monte-Carlo simulations which simply simulated multiple SDEs and then used a projection onto a particular basis in order to estimate the resulting initial value of the BSDE. In [11] one example provided was the following:

Consider an investor interested in the price of an option on a stock or other financial instrument in an incomplete market. We consider an underlying stock X_t defined by

$$dX_t = \mu dt + \sigma dW_t$$

where W_t is a standard Brownian motion, and μ and σ are constants. In this market, there are different rates for lending and borrowing, denoted by r and R respectively. It is of interest to figure out what the fair price of a straddle option denoted by Y may be at time $t = 0$. Let π^L denote the investment that is loaned out with rate r, π^B denote the amount borrowed according to rate R and π^x denote the amount invested in the stock X_t. The option then will be given by $Y_t = \pi_t^L + \pi_t^x + \pi_t^B$ and the change in the investment Y_t can be modelled by the following equation:

$$\begin{aligned} dY_t &= r\pi_t^L dt + \pi_t^x dX_t + R\pi_t^B dt \\ &= r\pi_t^L dt + \pi_t^x \mu dt + \pi_t^x \sigma dW_t + \pi_t^B R dt \\ &= (\pi_t^L + \pi_t^x + \pi_t^B) r dt + \pi_t^x(\mu - r) + \pi_t^B(R - r)dt + \pi_t^x \sigma dW_t \end{aligned}$$

[1] All simulations were computed on CanadaCompute Cedar nodes.

Set $Z_t = \pi_t^x \sigma$, and notice that

$$\pi_t^B = ((\pi_t^L + \pi_t^x + \pi_t^B) - (\pi_t^L + \pi_t^x)) = (Y_t - (\pi_t^L + \pi_t^x))$$

Now, there are strategies where loaning and borrowing at the same time are possible, however if the borrowing rate R is greater than r, then this is never optimal, since any investment being loaned out will decrease the value of the investment faster than using that money to reduce π_t^B. Therefore, whenever $\pi_t^B > 0$, it must be the case that $\pi_t^L = 0$ optimally. Thus,

$$\pi_t^B = (Y_t - \pi_t^x) = (Y_t - \frac{Z_t}{\sigma})$$

and since $\pi_t^B = 0$ whenever $(Y_t + \frac{Z_t}{\sigma}) >= 0$ we must have the following equation

$$dY_t = rY_t + \frac{\mu - r}{\sigma} Z_t - (R - r)(Y_t - \frac{Z_t}{\sigma})_- dt + Z_t dW_t \tag{4.1}$$

Where

$$(Y_t - \frac{Z_t}{\sigma})_- := \begin{cases} -(Y_t - \frac{Z_t}{\sigma}) & \text{if } (Y_t - \frac{Z_t}{\sigma} <= 0 \\ 0 & \text{otherwise} \end{cases}$$

If we assign the terminal condition $Y_T = |X_T - K|$ then Y_t represents the fair price of a straddle option in an incomplete market. Recall that a saddle option is both a put and call option purchased on the same financial asset with the same terminal time T. Saddle options are useful because they can be used when an investor believes that the price of an asset is going to increase or decrease significantly in the future, but is not sure in which direction the price of the asset will go. Lenders can offer these options at a fee to ensure that they do not lose money on the trade regardless of the later price of the asset. This leads to the question of what is a fair price of this saddle option in an incomplete market.

This is an adaptation of the traditional Black-Scholes model for the fair pricing of a straddle option. We adopt this example and compare the performance between the Monte Carlo methods given by [11] and the newer techniques utilizing deep learning

and neural networks found in [2]

For the experiments, the constants $\sigma = 0.2$, $R = 0.06$, $r = 0.01$, $T = 2$, $K = 100$, $\mu = 0.05$ and $X_0 = 100$ where used. We discretize the BSDE for t he Monte-Carlo method and Deep BSDE method as described in their sections, where we take a partition of $[0, T]$ with time grid given by $\{t_i\}_{i=0}^{N}$ such that $t_0 = 0$ and $t_N = T$.

For the Monte-Carlo algorithm, the following basis is used for the subspace of $L^2(\mathbb{R})$:

$$e_1 = |x - K|$$
$$e_k = (x - X_0)^{k-2}, \qquad 2 \leq k \leq \kappa$$

We also implemented these same equations using the Deep BSDE algorithm. Here instead of a choice of basis, the loss function was chosen to be $f(x) = |x-K|$, and the value at the end of the simulation was taken to be the mean of the dimensions of the resulting vector. The one main question is to what extent does the Deep BSDE algorithm provide optimization benefits over traditional Monte-Carlo simulations when given many dimensions to work with.

When looking at these algorithms it is important to note that they are not directly comparable. That being said, it is a simple argument to make that at the very least in this case, the Deep BSDE method is superior.

In the case of the Deep BSDE algorithm, the forward discretization 3.1. For the Monte-Carlo algorithm, the discretization detailed in Chapter 2 is used. The primary tunable parameters for the Monte-Carlo algorithm is the number of simulations, L and the number of parition points N. For the Deep BSDE algorithm, in addition to the same number of partition points N, changes can be made to the exponential decay rate, the number of decay steps (how often the exponential decay is applied to the learning rate), the batch size per learning step, and the number of learning steps. We detail some of our tuning choices for the Deep BSDE algorithm below, and compare the Deep BSDE algorithm with batch size 100 against the Monte-Carlo algorithm using 1000, 10000, and 100000 simulations, with a batch size of 100.

Tuning the Deep BSDE algorithm required some patience. The initial learning rate α was started at 0.06. We found that values as high as 0.08 would work, however

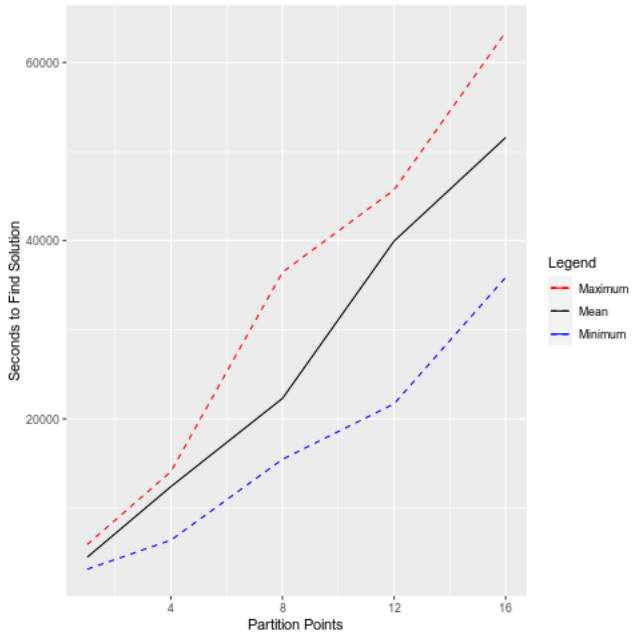

Figure 4.1: performance of the Monte-Carlo simulation algorithm over different numbers of partition points. an average of 10 runs per point. the algorithm complexity is linear in regard to the number of partition points.

a learning rate of 0.09 or higher resulted in the chosen Y_0 value quickly becoming infinite. In addition, an exponential decay was applied to the learning rate. This was a necessity as otherwise the variance of solutions given by the algorithm greatly increased. In addition, 20000 learning steps was rather arbitrarily chosen. This was to make sure the algorithm had enough time to converge properly given the initial learning rate.

For the implemented exponential decay on the learning rate, a decay step occurred every 4000 steps. Learning decay rate was chosen to be 0.8, although there are several decay rates that can be chosen here to minimize variance (see Figure 4.2). The Deep BSDE algorithm provided similar estimates to the Monte-Carlo algorithm, at a fraction of the computational time required. In addition, the Deep BSDE algorithm is significantly faster as you increase the number of partition points to be calculated.

The Monte-Carlo algorithm suffers from the curse of dimensionality, in the sense that increasing the number of dimensions substantially increases the numbers of computations required and thus it becomes infeasible to compute the solutions for larger

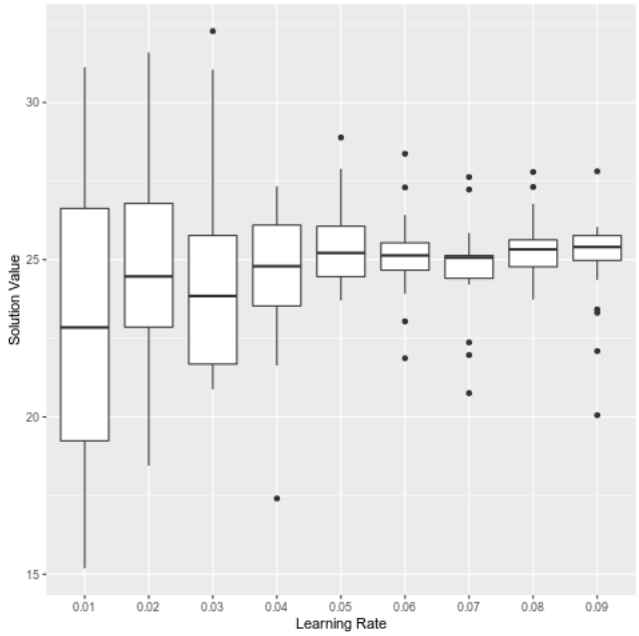

Figure 4.2: Boxplot of solutions given at various exponential decay rates for the learning rate of the Deep BSDE algorithm. Variance of solutions decreases as the learning rate increases, suggesting that the Deep BSDE algorithm is converging and that learning rates 0.06-0.09 give reasonable solutions.

problems (see Figure 4.1). This is because the algorithm has exponential complexity in time with respect to the dimension of the problem. Thus computing higher dimensional examples with the Deep BSDE algorithm should be particularly more efficient than the use of traditional Monte-Carlo algorithms, without sacrificing accuracy. The trade off is of course that tuning needs to be done to the meta parameters of the model which can take time that is not needed for the Monte-Carlo implementation. To improve accuracy of the Monte-Carlo simulation, one need only increase the number of simulated paths.

We assess to what extent the number of simulated paths can be reduced in the Monte-Carlo algorithm relative to a decrease in accuracy and how that accuracy difference compares to the Deep BSDE algorithm. At approximately 1000 paths simulated, we observe a similar amount of computational time for solving $N = 4$ partition points as with our tuned Deep BSDE algorithm utilizing a batch size of 100. However, this greatly increases the relative standard error to be on par with the Deep BSDE algorithm, and also once again has a linear growth in computational time as the partition size is decreased using the same T. Thus the Monte Carlo algorithm can be tuned to perform similarly to the Deep BSDE method at a low number of partition points, however this does not eliminate its weakness of performing poorly as the number of partition points increases compared to the Deep BSDE algorithm. However, it does provide more information in so far as the requisite simulations for the Monte-Carlo algorithm allows for approximating the stochastic process Y_{t_k} at each time t_k as opposed to simply the expected value. For this reason, we suggest that the least-squares Monte Carlo algorithm is best suited to solving low-dimensional problems where high computational time is available and significant accuracy is required, as the algorithm provides significantly more accurate approximations compared to the Deep BSDE as the number of simulated paths increases to 100000 or more.

The Monte-Carlo algorithm did converge to similar solution values as the Deep BSDE algorithm. Notice that in 4.1 for various different number of partition points, the simulated values slowly converge to some value. Running the Monte-Carlo algorithm 100 times at 100000 simulations for 16 partitions and taking the average of the solution gives an approximate expected value of Y_0 to be 24.6399. The absolute value of the difference in value of the Monte-Carlo algorithm vs this estimated solution is available in 4.1. Overall, both algorithms have their own advantages, with the Monte-Carlo

Algorithm	Simulations/Batch Size	Average Running Time (Seconds)
Monte-Carlo	1000	53.302
Monte-Carlo	10000	539.66
Monte-Carlo	100000	51569.6
Deep BSDE	100	87.9
Deep BSDE	1000	189.52

Figure 4.3: System running times via the time unix command for each method, calculated on Canada Computer Cedar nodes. Each method was evaluated at 16 partition points. The Deep BSDE gives an approximation of the expected value of Y_0 in orders of magnitude less time than the Monte-Carlo algorithm. The Monte-Carlo algorithm takes a comparable amount of computational speed at 1000 simulations, but provides less accuracy on average (see Figure 4.5)

algorithm offering slower performance in exchange for a lack of initial tuning and more total information, while the Deep BSDE algorithm offers substantial performance boosts but does not provide information for recovering probability densities.

4.2 Deep BSDE Compared with Other Numerical Methods

One comparison to be done is to compare the speed and accuracy of older, Monte-Carlo methods with that of the Deep BSDE methods. It is clear that, as the dimension of the space the BSDE is operating over increases in dimension, the Deep BSDE method appears to remain a much better approximator, in the sense that the growth of the Monte-Carlo solver in terms of calculation time is exponential. This is the "curse of dimensionality" that the Deep BSDE solver manages to avoid, and its main advantage to use. However, even in lower dimensions it appears that it can be quite reasonably accurate and significantly faster than the Monte Carlo methods. This is largely because accuracy in the Monte-Carlo methods used by Gobet or Bender requires a significant number of simulations to achieve comparable accuracy to the Deep BSDE method.

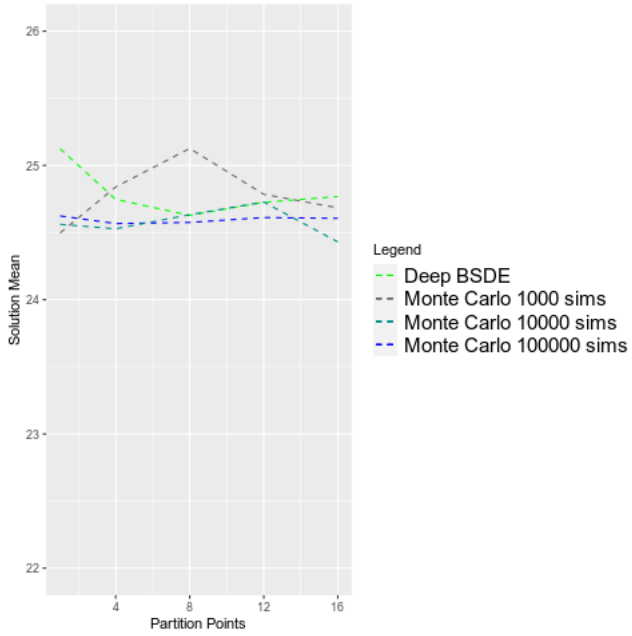

Figure 4.4: Average solution values for 10 runs of each algorithm per number of partitions: Deep BSDE vs Monte Carlo methods. Deep BSDE algorithm here using a batch size of 1000.

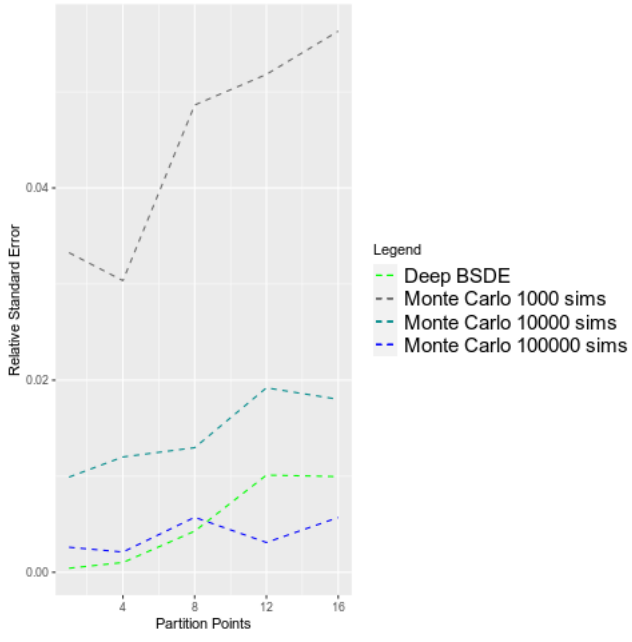

Figure 4.5: Relative Standard Error comparison between Deep BSDE algorithm and Least-Squares Monte-Carlo for various numbers of partition points. 10 samples per partition point per algorithm.

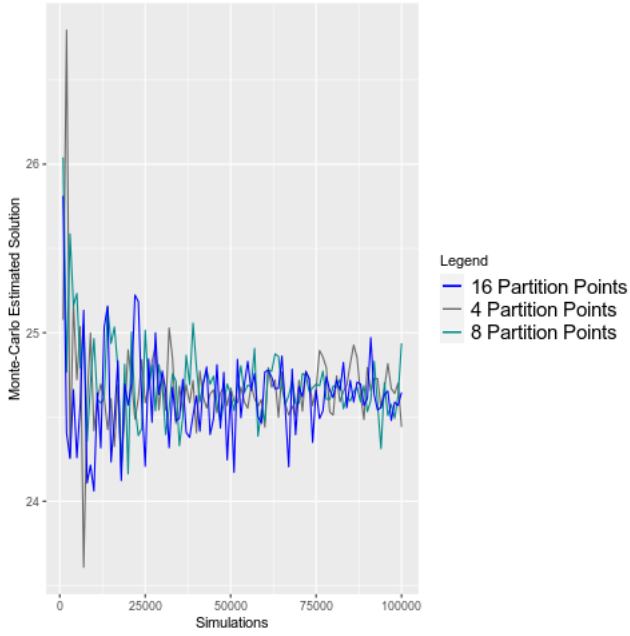

Figure 4.6: Solutions of Monte-Carlo algorithm as the number of simulations increases.

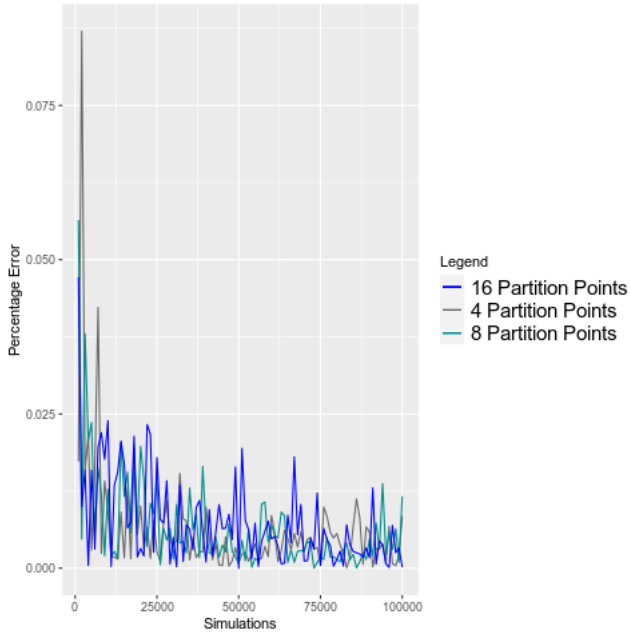

Figure 4.7: Percentage error of the solution from the previous figure from the estimated solution 24.6399 calculated from the average of 100 runs of the Monte-Carlo simulation with 16 partition points at 100000 sims.

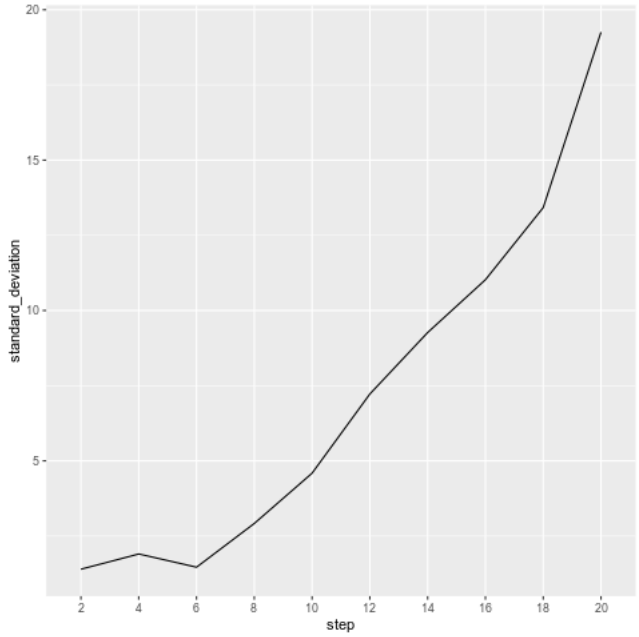

Figure 4.8: Standard deviation of simulated values of Y0 as a function of the number of decay steps. Less is better. This supports taking between 2-6 decay steps for the algorithm.

4.3 Deep BSDE and Deep PDE method Comparison

In this section we give an example of the use of the Deep PDE method discussed in the previous chapter. For the Deep BSDE method, a Monte-Carlo simulation of the underlying SDE is done multiple times, and in parallel the coupled SDE is simulated based on an initial estimate of the starting value. The main difference is that the Deep PDE Method learns the dynamics of a given PDE at any given point $(t, x) \in [0, T] \times \mathbb{R}^d$, whereas the Deep BSDE method is designed to find the initial condition of a point in $\mathbb{R}^d$ from a terminal condition (or vice-versa). While they both use neural networks to find solutions to PDEs, they do so in very different ways

4.3.1 Example: Allen-Cahn PDE

Consider the parabolic PDE of the form

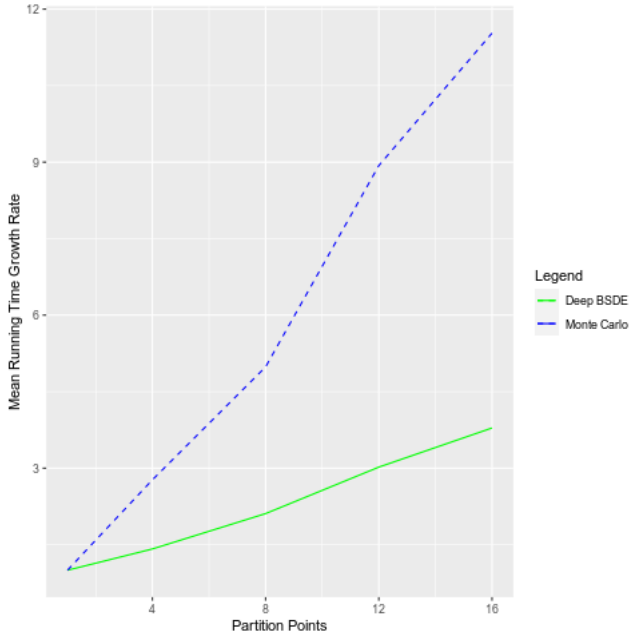

Figure 4.9: Relative average growth in computationinal time as a function of N. Each algorithm is normalized to the average computational time for $N = 1$. Note that the Deep BSDE algorithm grows substantially slower than the traditional Monte Carlo algorithm.

$$\partial_t u + \Delta_x u + u - u^3 = 0$$

$$t \in [0, T], \quad x \in [-1, 1]^d$$

where d is the number of dimensions and with terminal condition given by

$$g(x) = \frac{1}{2(1 + \frac{2}{5}x^2)}$$

.

We initialize the Deep PDE method using the layer structure given in 3.13, it is suggested in [38] to use 4 hidden layers of said structure, with each layer having approximately 5 sublayers consisting of 50 nodes.

Initialization of the weights of each layer was originally done with standard normal variables with mean 0, standard deviation of 1. However, this resulted in what is known as gradient explosion - that is, during the calculation of terms such as $\partial_t u$ or $\Delta_x u$ the gradient would quickly grow towards infinity, and would thus be useless. It is for this reason that initialization of weights eventually took values from a uniform random distribution for each weight, with the interval on which values were taken given by

$$[-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}]$$

where the values n_i and n_{i+1} are simply the dimension of the input to the sublayer, and the size of the output of the sublayer.

The differences between these algorithms here are rather stark. The Deep BSDE algorithm has a significantly simpler ANN than does the Deep PDE method. For this reason, training of the Deep PDE method takes significantly more computational resources. In addition, the Deep PDE method is attempting to be able to simulate the dynamics for all points in $[0, T] \times \mathbb{R}^d$ once trained, whereas the Deep BSDE method is simply finding the Y value of the given BSDE at time 0, which gives the solution to the BSDE at the given x value. Given the need to remember complex dynamics, the Deep PDE method method requires significantly longer training than the Deep

Dimensions	Y_0 Deep BSDE	$u(0, \mathbf{0})$ Deep PDE
1	0.5835275	0.4669369
10	0.4118325	0.31361356
20	0.3057440	0.25555387
30	0.2408965	0.11842176
40	0.1936964	0.23099062

Figure 4.10: Solution Results from the Deep BSDE method proposed by Jentzen et al. from [2] and the Deep PDE Method from Sigiriano et al. in [38] after some 80 epochs of 10 samples per, with $x = 0, t = 0$ added to each epoch for training for the Deep PDE method.

BSDE method for providing a similar answer. On the other hand, the Deep PDE method method once trained provides solutions to more than a single time and space value, which the Deep BSDE method forgoes.

Note that in the Deep BSDE method, the space value solved for is the initial value X_0 of the SDE

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

For the Allen-Cahn PDE, this value is taken to be zero and thus the Deep BSDE method is equivalent to querying the ANN trained via the Deep PDE method on the value of the 0 vector. However, one could query many other space-time points utilizing the Deep PDE Method and thus, while it takes significantly longer to train, there are substantial benefits to doing so for other applications.

Finally, it is hard to make a direct comparison between the Deep BSDE method and the Deep PDE Method due to their structure. The Deep BSDE method trains a neural network to find an initial value to a BSDE by utilizing a merged PDE/BSDE formulation but importantly, the PDE does not have any boundary conditions: the PDE is essentially evaluated on all of $\mathbb{R}^d$. Unfortunately, the Deep PDE method requires a boundary condition. This boundary condition is utilized in the objective function above. Specifically, there is no $g(x)$ function that we can associate with a particular boundary that would allow the Deep PDE method to replicate the results from the Deep BSDE method. Work should be done in this direction, to see if the Deep BSDE method can be updated to work with specific boundary conditions on a

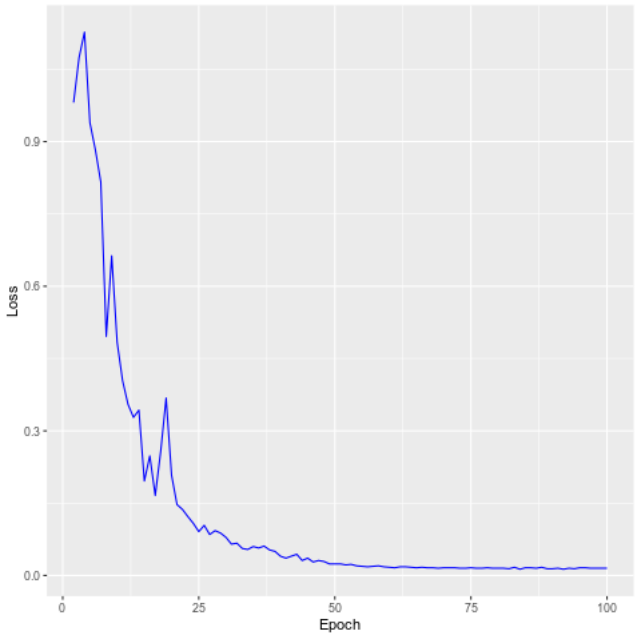

Figure 4.11: Loss results on a per epoch basis for the Deep PDE method with L = 20 nodes, 1-dimension

bounded domain.

Chapter 5

Suggested Future Research

Our comparison between Jentzen et al.'s Deep BSDE algorithm and Bender et al.'s least-squares Monte Carlo algorithm was suggestive that both algorithms still have a use in the modern day. However, the Deep BSDE algorithms has seen several potential improvements suggested that were not incorporated into our implementation. For instance, Naito and Yamada's a priori improvement on the initial estimate of Y_0 has been shown in their paper to significantly decrease learning time required for a good approximation as well as error in the final result. Furthermore, there are several other algorithms that could be implemented and tested against an improved Deep BSDE method. Other BSDEs could also potentially be used in comparisons, and different optimizers could be used for the Deep BSDE algorithm.

In addition, our attempt to compare the Deep PDE Method to the use of the Deep BSDE and Monte-Carlo methods for solving PDEs revealed an interesting problem - namely, that the Deep PDE Method only properly functions with well defined border conditions and bounded domains. Meanwhile, the Deep BSDE method solves PDEs over the entire space $\mathbb{R}^d$ and thus there is no border to define. Future attempts should be made to reconcile these methods. Due to how the Deep PDE Method uses generated samples of random points, it does not seem possible to change this method to extend to the entire domain of the solution of the PDE. However, it is likely that the Deep BSDE algorithm could be updated to do so by limiting the domain it operates on and training the network to find the appropriate Y_0 value.

In order to do so, one could consider an algorithm that implements the Deep BSDE method at various randomly selected points in space similar to how points are randomly selected in the Deep PDE method. This would allow simulating the dynamics of a PDE over a space grid, but not a time-grid. Once again however the dynamics of the PDE on the boundary will need to be determined and it is not clear how to implement that in the current Deep BSDE algorithm. Other avenues of research could be to combine the Primal Dual Deep BSDE algorithm with the a priori knowledge improvements from [26] which should in theory greatly improve the algorithms efficiency if properly implemented.

Chapter 6

Conclusions

In this thesis we reviewed some of the theory related to BSDEs before comparing several algorithms that either find solutions for given BSDEs, or solve PDEs associated with those BSDEs. We had a few novel contributions: we gave an explicit derivation for the BSDE associated with the fair price of an option in an incomplete market with differing lending and borrowing rates. We directly compared the Least-Squares Monte-Carlo method for solving BSDEs to the Deep BSDE method. We evaluated the solutions of the Deep PDE method with those of the Deep BSDE method for an associated PDE-BSDE pair, and modified the Deep PDE algorithm in an attempt to allow it to work on the same domain as the Deep BSDE algorithm for the given problem.

We provided the proof that the BSDE in Section 4.1 yields the price of the given financial options, in an incomplete market with differing lending and borrowing rates. Our comparison of the two algorithms in this section is also novel to the best of our knowledge, in particular the comparison of running times to accuracy. While some numerical estimates have been provided in [4] for the Monte-Carlo algorithm for this example for 100000 simulations, we add additional numerical results for the Monte-Carlo algorithm and test its performance compared to the newer Deep BSDE method. By comparing the results of the two algorithms on this example, we found that the Monte Carlo algorithm was able to provide extremely accurate approximations when given enough computational time, while the Deep BSDE algorithm was able to find solutions as accurately with a smaller amount of computational time.

In exchange, the Deep BSDE algorithm requires hyper-parameter tuning wherein a priori knowledge of the solution is useful for ensuring proper tuning. Finally, unlike the Deep BSDE algorithm, the Monte-Carlo algorithm is able to fully reproduce the probability distribution of the BSDE at any point of calculated time.

We also reviewed a modern technique for solving PDEs, the Deep PDE method which also utilizes a neural network to solve non-linear PDEs. To our knowledge, no direct comparisons of these algorithms have been made, although the Deep BSDE method has been evaluated against the Allen-Cahn example in [15], results are provided only for 20 dimensions in the underlying space. We provide solution values for the Deep BSDE method on this example for 1, 10, 20, 30, and 40 dimensions. The Deep PDE method differs significantly from the BSDE methods wherein the given approximate solution is itself a Neural Network trained to approximate the solution $u(t, x)$ to any given PDE. This allowed the fully trained ANN to approximate any point $(t, x) \in [0, T] \times \mathbb{R}^d$, however it comes with the drawback that the method is significantly more computationally intensive to train, due to how it must account for and remember over the various training steps the different dynamics at various points in space-time. For this reason, the Deep PDE neural network is a good method should one need to simulate the dynamics of a PDE over a variety of different points in the domain of the PDE, but the Deep BSDE method and other methods are more efficient should a solution only at one particular space-time point be required, such as estimating the initial wealth required to reach a particular portfolio value or other such problems.

Appendices

Appendix A

A.0.1 Probabilistic Background

Definition A.0.1 (σ-algebra) *Let Ω be a set. A σ-algebra over Ω is a set $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ (where $\mathcal{P}(\Omega)$ stands for the powerset of Ω) that has the following properties:*

i) *$X \in \mathcal{F}$,*

ii) *If $A \in \mathcal{F}$ then $A^c \in \mathcal{F}$,*

iii) *Let $N \in \mathbb{N} \cup \{\infty\}$. Then for any sequence $\{A_i\}_{i=1}^N$ such that $A_i \in \mathcal{F}$ for every i, then $\cup_{i=1}^N A_i \in \mathcal{F}$.*

We also define a sub-σ-algebra to be a subset of a σ-algebra that is itself a σ-algebra.

As an example, we can consider all intervals of $\mathbb{R}$ of the form $(a, b), [a, b), (a, b]$ or $[a, b]$ where $-\infty \leq a < b \leq \infty$. Taking the smallest σ-algebra that contains all of these subsets, we end up with the Borel σ-algebra, denoted $\mathcal{B}$.

If X is a topological space, and $\mathcal{F}$ is a σ-algebra, then the pair $(X, \mathcal{F})$ is called a *measurable space*. This allows us to introduce the concept of a measurable function. In this case, we say that $f : X \to Y$ is a measurable function if, for all open sets $U \subseteq Y$, the pre-image of U under f, denoted $f^{-1}(U)$ is a member of $\mathcal{F}$. A *probability measure* is simply a measurable function

Finally, a *probability space* is a tumple $(X, \mathcal{F}, P)$ where X is a topological space, $\mathcal{F}$ is a filtration and P is a probability measure (that is, P takes values only in the interval

$[0,1]$ which also satisfies that $P(\cup_{i\in I}E_i = \sum_{i\in I} P(E_i)$ where I is a countable set and $E_i \subseteq X$ for every $i \in I$).

We use *sigma*-algebras to give a sense of the information available to a given process at a given time. This is formalized using two important concepts. The first of these concepts, known as a filtration, gives the known values that a given process has taken over a set period of time.

Definition A.0.2 *Filtration Given a probability space $(X, \mathcal{F}, P)$, a sequence of σ-algebras given by $(\mathcal{F}_t)_{t\in[0,T]}$ where $\mathcal{F}_t \subseteq \mathcal{F}$ for all $t \in [0,T]$ is a filtration if it satisfies $\mathcal{F}_s \subseteq \mathcal{F}_t$ for any pair $s,t \in [0,T]$ such that $s \leq t$.*

Brownian Motion

Brownian motion is an important stochastic process for understanding SDEs and the eventual BSDEs that will be the topic of this thesis. Historically, the idea of Brownian motion came about in 182, when the botanist Robert Brown noticed the motion of pollen grains suspended in liquid. He noticed that there motion was irregular, but may be explained by random collisions of the pollen grain with the molecules of the water. This idea eventually led to several mathematical models which taken to the limit describe the following stochastic process.

Brownian motion can be adequately described with the following definition:

Definition A.0.3 *Brownian Motion Brownian motion is a stochastic process W_t which satisfies the following properties:*

i) $W_t = 0$ *for* $t = 0$.

ii) The process W_t is almost surely continuous.

iii) Any two increments of W_t are independent.

iv) $W_t - W_s$ *is normally distributed with mean* 0 *and variance* $t - s$.

Importantly, Brownian motion has been used to build a theory of stochastic integration, on which we build the stochastic differential equations. In general, we will assume the use of the *Itô integral*. It is defined in the following way

Definition A.0.4

The Itô integral is an extension of the Riemann-Stieljes integral. Given an interval $[0, t]$ we define a sequence of partitions $\{\pi_n\}$ of $[0, t]$ such that $\lim_{n\to\infty} |\pi_n| = 0$. Then

$$\int_0^t f dW = \sum_{i=1}^{N} f(t_{i-1})(W_{t_i} - W_{t_{i-1}})$$

Using this integral, we can define an Itô process. Specifically, an Itô process is a process of the form

$$X_t = X_0 + \int_0^t f_s ds + \int_0^t \sigma_s dW_s,$$

where B is a Brownian motion. In general we consider σ_s to be a predictable process and f_s to be a predictable and Lebesgue integrable process.

Appendix B

Proof of error convergence for Monte-Carlo

Proof : The proof utilizes pieces from [41] as well as [4].

We need to show that

$$\sup_{0\leq t\leq T} E[|Y_t - Y_t^N|^2] + \int_0^T E[|Z_t - Z_t^N|^2] \leq C\left(|\pi| + E[|\xi - \xi^\pi|^2] + \left(\frac{1}{2} + C|\pi|\right)^N\right)$$

Define an approximation of Z, $\bar{Z}$ using the Martingale Representation Theorem applied to $Y_{t_{i+1}}^{(\infty,\pi)}$ as in the following equation:

$$Y_{t_{i+1}}^{(\infty,\pi)} = E\left[Y_{t_{i+1}}^{(\infty,\pi)}|\mathcal{F}_{t_i}\right] + \int_{t_i}^{t_{i+1}} \bar{Z}_s dW_s.$$

We now look at the difference between Y_{t_i} and the approximation of Y_{t_i} given by $Y_{t_i}^{(\infty,\pi)}$ based on this representation, and compared with our original definitions:

$$\begin{aligned} &Y_{t_i}^{(\infty,\pi)} - Y_{t_i} + \int_{t_1}^{t_{i+1}} (\bar{Z}_s - Z_s) ds \\ =&Y_{t_i}^{(\infty,\pi)} - Y_{t_i} - \int_{t_i}^{t_{i+1}} \left(f(t_i, X_{t_i}^\pi, Y_{t_i}^{\infty,\pi}, Z_{t_i}^{\infty,\pi}) - f(s, X_s, Y_s, Z_s)\right) ds \end{aligned}$$

Now, we can use a trick to allow us to utilize Itô's inequality. First, square both sides

of the equation and then take expectations. This gives us the following equation

$$
\begin{aligned}
& E\left[\left|Y_{t_i}^{\infty,\pi}-Y_{t_i}\right|^2\right]+E\int_{t_i}^{t_{i+1}}\left|\hat{Z}_s-Z_s\right|^2 ds \\
= \;& E\left[\left(-\int_{t_i}^{t_{i+1}}\left(f\left(t_i,X_{t_i}^{\pi},Y_{t_i}^{\infty,\pi},Z_{t_i}^{\infty,\pi}\right)-f\left(s,X_s,Y_s,Z_s\right)\right)ds+Y_{t_{i+1}}^{\infty,\pi}-Y_{t_{i+1}}\right)^2\right] \\
\leq \;& E\Bigg[\Bigg(|Y_{t_{i+1}}^{\infty,\pi}-Y_{t_{i+1}}|+|\pi|^{3/2}+C|\pi|(|X_{t_t}-X_{t_i}^{\pi}|+\sup_{t_i\leq t\leq t_{i+1}}|X_t-X_{t_i}|) \\
& +C|\pi|(|Y_{t_i}-Y_{t_i}^{\infty,\pi}|+\sup_{t_i\leq t\leq t_{i+1}}|Y_t-Y_{t_i}|)+C\int_{t_i}^{t_{i+1}}|Z_s-Z_{t_i}^{\infty,\pi}|ds\Bigg)^2\Bigg] \\
\leq \;& (1+\frac{||\pi|}{\varepsilon}E[|Y_{t_{i+1}^{\infty,\pi}}-Y_{t_{i+1}}|^2]+C(1+\frac{\varepsilon}{|\pi|})|\pi|^3 \\
& +C(1+\frac{\varepsilon}{|\pi|}(|\pi|^2E[|Y_{t_i}^{\infty\pi}-Y_{t_i}|^2]+|\pi|E[\int_{t_i}^{t_{i+1}}|Z_s-Z_{t_i}^{\infty,\pi}|^2)
\end{aligned}
$$

Where the assumption 2.1 is used for the second line, and Young's Inequality combined with the assumption in the theorem is used. It remains to estimate the term $\hat{Z}_{t_i}^{\pi}=\frac{1}{|\pi|}E\left[\int_{t_i}^{t_{i+1}}Z_s ds\middle|\mathcal{F}_{t_i}\right]$.

In addition, the upper bound

$$\sum_{i=0}^{N-1}E\left[\int_{t_i}^{t_{i+1}}\left|Z_s-\hat{Z}_{t_i}^{\pi}\right|ds\right]\leq C|\pi|$$

from [41] is required.

Also note that by our introduction of $\hat{Z}_t(\pi)$ and using $Itô$'s isometry we get

$$Z_{t_i}^{(\infty,\pi)}=\frac{1}{\Delta_{t_i}}E\left[\int_{t_i}^{t_{i+1}}\hat{Z}^{(\pi)}ds\middle|\mathcal{F}_{t_i}\right] \tag{B.1}$$

Therefore we can obtain the following inequality

$$E\left[\int_{t_i}^{t_{i+1}} |Z_s - Z_{t_i}^{(\infty,\pi)}|^2 ds\right] \leq 2\left(E\left[\int_{t_i}^{t_{i+1}} |Z_s - \hat{Z}_{t_i}^{(\pi)}|^2\right] + E\left[\int_{t_i}^{t_{i+1}} |Z_s - \tilde{Z}_s^{(\pi)}|^2 ds\right]\right) \tag{B.2}$$

Now we take

$$\begin{aligned}
&(1 - \frac{\Delta_{t_i}}{4})E\left[|Y_{t_i}^{(\infty,\pi)} - Y_{t_i}|^2\right] + \frac{1}{2}E\left[\int_{t_i}^{t_{i+1}} |\tilde{Z}_s^{(\pi)} - Z_s|^2 ds\right] \\
&\leq (1 + \frac{\Delta_{t_i}}{\varepsilon})E\left[Y_{t_{i+1}}^{(\infty,\pi)} - Y_{t_{i+1}}|^2\right] + C\Delta_{t_i} + \frac{1}{2}E\left[\int_{t_i}^{t_{i+1}} |\hat{Z}_s^{(\pi)} - Z_s|^2 ds\right]
\end{aligned}$$

Now with $|\pi|$ sufficiently small we have the identity

$$(1 + \frac{\Delta_{t_i}}{\varepsilon})\frac{1}{1 - \frac{\Delta_{t_i}}{4}} \leq (1 + \frac{\Delta_{t_i}}{\varepsilon} + \frac{\Delta_{t_i}}{2}).$$

With said small $|\pi|$, $B.2$ and the inequality from the earlier trick, we find

$$\begin{aligned}
&E\left[|Y_{t_i}^{(\infty,\pi)} - Y_{t_i}|^2\right] + \frac{1}{2}E\left[\int_{t_i}^{t_{i+1}} |\tilde{Z}_s^{(\pi)} - Z_s|^2 ds\right] \\
&\leq (1 + C\Delta_{t_i})E\left[|Y_{t_{i+1}}^{(\infty,\pi)} - Y_{t_{i+1}}|^2\right] + C\Delta_{t_i}^2 + CE\left[\int_{t_i}^{t_{i+1}} |\hat{Z}_s^{(\pi)} - Z_s|^2 ds\right]
\end{aligned}$$

Recalling the upper bound on $\sum_{i=0}^{N-1} E\left[\int_{t_i}^{t_{i+1}} \left|Z_s - \hat{Z}_{t_i}^{\pi}\right| ds\right]$ from earlier, along with the bound on $E\left[\int_{t_i}^{t_{i+1}} |Z_s - Z_{t_i}^{(\infty,\pi)}|^2 ds\right]$, we see

$$\begin{aligned}
&\max_{0\leq i\leq N} E\left[|Y_{t_i}^{(\infty,\pi)} - Y_{t_i}|^2\right] \\
&\leq C\left(E[|Y_T - Y_T^{(\pi)}|^2] + \sum_{i=0}^{N-1} \Delta_{t_i} + E\left[\int_{t_i}^{t_{i+1}} |Z_s - \hat{Z}_{t_i}^{(\pi)}|^2 ds\right]\right) \leq C\left(E[|Y_T - Y_T^{(\pi)}|^2] + |\pi|\right)
\end{aligned}$$

where the Gronwall lemma is used to eliminate the integral term on the second line. Convergence of $Z^{(\infty,\pi)}$ is now derived

$$\begin{aligned}
&\sum_{i=0}^{N-1} E\left[\int_{t_i}^{t_{i+1}} |\tilde{Z}_s^{(\pi)} - Z_s|^2 ds\right] \\
&\leq C\sum_{i=1}^{N-1} E\left[|Y_{t_i}^{(\infty,\pi)} - Y_{t_i}|^2 \Delta_{t_i}\right] + CE\left[|Y_T - Y_T^{(\pi)}|^2\right] + C|\pi| + C\sum_{i=0}^{N-1} E\left[\int_{t_i}^{t_{i+1}} |\hat{Z}_s^{(\pi)} - Z_s|^2 ds\right] \\
&\leq C(E[|Y_T - Y_T^{(\pi)}|^2] + |\pi|).
\end{aligned}$$

From which the convergence of the estimate to $Z^{(\infty,\pi)}$ can be derived. We still need to derive the bounds on the error between $Y_{t_i}^{\infty,\pi}$ and $Y_{t_i}^{n,\pi}$ (and similarly for $Z_{t_i}^{\infty,\pi}$ and $Z_{t_i}^{n,\pi}$) to complete the proof, which is done by the theorem below.

Theorem B.0.1 *Finally, now that we have some the upper bound on the error between Y_t, $Y_t^{\infty,\pi}$, Z_t, and $Z_t^{\infty,\pi}$ we now show there exists constants C_1, C_2 such that*

$$\max_{0\leq i\leq N} E\left[\left|Y_{t_i}^{\infty,\pi} - Y_{t_i}^{n,\pi}\right|^2\right] + \sum_{i=0}^{N-1} E\left[\left|Z_{t_i}^{\infty,\pi} - Z_{t_i}^{n,\pi}\right|^2\right]\Delta t_i \leq C_1\left(\frac{1}{2} + C_2|\pi|\right)^n$$

so long as $|\pi|$ is sufficiently small. This is based on the proof from Theorem 5 in [3].

Proof:

It follows from Lemma 7 in [3] that

$$\begin{aligned}
&\max_{0\leq i\leq N} \lambda_i E\left[|y_{t_i}^{(n+1,\pi)}|^2\right] + \sum_{i=0}^{N-1} \lambda_i E\left[|z_{t_i}^{(n+1,\pi)}|^2\right]\Delta_{t_i} \\
&\qquad \leq K^2(T+1)\left(\left(|\pi| + \Gamma^{-1}\right)(\gamma DT + 1) + \frac{D}{\gamma}\right) \\
&\qquad \times \left(\max_{0\leq i\leq N} \lambda_i E\left[|y_{t_i}^{(n,\pi)}|^2\right] + \sum_{i=0}^{N-1} \lambda_i E\left[|z_{t_i}^{(n,\pi)}|^2\right]\Delta_{t_i}\right).
\end{aligned}$$

Where $\lambda_0 = 1$ and $\lambda_i = (1 + \Gamma\Delta_{t_{i-1}})\lambda_{i-1}$, $y_{t_i}^{(n+1,\pi)} = Y_{t_i}^{(n+1,\pi)} - Y_{t_i}^{(n,\pi)}$ and $z_{t_i}^{(n+1,\pi)} = Z_{t_i}^{(n+1,\pi)} - Z_{t_i}^{(n,\pi)}$.

Choosing $\gamma = 4DK^2(T+1)$ and $\Gamma = 4K^2(T+1)(\gamma DT+1)$ and iterating the inequality from Lemma 7, the following is obtained

$$\begin{aligned}
&\max_{0\le i\le N} \lambda_i E\left[|y_{t_i}^{(n+1,\pi)}|^2\right] + \sum_{i=0}^{N-1} E\left[|z_{t_i}^{(n+1,\pi)}|^2\right]\Delta_{t_i} \\
&\le \left(\frac{\Gamma|\pi|}{4} + \frac{1}{2}\right)^n \left(\max_{0\le i\le n} \lambda_i E\left[|Y_{t_i}^{(1,\pi)}|^2\right] + \sum_{i=0}^{N-1} \lambda_i E\left[|z_{t_i}^{(1,\pi)}|^2\right]\Delta_{t_i}\right)
\end{aligned}$$

By the definition of λ_i we obtain

$$\begin{aligned}
&\max_{0\le i\le N} E\left[|y_{t_i}^{(n+1,\pi)}|^2\right] + \sum_{i=0}^{N-1} E\left[|z_{t_i}^{(n+1,\pi)}|^2\right]\Delta_{t_i} \\
&\qquad \le e^{\Gamma T}\left(\frac{\Gamma|\pi|}{4} + \frac{1}{2}\right)^n \left(\max_{0\le i\le N} E\left[|Y_{t_i}^{(1,\pi)}|^2\right] + \sum_{i=0}^{N-1} E\left[|Z_{t_i}^{(1,\pi)}|^2\right]\Delta_{t_i}\right).
\end{aligned}$$

If we can take the the square root of the left hand side of the above equation and denote it $A^{(n,\pi)}$ then we see that $\sum_n A^{(n,\pi)}$ converges so long as $|\pi|$ is sufficiently small. Therefore, $(Y^{n,\pi)}, Z^{(n,\pi)})$ is a Cauchy sequence and so converges to $(Y^{(\infty,\pi)}, Z^{(\infty,\pi)})$ as required when $|\pi|$ is sufficiently small. □

An application of both of these Theorems provides the convergence bounds given by 2.2.1 (see [3] and [4]).

Appendix C

C.1 The relationship between BSDEs and PDEs

One of the main interest in developing a theory of BSDEs is a duality between solutions of BSDEs and the solution of a corresponding PDE. Traditionally, it has been known that SDEs and PDEs have a link due to the Feynman-Kac formula, which provides the solution a given linear PDE by the expected value of a solution to an SDE.

Theorem C.1.1 (Feynman-Kac Formula) *[28] Let $f \in C^2(\mathbb{R}^n)$ and $g \in C(\mathbb{R}^n)$, where this is some lower bound $L \in \mathbb{R}^n$ such that $g(\boldsymbol{x}) \geq L$ for all $\boldsymbol{x} \in \mathbb{R}^n$.*

Define

$$v(t,x) = \mathbb{E}\left[\exp\left(-\int_0^t g(X_s)ds\right) f(X_t)\right]$$

and consider the differential equation

$$\frac{\partial v}{\partial t} = Av - gv \qquad t > 0, x \in \mathbb{R}^n$$

with initial condition given by $v(0,x) = f(x)$. If $u(t,x) \in C^{1,2}(\mathbb{R} \times \mathbb{R}^n)$ is bounded on $K \times \mathbb{R}^n$ for each compact $K \subseteq \mathbb{R}$, where u solves the above differential equation then $u(t,x) = v(t,x)$. Here A is a linear operator (see [28]).

Likewise, the solutions to certain PDEs give rise to the solutions to associated BSDEs, and solutions to BSDEs can be used to find weaker solutions to PDEs in the form of viscosity solutions [32]. For this reason, BSDEs are also of interest outside

of Mathematical finance, as their solutions give rise to solutions of many different complex PDEs.

Although this initial formula created a direct link between linear PDEs and SDEs, more was needed to allow stochastic formulas to approximate the solutions to larger classes of PDEs, especially those that had at least some non-linearity. Research from Peng et al. eventually allowed for this relationship to be expanded by a non-linear Feynman-Kac formulation. In particular, given a PDE of the following form:

$$\partial_t u + \frac{1}{2}\Delta u + f(t, u, \nabla u) = 0 \tag{C.1}$$

The solution of the PDE, $u(t, x)$ is given by the BSDE of the form

$$Y_t = g(B_T) + \int_t^T f(t, X, Z)dt + \int_t^T Z dW_t \tag{C.2}$$

This was eventually extended by Peng et al. [31] to an association between systems of quasilinear parabolic PDEs of the below form, with their solution not corresponding to the expected value of a simple SDE, but requiring a BSDE associated with it since these equations were more complex.

$$\partial_t u_k + \frac{1}{2}\Delta u + \langle f(t, x, u, \nabla u\sigma(t, x, u)), \nabla u_k\rangle + g_k(t, x, u, \nabla u\sigma(t, x, u)) = 0 \tag{C.3}$$

$$\text{for } k = 1, \ldots, m, t \in (0, T), x \in \mathbb{R}^n, \tag{C.4}$$

$$u_k(T, x) = h_k(x), \text{ for } k = 1, \ldots, m, x \in \mathbb{R}^n \tag{C.5}$$

Where here the corresponding BSDE takes the form of a Fully Coupled Forward-Backward Stochastic Differential Equation (FBSDE) pair:

$$X_t = x + \int_0^t f(s, X_s, Y_s, Z_s)ds + \int_0^t \sigma(s, X_s, Y_s, Z_s)dW_s \tag{C.6}$$

$$Y_t = h(X_T) + \int_t^T g(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s dW_s \qquad \text{for } t \in [0, T] \tag{C.7}$$

For the associated PDE, we will shortly show that the viscosity solution u of this PDE takes the form

$$u(t, x) := Y^{t,x}(t)$$

where here we have $Y^{t,x}$ is the solution of the FBSDE pair

$$X_s^{t,x} = x + \int_t^s f(r, X_r^{t,x}, Y_r^{t,x}, Z_r^{t,x})dr + \int_t^s \sigma(r, X_r^{t,x}, Y_r^{t,x}, Z_r^{t,x})dW_r$$

$$Y^{t,x} = h(X_T^{t,x}) + \int_s^T g(r, X_r^{t,x}, Y_r^{t,x}, Z_r^{t,x})dr - \int_s^T Z_r^{t,x} dW_r$$

This holds, so long as the functions f, σ, g, h are continuous and satisfy the assumptions $(A1) - (A4)$ from Section 1.2.2.

We now give a proof of this result, reproduced from [31].

Proof: We consider a function $\varphi \in C^{1,2}([0, T] \times \mathbb{R}^n)$. And consider a local minimum $(t, x) \in [0, T] \times \mathbb{R}^n$ of the function $u - \varphi$. This proof shows that u is a superviscosity solution of the PDE C.3. A similar argument can be used to prove it is a subviscosity solution.

We assume without loss of generality that $\frac{\partial \varphi}{\partial t}(t, x)+(L\varphi)(t, x, u(t, x), \nabla\varphi(t, x)\sigma(t, x, u(t, x)))+ g(t, x, u(t, x), \nabla\varphi(t, x)\sigma(t, x, u(t, x))) < 0$. Then, we attempt to derive a contradiction.

Given the above assumption, there exists an $\alpha \in \mathbb{R}$ such that $0 < \alpha < T - t$ and such that for all $(s, y) \in [t, T] \times \mathbb{R}^n$ satisfying

$$t \leq s \leq t + \alpha, \qquad |x - y| \leq \alpha$$

we have $u(s,y) \leq \varphi(s,y)$ and

$\frac{\partial \varphi}{\partial t}(s,y)+(L\varphi)(s,y,u(s,y),\nabla\varphi(s,y)\sigma(s,y,u(s,y)))+g(s,y,u(s,y),\nabla\varphi(s,y)\sigma(s,y,u(s,y))) < 0$

If we let τ denote the stopping time, that is

$\tau := \inf\{s > t : |X_s^{t,s} - x| \geq \alpha\} \wedge (t+\alpha)$ and consider the pair of processes

$$(\bar{Y}_s, \bar{Z}_s) := (Y_{s\wedge t}^{t,x}, \mathbf{1}_{s\in[t,T]} Z_s^{t,x}), \qquad t \leq s \leq t+\alpha$$

solves the BSDE given by

$$\bar{Y}_s = u(\tau, X_\tau^{t,x}) + \int_{s\wedge\tau}^{t} g(r, X_r^{t,x}, u(r, X_r^{t,x}), \bar{Z}_r)dr - \int_s^{t+\alpha} Z_r dW_r \tag{C.8}$$

$$\tag{C.9}$$

And it follows from an application of Itô's formula that the pair of processes $(\hat{Y}_s, \hat{Z}_s)$ defined by

$$(\hat{Y}_s, \hat{Z}_s)L = (\varphi(s\wedge\tau, X_{s\wedge\tau}^{t,x}), \mathbf{1}_{s\in[t,T]}\nabla\varphi(s, X_s^{t,x})\sigma(s, X_s^{t,x}, u(s, X_s^{t,x})) \qquad t \leq s \leq t+\alpha$$

is the solution to the BSDE of the form

$$\hat{Y}_s = \varphi(\tau, X_\tau^{t,x}) - \int_{s\wedge\tau}^{\tau} \left(\frac{\partial\varphi}{\partial t}(r, X_r^{t,x}) + L\varphi(r, X_r^{t,x}, u(r, X_r^{t,x}), \bar{Z}_r)\right) dr - \int_s^{t+\alpha} \hat{Z}_r dW_r,$$

$$t \leq s \leq t+\alpha$$

Defining the following functions

$$\hat{\beta}_r = -\left(\frac{\partial \varphi}{\partial t}(r, X_r^{t,x}) + L\varphi(r, X_r^{t,x}, u(r, X_r^{t,x}), \hat{Z}_r)_g(r, X_r^{t,x}, u(r, X_r^{t,x}), \hat{Z}_r)\right)$$
$$\bar{\beta}_r = -\left(\frac{\partial \varphi}{\partial t}(r, X_r^{t,x}) + L\varphi(r, X_r^{t,x}, u(r, X_r^{t,x}), \bar{Z}_r)_g(r, X_r^{t,x}, u(r, X_r^{t,x}), \bar{Z}_r)\right)$$

Then for some $c > 0$ it must be the case from our prior assumptions that

$$|\hat{\beta}_r - \bar{\beta}_r| \leq c||\hat{Z}_r - \bar{Z}_r||$$

from which it follows that there is a bounded, $\mathcal{F}_t$-adapted process $\{\gamma_r\}$ such that $\gamma_r \in \mathbb{R}^d$ for all r such that

$$\hat{\beta}_r - \bar{\beta}_r = \langle \gamma_r, \hat{Z}_r - \bar{Z}_r \rangle$$

Defining $(\tilde{Y}_s, \tilde{Z}_s) = (\hat{Y}_s - \bar{Y}_s, \hat{Z}_s - \bar{Z}_s)$ then we can write $\tilde{Y}_s$ in the form

$$\tilde{Y}_s = \varphi(\tau, X_\tau^{t,x}) - u(\tau, X_\tau^{t,x}) + \int_{s\wedge\tau}^{\tau} [\hat{\beta}_r + \langle \gamma_r, \tilde{Z}_r \rangle] dr - \int_{s\wedge\tau}^{\tau} \tilde{Z}_r dW_r$$

To continue, we need to borrow a theorem from [29] which gives us that

$$\tilde{Y}_t = E\left[\Gamma_{t,\tau}\tilde{Y}_\tau + \int_t^\tau \Gamma_{t,s}\hat{\beta}_s ds\right]$$

where the function $\Gamma_{t,\tau}$ is given by

$$\Gamma_{t,s} = \exp\left\{\int_t^s \langle \gamma_r, dW_r \rangle - \frac{1}{2}\int_t^s |\gamma_r|^2 dr\right\}$$

It follows that $u(\tau, X_\tau^{t,x}) \leq \varphi(\tau, X_\tau^{t,x})$, with $0 < \hat{\beta}_r$ on the interval $[t, \tau]$, where $\tau > t$. This implies that $\tilde{Y}_t > 0$ which necessarily means $u(t,x) < \varphi(t,x)$ which is a contradiction. Hence $u(t,x)$ is a viscosity subsolution of the given PDE. $\square$

Bibliography

[1] Francis Bach. "Breaking the curse of dimensionality with convex neural networks". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 629–681.

[2] Christian Beck, E Weinan, and Arnulf Jentzen. "Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations". In: *Journal of Nonlinear Science* 29.4 (2019), pp. 1563–1619.

[3] Christian Bender and Robert Denk. "A forward scheme for backward SDEs". In: *Stochastic processes and their applications* 117.12 (2007), pp. 1793–1812.

[4] Christian Bender and Robert Denk. *Forward simulation of backward SDEs*. WIAS, 2005.

[5] Christian Bender and Jessica Steiner. "Least-squares monte carlo for backward sdes". In: *Numerical methods in finance*. Springer, 2012, pp. 257–289.

[6] Yoshua Bengio et al. "Learning deep architectures for AI". In: *Foundations and trends® in Machine Learning* 2.1 (2009), pp. 1–127.

[7] Bruno Bouchard et al. "Numerical approximation of BSDEs using local polynomial drivers and branching processes". In: *Monte Carlo Methods and Applications* 23.4 (2017), pp. 241–263.

[8] Philippe Briand and Céline Labart. "Simulation of BSDEs by Wiener chaos expansion". In: *The Annals of Applied Probability* 24.3 (2014), pp. 1129–1171.

[9] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[10] Nicole El Karoui, Shige Peng, and Marie Claire Quenez. "Backward stochastic differential equations in finance". In: *Mathematical finance* 7.1 (1997), pp. 1–71.

[11] Emmanuel Gobet, Jean-Philippe Lemor, Xavier Warin, et al. "A regression-based Monte Carlo method to solve backward stochastic differential equations". In: *The Annals of Applied Probability* 15.3 (2005), pp. 2172–2202.

[12] Emmanuel Gobet and Plamen Turkedjiev. "Linear regression MDP scheme for discrete backward stochastic differential equations under general conditions". In: *Mathematics of Computation* 85.299 (2016), pp. 1359–1391.

[13] Jiequn Han et al. "Deep learning approximation for stochastic control problems". In: *arXiv preprint arXiv:1611.07422* (2016).

[14] Jiequn Han, Arnulf Jentzen, and E Weinan. "Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning". In: *arXiv preprint arXiv:1707.02568* (2017), pp. 1–13.

[15] Jiequn Han and Jihao Long. "Convergence of the deep BSDE method for coupled FBSDEs". In: *arXiv preprint arXiv:1811.01165* (2018).

[16] Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[17] Côme Huré, Huyên Pham, and Xavier Warin. "Some machine learning schemes for high-dimensional nonlinear PDEs". In: *arXiv preprint arXiv:1902.01599* (2019).

[18] Peter Imkeller, Gonçalo Dos Reis, and Jianing Zhang. "Results on numerics for FBSDE with drivers of quadratic growth". In: *Contemporary Quantitative Finance*. Springer, 2010, pp. 159–182.

[19] Petar Jevtic, Minsuk Kwak, and Traian A Pirvu. "Longevity bond pricing in equilibrium". In: *Available at SSRN 3206195* (2019).

[20] Magdalena Kobylanski. "Backward stochastic differential equations and partial differential equations with quadratic growth". In: *Ann. Probab.* 28.2 (Apr. 2000), pp. 558–602. DOI: 10.1214/aop/1019160253. URL: https://doi.org/10.1214/aop/1019160253.

[21] Hyuk Lee and In Seok Kang. "Neural algorithm for solving differential equations". In: *Journal of Computational Physics* 91.1 (1990), pp. 110–131.

[22] Jin Ma, J-M Morel, and Jiongmin Yong. *Forward-backward stochastic differential equations and their applications*. 1702. Springer Science & Business Media, 1999.

[23] Jin Ma, Philip Protter, and Jiongmin Yong. "Solving forward-backward stochastic differential equations explicitly—a four step scheme". In: *Probability theory and related fields* 98.3 (1994), pp. 339–359.

[24] Andrew J Meade Jr and Alvaro A Fernandez. "The numerical solution of linear ordinary differential equations by feedforward neural networks". In: *Mathematical and Computer Modelling* 19.12 (1994), pp. 1–25.

[25] Riu Naito and Toshihiro Yamada. "A third-order weak approximation of multi-dimensional Itô stochastic differential equations". In: *Monte Carlo Methods and Applications* 25.2 (2019), pp. 97–120.

[26] Riu Naito and Toshihiro Yamada. "An acceleration scheme for deep learning-based BSDE solver using weak expansions". In: *International Journal of Financial Engineering* (2020), p. 2050012.

[27] M Negnevitsky. "The History Of Artificial Intelligence Or From The "Dark Ages" To the Knowledge-based Systems". In: *WIT Transactions on Information and Communication Technologies* 19 (1997).

[28] Bernt Oksendal. *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013.

[29] Étienne Pardoux. "Backward stochastic differential equations and viscosity solutions of systems of semilinear parabolic and elliptic PDEs of second order". In: *Stochastic Analysis and Related Topics VI*. Springer, 1998, pp. 79–127.

[30] Etienne Pardoux and Shige Peng. "Adapted solution of a backward stochastic differential equation". In: *Systems & Control Letters* 14.1 (1990), pp. 55–61.

[31] Etienne Pardoux and Shanjian Tang. "Forward-backward stochastic differential equations and quasilinear parabolic PDEs". In: *Probability Theory and Related Fields* 114.2 (1999), pp. 123–150.

[32] Shige Peng and Falei Wang. *BSDE, Path-dependent PDE and Nonlinear Feynman-Kac Formula*. 2011. arXiv: `1108.4317 [math.PR]`.

[33] Allan Pinkus. "Approximation theory of the MLP model in neural networks". In: *Acta numerica* 8 (1999), pp. 143–195.

[34] Pradeep Ramuhalli, Lalita Udpa, and Satish S Udpa. "Finite-element neural networks for solving differential equations". In: *IEEE transactions on neural networks* 16.6 (2005), pp. 1381–1392.

[35] SITU Rong. *Theory of stochastic differential equations with jumps and applications: mathematical and analytical techniques with applications to engineering*. Springer Science & Business Media, 2006.

[36] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[37] Justin Sirignano, Jonathan F MacArt, and Jonathan B Freund. "DPM: A deep learning PDE augmentation method with application to large-eddy simulation". In: *Journal of Computational Physics* 423 (2020), p. 109811.

[38] Justin Sirignano and Konstantinos Spiliopoulos. "DGM: A deep learning algorithm for solving partial differential equations". In: *Journal of computational physics* 375 (2018), pp. 1339–1364.

[39] E Weinan, Jiequn Han, and Arnulf Jentzen. "Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations". In: *Communications in Mathematics and Statistics* 5.4 (2017), pp. 349–380.

[40] Jiongmin Yong and Xun Yu Zhou. *Stochastic controls: Hamiltonian systems and HJB equations*. Vol. 43. Springer Science & Business Media, 1999.

[41] Jianfeng Zhang. "Some fine properties of backward stochastic differential equations". PhD thesis. PhD thesis, Purdue University, 2001.