Master's thesis

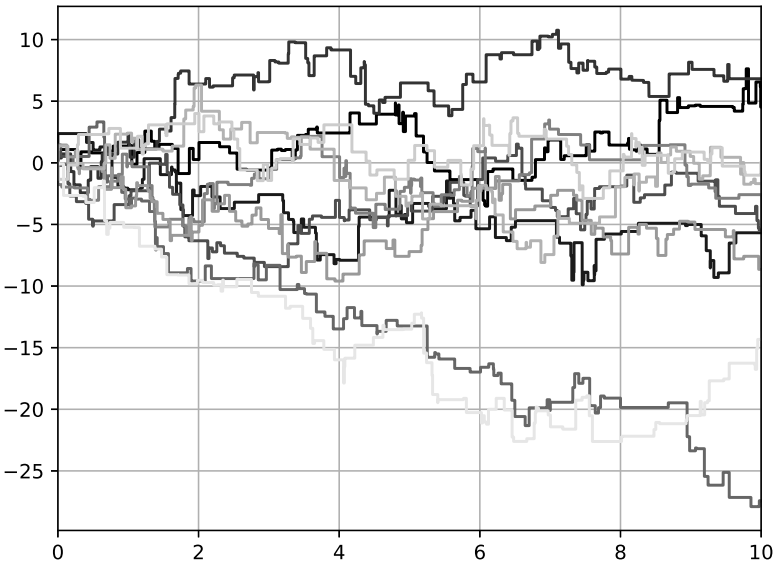
NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Oscar Christian Ameln

Deep Learning Algorithms for Solving PDEs

Presentation and Implementation of Deep Learning Algorithms for Solving Semi-Linear Parabolic PDEs with an Extension to the Fractional Laplace Operator

Master's thesis in Applied Physics and Mathematics
Supervisor: Espen Robstad Jakobsen
July 2020



NTNU
Norwegian University of Science and Technology

Oscar Christian Ameln

Deep Learning Algorithms for Solving PDEs

Presentation and Implementation of Deep Learning Algorithms for Solving Semi-Linear Parabolic PDEs with an Extension to the Fractional Laplace Operator

Master's thesis in Applied Physics and Mathematics
Supervisor: Espen Robstad Jakobsen
July 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Summary

In June 2017 Weinan E, Jiequn Han and Arnulf Jentzen present a pioneering algorithm, Deep Backward Stochastic Differential Equation (Deep BSDE), to solve partial differential equations (PDEs) using deep learning. In February 2019 Côme Huré, Huyên Pham and Xavier Warin introduce a modification of Deep BSDE, Deep Backward Dynamic Programming (DBDP). Furthermore, DBDP has two different variants. The goal of the algorithms is to avoid the curse of dimensionality. This is done by reformulating the PDEs to learning problems.

A thorough description is given of the theoretical foundation behind the algorithms. We need stochastic calculus to understand how the PDE is reformulated to a pair of stochastic differential equations. Neural networks act as approximators for unknowns in the stochastic differential equations.

The source code for DBDP is not publicly available. Hence, the two variants of DBDP are implemented in Python using the TensorFlow 2.0 framework. Deep BSDE and DBDP are tested on a wide range of problems within different fields of science. The numerical results verify that both Deep BSDE and the two variants of DBDP successfully solves 100-dimensional semi-linear parabolic PDEs in most cases. Both variants of DBDP converges to the wrong value for only one of the test examples. Although the relative approximation error is somewhat high, of the order 1%, for most of the cases, being able to solve such high dimensional PDEs is in practice not possible for traditional methods.

At last, an algorithm which solves fractional Laplace equations is developed. The algorithm is inspired by the deep learning algorithms for solving PDEs, in particular DBDP. The algorithm is implemented in TensorFlow 2.0. Some numerical results are provided and shows that the algorithm suffer from instability, but still produces meaningful results for some cases in one dimension.

Sammendrag

I juni 2017 presenterer Weinan E, Jiequn Han og Arnulf Jentzen en banebrytende algoritme, Deep Backward Stochastic Differential Equation (Deep BSDE), for å løse partielle differensiallikninger (PDEer) ved bruk av dyp læring. I februar 2019 introduserer Côme Huré, Huyên Pham og Xavier Warin en modifikasjon av Deep BSDE, Deep Backward Dynamic Programming (DBDP). DBDP kommer i to varianter. Målet til algoritmene er å unngå dimensjonenes forbannelse. Dette gjøres ved å reformulere PDEene til læringsproblemer.

En grundig beskrivelse av det teoretiske fundamentet bak algoritmene er gitt. Vi trenger innsikt i stokastisk analyse for å forstå hvordan PDEer reformuleres til et par stokastiske differensiallikninger. Nevrale nettverk introduseres slik at de kan brukes til å tilnærme ukjente i de stokastiske differensiallikningene.

Kildekoden til DBDP er ikke offentliggjort. Derfor har de to variantene av DBDP blitt implementert i Python ved bruk av TensorFlow 2.0-rammeverket. Deep BSDE og DBDP er testet på et utvalg av problemer innen forskjellige vitenskapsgrener. Numeriske resultater viser at både Deep BSDE og de to variantene av DBDP løser 100-dimensjonale semilineære parabolske PDEer i de fleste tilfeller. Begge variantene av DBDP konvergerer til en feil verdi for kun ett av testeksemplene. Selv om den relative approksimasjonsfeilen er noe høy, av orden 1%, i de fleste tilfeller, vil slike høydimensjonale likninger ikke være mulig å løse ved tradisjonelle metoder.

Til slutt utledes en algoritme som kan løse likninger som inneholder den fraksjonelle Laplace-operatoren. Algoritmen er inspirert av de dype læringsalgoritmene for å løse PDEer, i særdeleshet DBDP. Algoritmen er implementert i Python ved bruk av TensorFlow 2.0-rammeverket. Noen numeriske resultater er presentert og viser at algoritmen lider av ustabilitet, men at den likevel klarer å produsere meningsfulle resultater i noen endimensjonale tilfeller.

Preface

This thesis concludes the degree of Master of Science (M.Sc.) in Applied Physics and Mathematics with specialization in Industrial Mathematics. The degree is accomplished at the Department of Mathematical Sciences (IMF) at the Norwegian University of Science and Technology (NTNU) in Trondheim. The work was carried out in the spring of 2020 under the supervision of Professor Espen Robstad Jakobsen at the Department of Mathematical Sciences.

I would like to thank him for his commitment, our weekly discussions and for supporting me way more than I would expect. Further I am grateful that he introduced me to a new field of mathematics which I find really interesting, but also quite challenging. A huge thanks to Bergitte Viste for pulling me across the finish line.

Oscar Christian Ameln
NTNU, Trondheim
July 20, 2020

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Partial differential equations (PDEs) are used to model a wide range of phenomena within all fields of science. However, only a few PDEs have a closed form solutions. Hence, a numerical approximation must be done in most cases. The time complexity of traditional methods, e.g. the finite element method and the finite difference method, scale exponentially. In practice, such methods are not able to solve high dimensional PDEs. These methods are said to suffer from the curse of dimensionality. The goal of the methods presented in this thesis is to avoid the curse of dimensionality by reformulating the PDEs to learning problems.

1.1 Problem and Motivation

In recent years, pioneering research have been carried out, reformulating the PDE to a pair of stochastic differential equations (SDE). One of the SDEs have an initial condition, a so-called forward stochastic differential equation . The other SDE depends on the solution of the first SDE and are equipped with a terminal condition. This SDE is a backward stochastic differential equation (BSDE). The solution of the BSDE at the initial time evaluated at the initial condition of the forward SDE, is the solution of the PDE at that point. However, one problem remains. Some of the components of the BSDE is still unknown. The unknown components are approximated by neural networks. This reformulation and neural network parametrization turns the PDE to a learning problem, avoiding the curse of dimensionality. The focus in this thesis will mainly be on the algorithms presented in Han et al. (2017) and Huré et al. (2019).

The former article considers a semi-linear parabolic PDE reformulated to a decoupled pair of SDEs. The SDEs are approximated numerically by Euler-Maruyama schemes and the unknown gradient of the solution is approximated by a neural network. The problem now resemble a deep reinforcement learning problem, where the gradient acts like a policy function. The full set of neural networks are optimized simultaneously , or trained in the machine learning jargon, by a stochastic gradient descent-like method. The algorithm is called the Deep BSDE.

The latter article presents two variants of the first algorithm. Both variants reformu-

late a semi-linear parabolic PDE to a decoupled pair of SDEs and use Euler-Maruyama schemes to approximate the SDEs. The first variant approximates the gradient of the solution, as well as the solution itself as a neural network. The second variant approximates only the solution as a neural network, and applies numerical- or automatic differentiation to compute the gradient. Contrary to the Deep BSDE algorithm, which trains all the neural networks simultaneously, these variants employ an iterative procedure. They iterate through the time steps, backwards in time, and for each time step the neural network parametrization(s) at the current time step are trained.

1.2 Outline

Necessary theory to understand the PDE solving algorithms is presented in chapter 2 and chapter 3. Chapter 2 presents stochastic calculus which is key to understand the reformulation from the PDE to the forward and backward stochastic differential equation. Chapter 3 presents the role of neural networks in learning theory. Chapter 4 introduces the PDE solving algorithms and presents some numerical results. Chapter 5 presents an extension to equations involving the fractional Laplace operator. The thesis is wrapped up in chapter 6 by some concluding remarks.

Chapter 2

Stochastic Calculus

The concepts behind the deep learning based algorithms for solving PDEs/PIDEs are based on some fundamental theory, such as basic stochastic calculus, to be covered in this chapter. Stochastic calculus is an important part of the deep learning based algorithms as it is used to turn the deterministic differential equation to a pair of stochastic differential equations.

In the first section, some formal probability theoretical concepts and Itô calculus for Brownian motions are introduced. The theory are mostly based on Øksendal (2013). The second section covers a larger class of stochastic processes, Lévy processes, and follows Cont and Tankov (2004) closely.

2.1 Probability Theory and Itô Calculus for Brownian Motions

The formal probability theoretical consepts are introduced such that the preceding concepts can be rigorously presented. We introduce the Itô calculus for stochastic processes driven by Brownian motions and Itô's lemma. As well, the connections between semilinear parabolic PDEs and forward backward stochastic differential equations are looked into.

2.1.1 Probability Theory

The rigorous probability theory poses a theoretical basis for some concepts within stochastic calculus. The relevant probability theoretical concepts now follows.

σ-Algebra

Given a set Ω, then a σ-algebra, $\mathcal{F}$, on Ω is a family of subsets of Ω with the following properties

- $\emptyset \in \mathcal{F}$

- $F \in \mathcal{F} \implies \Omega \setminus F = F^C \in \mathcal{F}$
- $A_1, A_2, \cdots \in \mathcal{F} \implies A = \cup_{i=1}^{\infty} A_i \in \mathcal{F}$

The pair $(\Omega, \mathcal{F})$ is called a measurable space. An important example is the Borel σ-algebra which is the smallest σ-algebra containing all open sets.

Probability Space

A probability measure $P : \mathcal{F} \to [0, 1]$ on a measurably space satisfy:

- $P(\emptyset) = 0$
- $P(\Omega) = 1$
- $P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$

for $A_1, A_2, \cdots \in \mathcal{F}$ disjoint. The triplet $(\Omega, \mathcal{F}, P)$ is called a probability space.

Measurable Function

Given a probability space $(\Omega, \mathcal{F}, P)$, a function $Y : \Omega \to \mathbb{R}^n$ is called measurable if

$$Y^{-1}(U) := \{\omega \in \Omega : Y(\omega) \in U\} \in \mathcal{F}$$

for all open sets $U \in \mathbb{R}^n$.

Filtration

We consider the measurable space $(\Omega, \mathcal{F})$. A filtration on $(\Omega, \mathcal{F})$ is a family $\{\mathcal{M}_t\}_{t \geq 0}$ of σ-algebras $\mathcal{M} \subset \mathcal{F}$ such that

$$0 \leq s < t \implies \mathcal{M}_s \subset \mathcal{M}_t.$$

Martingale

A stochastic process $\{X_t\}_{t \geq 0}$ is called a martingale with respect to a filtration $\{\mathcal{M}\}_{t \geq 0}$ if

- X_t is $\mathcal{M}_t$-measurable for all t
- $\mathrm{E}[|X_t|] < \infty$ for all t
- $\mathrm{E}[X_s | \mathcal{M}_t] = X_t$ for all $s \geq t$

Adapted Process

Let $\{\mathcal{N}_t\}_{t \geq 0}$ be an increasing family of σ-algebras of subsets of Ω. A process $g(t, \omega) : [0, \infty) \times \Omega \to \mathbb{R}$ is called $\mathcal{N}_t$-adapted if for each $t \geq 0$ the function

$$\omega \mapsto g(t, \omega)$$

is $\mathcal{N}_t$-measurable.

L^p-norm

Let $X : \Omega \to \mathbb{R}^d$ be a random variable and $p \in [1, \infty)$ be a constant. Then the L^p-norm of X is

$$||X||_p = ||X||_{L^p} = \Big(\int_\Omega |X(\omega)|^p dP(\omega) \Big)^{\frac{1}{p}} = \Big(\mathrm{E}[|X(\omega)|^p] \Big)^{\frac{1}{p}}. \tag{2.1}$$

2.1.2 Itô Calculus for Brownian Motions

One of the most well known stochastic processes is the Brownian motion. It occurs frequently in mathematics, finance and physics. It is named after the botanist Robert Brown which in 1828 used the Brownian motion to model the collision of pollen grains and molecules of a liquid. The Brownian motion is defined in the following.

Brownian Motion

A stochastic process, $\{B_t\}_{t \geq 0}$, is an n-dimensional Brownian motion if it satisfies the following 3 properties:

1. $B_0 = 0$ almost surely.
2. $B_t - B_s \sim \mathcal{N}(0, (t-s)I)$ for $0 \leq s < t$.
3. $B_{t_1}, B_{t_2} - B_{t_1}, \ldots, B_{t_N} - B_{t_{N-1}}$ is independent for $0 = t_0 < t_1 < t_2 < \cdots < t_N$.

Here I is the identity matrix and $\mathcal{N}(\mu, \Sigma)$ denotes the normal distribution with mean μ and covariance matrix Σ.

Several methods to simulate the Brownian motions exists. Glasserman (2003) describes the random walk construction, Brownian bridge construction and principal component construction. The random walk construction consists of fixing a grid $0 = t_0 < t_1 < \cdots < t_N$ and set $B_0 = 0$. Next step is to use the independent increments property, the third part of the definition, to simulate the Brownian motion based on the increment. We now use the fact that the increments are normally distributed

$$\Delta B_{t_n} = B_{t_{n+1}} - B_{t_n} \sim \mathcal{N}(0, (t_{n+1} - t_n)I), \quad \text{for } n = 0, \ldots, N-1$$

and it is now possible to compute

$$B_{t_n} = \sum_{i=0}^{n-1} \Delta B_{t_i}, \quad \text{for } n = 1, \ldots N$$

which will be an approximated Brownian motion. It is approximate in the sense that the joint distribution of the simulated values, $(B_{t_0}, \ldots, B_{t_n})$ coincides with the joint distribution of the Brownian motion, however the simulated values say nothing about how the Brownian motion behave between the grid points.

The Brownian motion can be extended to have drift, μ, and covariance, Σ, as follows:

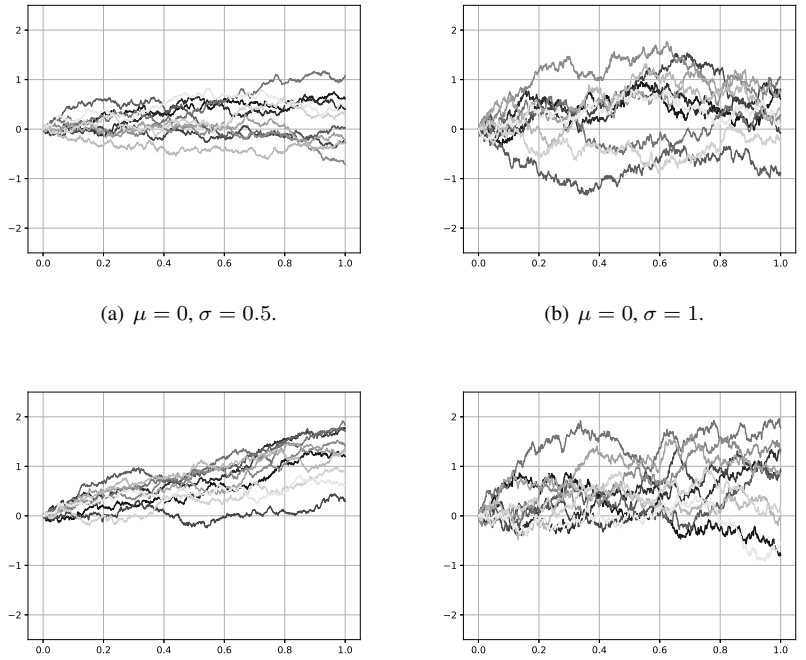
$$X_t = \mu t + \Sigma^{\frac{1}{2}} B_t. \tag{2.2}$$

Here B_t is a Brownian motion in $\mathbb{R}^n$, $\mu \in \mathbb{R}^n$ and $\Sigma^{\frac{1}{2}} \in \mathbb{R}^{n \times n}$ is the principal square root of the desired covariance matrix, Σ. Consider the Brownian motion with constant (not time dependent) μ and Σ. The expression (2.2) can be rearranged to get:

$$\Sigma^{-\frac{1}{2}}(X_t - \mu t) = B_t.$$

Now, B_t can be sample as for the standard case and finally solving for X_t gives the desired properties.

Figure 2.1 displays sample paths for one dimensional Brownian motions for a few combinations of drift, μ, and volatility, $\sigma = \Sigma^{\frac{1}{2}}$ parameters. **Figure 2.1(b)** displays the standard case with $\mu = 0$ and $\sigma = 1$. The sample paths have been generated by the random walk construction.



(a) $\mu = 0, \sigma = 0.5$.

(b) $\mu = 0, \sigma = 1$.

(c) $\mu = 1, \sigma = 0.5$.

(d) $\mu = 1, \sigma = 1$.

Figure 2.1: Brownian motions with drift and volatility parameters.

Itô Integral

We introduce a class of functions, which we denote $\mathcal{L}^2_{(S,T)}$, to help us define the Itô integral. These functions, $f(t, \omega) : [0, \infty) \times \Omega \to \mathbb{R}$, satisfy

- $(t, \omega) \to f(t, \omega)$ is $\mathcal{B} \times \mathcal{F}$-measurable, where $\mathcal{B}$ is the Borel σ-algebra on $[0, \infty)$.

- $f(t,\omega)$ is $\mathcal{F}_t$-adapted.
- $\mathrm{E}[\int_S^T f(t,\omega)^2 dt] < \infty$.

Next step is to define the Itô integral for elementary functions, $\phi \in \mathcal{L}^2$. It is called elementary if it is on the form

$$\phi(t,\omega) = \sum_j e_j(\omega)\chi_{[t_j,t_{j+1})}(t)$$

where each e_j must be $\mathcal{F}_{t_j}$-measurable. Let now $\phi \in \mathcal{L}^2_{(S,T)}$ be an elementary function. We then define the Itô integral to be:

$$\int_S^T \phi(t,\omega)dB_t = \sum_j e_j(\omega)[B_{t_{j+1}} - B_{t_j}]$$

where B_t is a Brownian motion.

We now proceed to define the Itô integral for $f \in \mathcal{L}^2_{(S,T)}$ (not necessarily elementary function). Then

$$\int_S^T f(t,\omega)dB_t = \lim_{n\to\infty}\int_S^T \phi_n(t,\omega)dB_t$$

where $\{\phi_n\}$ is a sequence of elementary functions such that

$$\mathrm{E}\Big[\int_S^T (f(t,\omega) - \phi_n(t,\omega))^2 dt\Big] \to 0 \quad \text{as} \quad n \to \infty$$

Forward Stochastic Differential Equations

A forward stochastic differential equation is a noisy differential equation where one or more of the terms are stochastic processes. A typical equation is on the form

$$X_t = X_s + \int_s^t \mu(r, X_r)dr + \int_s^t \sigma(r, X_r)dB_r, \quad 0 \le s < t \le T, \tag{2.3}$$

with some initial condition $X_0 = \xi : \Omega \to \mathbb{R}^n$. Here $\{B_r\}_{r\in[0,T]}$ is an d-dimensional Brownian motion, where

- $\mu : \Omega \times [0,T] \times \mathbb{R}^n \to \mathbb{R}^n$
- $\sigma : \Omega \times [0,T] \times \mathbb{R}^n \to \mathbb{R}^{n\times d}$

The expression in (2.3) is rarely seen, as the short-hand notation

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dB_t, \quad t \in (0,T] \tag{2.4}$$

is more frequently used.

We assume the initial value, ξ to be independent of the Brownian motion and have finite second moment, that is $\mathrm{E}[|\xi|^2] < \infty$. Secondly, not $\mu(t,x)$ nor $\sigma(t,x)$ must exceed linear growth in x. That is,

$$|\mu(t,x)| + |\sigma(t,x)| \le C(1+|x|), \quad x \in \mathbb{R}^n, \quad t \in [0,T], \tag{2.5}$$

where $|\sigma(t,x)|^2 = \sum_{i,j} |\sigma_{ij}(t,x)|^2$ and C is some constant. Finally, the $\mu(t,x)$ and $\sigma(t,x)$ must be Lipschitz continuous in x,

$$|\mu(t,x) - \mu(t,y)| + |\sigma(t,x) - \sigma(t,y)| \leq D|x-y| \tag{2.6}$$

for some constant D. If these three conditions are satisfied, then the solution exists almost surely and is unique.

Itô's Lemma

Itô's lemma is the stochastic calculus counterpart of the chain rule known from calculus. We consider an n-dimensional stochastic process $\{X_t\}_{t\geq 0}$ which satisfies $dX_t = u(t,X_t)dt + v(t,X_t)dB_t$ and $Y_t = g(t,x) = (g_1(t,x),...,g_p(t,x)) \in C^2([0,\infty) \times \mathbb{R}^n, \mathbb{R}^p)$. Then,

$$dY_t^{(k)} = \frac{\partial g_k}{\partial t}(t,X_t)dt + \sum_{i=1}^{n} \frac{\partial g_k}{\partial x_i}(t,X_t)dX_t^{(i)} + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 g_k}{\partial x_i \partial x_j}(t,X_t)dX_t^{(i)}dX_t^{(j)}.$$

By using $(dt)^2 = dtdB_t^{(i)} = 0$ and $dB_t^{(i)}dB_t^{(j)} = \delta_{ij}dt$ we get:

$$\begin{aligned} dY_t^{(k)} = &\Big[\frac{\partial g_k}{\partial t}(t,X_t) + \sum_{i=1}^{n} u_i \frac{\partial g_k}{\partial x_i}(t,X_t) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} v_i v_j \frac{\partial^2 g_k}{\partial x_i \partial x_j}(t,X_t)\Big]dt \\ &+ \sum_{i=1}^{n} u_i \frac{\partial g_k}{\partial x_i}(t,X_t)dB_t^{(i)}. \end{aligned}$$

We observe that the term involving the second derivative does not appear in the chain rule known from calculus.

An Example: Geometric Brownian Motion

The preceding concepts will be wrapped up by a motivating example. We consider one of the most fundamental examples of an SDE, which in one dimension is on the form:

$$dX_t = X_t\mu dt + X_t\sigma dB_t, \quad X_0 = x_0 \tag{2.7}$$

where μ, σ and x_0 are constants. The equation often appears in finance, where the interpretation is that the rate of return in a market at time t, dX_t/X_t, consists of a drift component μdt and a stochastic noise component, σdB_t. The tool used to solve (2.7) will be Itô's lemma, which will be applied to $g(t,x) = \ln x$:

$$\begin{aligned} d\ln X_t &= \frac{1}{X_t}dX_t - \frac{1}{2}\frac{1}{X_t^2}dX_t^2 = \big(\mu - \frac{\sigma^2}{2}\big)dt + \sigma dB_t \\ X_t &= x_0 \exp\Big(\big(\mu - \frac{\sigma^2}{2}\big)t + \sigma B_t\Big). \end{aligned} \tag{2.8}$$

The differential equations satisfies the criteria for uniqueness and existence, by $C = D = |\mu| + |\sigma|$ in (2.5) and (2.6) and x_0 is a constant. Hence it satisfies independence of the Brownian motion and finite variation trivially. The stochastic process in (2.8) is called a geometric Brownian motion.

Backward Stochastic Differential Equation

The backward SDEs (BSDEs) have a specified terminal condition contrary to the forward SDES, where the initial condition is specified. We could try to consider an equation on the same form as (2.3), this would yield

$$X_t = \xi - \int_t^T \mu(s, X_s)ds - \int_t^T \sigma(s, X_s)dB_s. \tag{2.9}$$

For the non-stochastic case, such terminal condition problems could under certain regularity assumptions be transformed to an initial value problem by a time change $t \mapsto T - t$. An example will show why (2.9) in general is not well-posed.

We consider (2.9) in one dimension. Let $\xi = 1$, $\mu = 0$ and $\sigma = 1$, i.e.,

$$X_t = 1 - \int_t^T dB_s = 1 + B_t - B_T. \tag{2.10}$$

The issue with (2.10) is that it is not adapted. We want solutions that are adapted, i.e. does not see the future. Since X_t depends on B_T for $t < T$, (2.9) is in general not well-posed.

Instead consider equations on the form

$$Y_t = \xi + \int_t^T F(s, Y_s, Z_s)ds - \int_t^T Z_s^\top dB_s, \tag{2.11}$$

where the pair $\{(Y_t, Z_t)\}_{t\geq 0}$ is the solution. Here ξ is the terminal condition, f is called the generator and the pair (ξ, F) is called the data and satisfy regularity conditions as in Pardoux (1995).

Forward Backward Stochastic Differential Equation

A forward backward stochastic differential equation (FBSDE) is a pair of stochastic differential equations:

$$\begin{aligned} X_t &= \xi + \int_0^t \mu(s, X_s, Y_s, Z_s)ds + \int_0^t \sigma(s, X_s, Y_s, Z_s)dB_s \\ Y_t &= g(X_T) + \int_t^T F(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s^\top dB_s \end{aligned} \tag{2.12}$$

for $t \in [0, T]$ where $\{B_t\}_{t\in[0,T]}$ is a d-dimensional Brownian motion,

- $f : \Omega \times [0, T] \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m\times d} \to \mathbb{R}^n$,
- $F : \Omega \times [0, T] \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m\times d} \to \mathbb{R}^m$,
- $\sigma : \Omega \times [0, T] \times \mathbb{R}^d \times \mathbb{R}^m \times \mathbb{R}^{m\times d} \to \mathbb{R}^{n\times d}$,
- $g : \Omega \times \mathbb{R}^d \to \mathbb{R}^m$.

are continuous with respect to $(x, y, z) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m\times d}$. Under certain regularity conditions which are stated in Pardoux and Tang (1999), there exists a unique adapted solution $\{(X_t, Y_t, Z_t)\}_{t\in[0,T]}$ with values in $\mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m\times d}$.

When the forward equation does not depend on the solution of the backward equations $\{(Y_t, Z_t)\}_{t\in[0,T]}$, or the backward equation does not depend on the solution of the forward equation, $\{X_t\}_{t\in[0,T]}$, the equations are said to be decoupled. Decoupled equations are rather easy to solve. Consider the case where the forward equation is not depending on the solution of the backward equation. That is,

$$X_t = \xi + \int_0^t \mu(s, X_s)ds + \int_0^t \sigma(s, X_s)dB_s \tag{2.13}$$

$$Y_t = g(X_t) + \int_t^T F(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s^\top dB_s \tag{2.14}$$

Such FBSDEs can be solved by first solving the forward equation (2.13) to determine the process $\{X_t\}_{t\in[0,T]}$ and then solve the backward equation (2.14) by inserting the solution of the forward process.

Semi-Linear Parabolic PDEs and Their Connections to FBSDEs

The connection between parabolic PDEs and Forward Backward Stochastic Differential Equations, FBSDEs, are studied exhaustively in Pardoux and Răşcanu (2014). A brief summary is now given.

We look at a family of PDEs, namely semi-linear parabolic PDEs, which can be represented as

$$\begin{aligned}\frac{\partial u}{\partial t}(t,x) + \frac{1}{2}\mathrm{Tr}\big(\sigma(t,x)\sigma(t,x)^\top \nabla^2 u(t,x)\big) + \mu(t,x)^\top \nabla u(t,x) \\ + f\big(t, x, u(t,x), \sigma(t,x)^\top \nabla u(t,x)\big) = 0\end{aligned} \tag{2.15}$$

with some specified terminal condition $u(T, x) = g(x)$. The following functions,

- $\sigma : [0, T] \times \mathbb{R}^d \to \mathbb{R}^{d\times d}$,
- $\mu : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$,
- $f : [0, T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$.

are all known. We seek the solution at $t = 0$ for some $x = \xi \in \mathbb{R}^d$.

We consider a n-dimensional stochastic process, $\{X_t\}_{t\in[0,T]}$, that satisfies (2.12). By using Itô's lemma on $Y_t = u(t, X_t)$, where $u \in C^2([0, T] \times \mathbb{R}^n, \mathbb{R})$ we obtain:

$$\begin{aligned} dY_t &= \frac{\partial u}{\partial t}(t, X_t)dt + \nabla u(t, X_t)^\top dX_t + \frac{1}{2}dX_t^\top \nabla^2 u(t, X_t)dX_t \\ &= \Big[\frac{\partial u}{\partial t}(t, X_t) + \nabla u(x, t)^\top \mu(t, X_t) + \frac{1}{2}\mathrm{Tr}\big(\sigma(t, X_t)^\top \sigma(t, X_t)\nabla^2 u(t, X_t)\big)\Big]dt \\ &\quad + \nabla u(t, X_t)^\top \sigma(t, X_t)dB_t. \end{aligned} \tag{2.16}$$

By inserting (2.15) into (2.16) we get

$$dY_t = -f\big(t, X_t, u(t, X_t), \sigma(t, X_t)^\top \nabla u(t, X_t)\big)dt + \nabla u(t, X_t)^\top \sigma(t, X_t) dB_t. \quad (2.17)$$

In other words, under certain regularity conditions on σ, μ, f, see Pardoux and Răşcanu (2014), we can find the solution of the (deterministic) partial differential equation in (2.15) at $t = 0$, by solving a pair of stochastic differential equations. The solution at $u(0, \xi) = Y_0$ corresponds to solving $(Y_t, Z_t) = (u(t, X_t), \sigma(t, X_t)^\top \nabla u(t, X_t))$ of the forward backward stochastic differential equations

$$X_t = \xi + \int_0^t \mu(s, X_s)ds + \int_0^t \sigma(s, X_s)dB_s \quad (2.18)$$

$$Y_t = g(X_T) + \int_t^T f\big(s, X_s, u(s, X_s), \sigma(s, X_s)^\top \nabla u(s, X_s)\big)ds \quad (2.19)$$
$$- \int_t^T \nabla u(s, X_s)^\top \sigma(s, X_s) dB_s.$$

2.2 Lévy Processes

Lévy processes are introduced, starting gently with the finite activity processes and proceeding to the infinite activity processes. Stochastic differential equations driven by Lévy processes are presented. At last, the numerical simulation of Lévy processes is discussed.

2.2.1 Introduction and Definition

Lévy processes are a natural extension from the Brownian motions. The Brownian motions, already discussed, are almost surely continuous, while Lévy processes may violate this property and have discontinuities. The possibility to model jumps allow us to model a greater range of phenomena. The Lévy processes appear frequently in quantitative finance, where empirical results show that the distribution of returns tend to have "fatter tails" than the normal distribution, which appears when a Brownian motion is used to model returns. Lévy processes can be used to model such fat tailed distribution.

Lévy Process

A stochastic process $\{X_t\}_{t\geq 0}$ with values in $\mathbb{R}^d$ and $X_0 = 0$ (almost surely) is a Lévy process if it satisfies the following three properties:

1. **Independence of increments**: For any $0 \leq t_0 \leq \cdots \leq t_n < \infty$, $X_{t_1} - X_{t_0}, X_{t_2} - X_{t_1}, \ldots, X_{t_n} - X_{t_{n-1}}$ are independent.

2. **Stationary increments**: $X_t - X_s$ is equal in distribution to X_{t-s} $\forall s \leq t$.

3. **Stochastic continuity**: $\lim_{h\to 0} \mathrm{P}[|X_{t+h} - X_t| > \epsilon] = 0$ for all $\epsilon > 0$.

The first condition is the same as the Brownian motion, while the second condition does not necessarily restrict the increments to have normally distributed increments, like the Brownian motion. The third condition does not necessarily mean that the sample paths of the processes are continuous, but it excludes processes that exhibit jumps at non-random times.

Both the standard Brownian motion and Brownian motions with drift and variance parameters are Lévy processes, in fact, they are the only Lévy process with continuous paths according to Lawler (2014).

2.2.2 Finite Activity Lévy Processes

The Poisson process and the compound Poisson process are two fundamental Lévy processes. They are both said to have finite activity in the sense that in every finitely sized time interval they will exhibit a finite number of jumps. There exists Lévy processes that can exhibit an infinite number of jumps on each proper time interval , which are introduced later.

Poisson Process

Let $\tau_1, \tau_2, \ldots$ be independent exponential random variables with parameter λ, that is, they have probability density function

$$f(\tau) = \lambda e^{-\lambda\tau}, \quad \tau \geq 0.$$

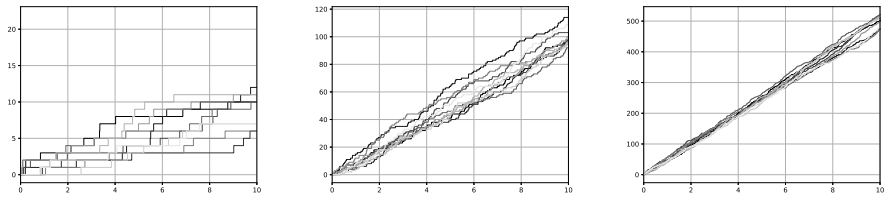
We further denote $T_n = \sum_{i=1}^{n} \tau_i$. Then

$$N_t = \#\{n \in \mathbb{N} : t \geq T_n\},$$

is a Poisson process with rate λ. The Poisson process has probability mass function

$$f(n) = \frac{(\lambda t)^n}{n!} e^{-\lambda}$$

and $\mathrm{E}[N_t] = \mathrm{Var}[N_t] = \lambda t$.



(a) $\lambda = 1$. (b) $\lambda = 10$. (c) $\lambda = 50$.

Figure 2.2: Poisson processes.

The Poisson process is a counting process. $\{N_t\}_{t\geq 0}$ counts the number of random times, T_n, which occur between 0 and t. It is the only counting process with stationary independent increments. **Figure 2.2** displays 10 sample paths for 3 different rates.

Compound Poisson Process

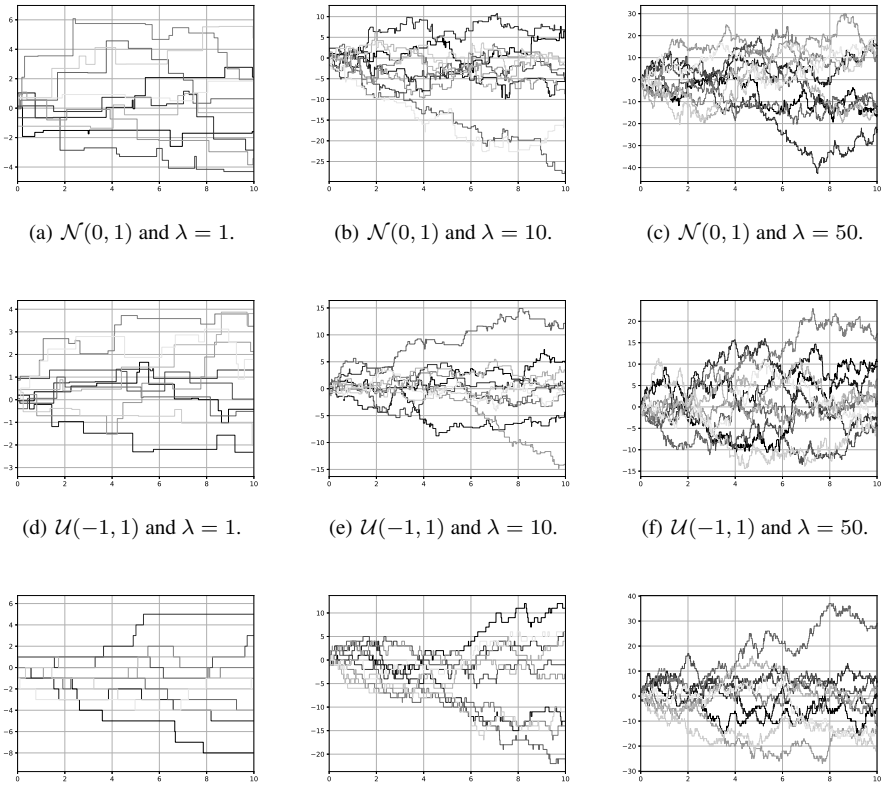
A compound Poison process with intensity $\lambda > 0$ and jump size distribution f is the stochastic process X_t defined by

$$X_t = \sum_{i=0}^{N_t} Y_i,$$

where N_t is a Poisson process with rate λ independent from $Y_1, Y_2, \ldots$ which are independent identically distributed with distribution f.

The following properties of the compound Poisson process can be deduced by using the law of total expectation:

$$\mathrm{E}[X_t] = \lambda t \mathrm{E}[Y_i] \quad \text{and} \quad \mathrm{Var}[X_t] = \lambda t(\mathrm{Var}[Y_i] + \mathrm{E}[Y_i]^2) \tag{2.20}$$



(a) $\mathcal{N}(0,1)$ and $\lambda = 1$. (b) $\mathcal{N}(0,1)$ and $\lambda = 10$. (c) $\mathcal{N}(0,1)$ and $\lambda = 50$.

(d) $\mathcal{U}(-1,1)$ and $\lambda = 1$. (e) $\mathcal{U}(-1,1)$ and $\lambda = 10$. (f) $\mathcal{U}(-1,1)$ and $\lambda = 50$.

(g) $\mathcal{U}\{-1,1\}$ and $\lambda = 1$. (h) $\mathcal{U}\{-1,1\}$ and $\lambda = 10$. (i) $\mathcal{U}\{-1,1\}$ and $\lambda = 50$.

Figure 2.3: Compound Poisson processes.

Figure 2.3 displays a few combinations of jump size distributions and intensities. Note the distinction between $\mathcal{U}(-1,1)$ and $\mathcal{U}\{-1,1\}$. $\mathcal{U}(-1,1)$ is the continuous uniform dis-

tribution on $(-1, 1)$, and $\mathcal{U}\{-1, 1\}$ is the discrete uniform distribution that takes values in $\{-1, 0, 1\}$ each with probability 1/3.

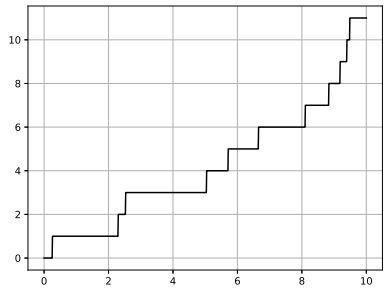
Compensated Processes

The compensated processes is a centred version of the process. We subtract a deterministic quantity from the process, such that the new process is a martingale. The quantity we subtract are the so-called compensator. For a Poisson process, $\{N_t\}_{t\geq 0}$, with parameter λ, the compensated Poisson process is then

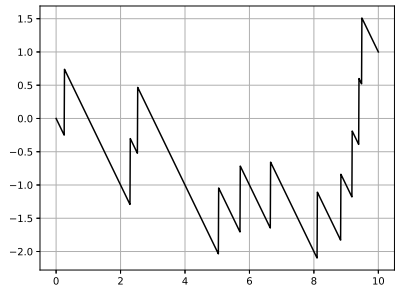
$$\widetilde{N}_t = N_t - \lambda t. \tag{2.21}$$

We can also introduce the concept of compensated compound Poisson processes. Consider a d-dimensional compound Poisson process, $\{X_t\}_{t\geq 0}$ with intensity λ and jump size distribution f. Then, by (2.20),

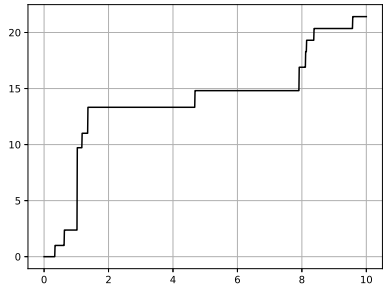
$$\widetilde{X}_t = X_t - \mu\lambda t. \tag{2.22}$$



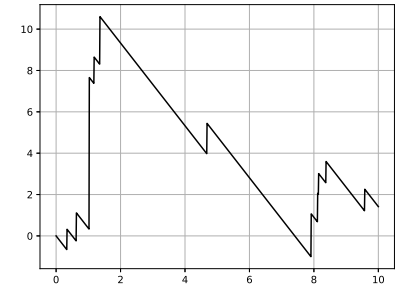
(a) Poisson process with $\lambda = 1$.



(b) Compensated Poisson process with $\lambda = 1$.



(c) Compound Poisson process with $\lambda = 1$ and Pareto distributed jump sizes with $\alpha = 2$.



(d) Compensated compound Poisson process with $\lambda = 1$ and Pareto distributed jump sizes with $\alpha = 2$.

Figure 2.4: Comparison of uncompensated and compensated processes.

Figure 2.4 displays the effect of the compensator. The left column shows a Poisson process, **Figure 2.4(a)**, and a compound Poisson process, **Figure 2.4(b)**, with Pareto distributed jump sizes with $\alpha = 2$. Both processes have intensity $\lambda = 1$. The right column shows the compensated version of the process to the left.

2.2.3 Infinite Activity Lévy Processes

Infinite Activity Lévy processes are a more general type of Lévy processes. We establish some notation and introduce some new concepts.

Jump Measure

The Poisson process and compound Poisson processes we have discussed so far can be expressed as

$$X_t = \int_{[0,t]\times\mathbb{R}^d} x J_X(ds \times dx). \tag{2.23}$$

This is a so-called Poisson integral, where J_X is the jump measure which describes the jumps of X_t. J_X is defined to be

$$J_X([t_1, t_2] \times A) = \#\{(t, \Delta X_t) \in [t_1, t_2] \times A\},$$

for every measurable set $A \subset \mathbb{R}^d$. In other words, $J_X([t_1, t_2] \times A)$ counts the number of jumps of X between t_1 and t_2 which size, ΔX_t, belong to A. It should be noted that $J_X(\omega, .)$ is a random measure, in the sense that it depends on ω. However, the dependence of ω is often omitted, as with random variables.

Lévy Measure

A jump measure is described by its intensity measure $\mu(dx \times dt)$ such that $\mathrm{E}[J_X(.)] = \mu(.)$. For a Lévy process, we have that $\mu(dt \times dx) = \nu(dx)dt$, where ν is the so-called Lévy measure and is a key concept when dealing with Lévy processes. For a d-dimensional Lévy process $\{X_t\}_{t\geq 0}$, the Lévy measure is

$$\nu(A) = \mathrm{E}[\#\{t \in [0, 1] : \Delta X_t \neq 0, \Delta X_t \in A\}]$$

where $A \subset \mathbb{R}^d$. In other words the expected number of jumps of size which belongs to A per unit time. The Poisson process and compound Poisson process both satisfy

$$\int_{\mathbb{R}^d} \nu(dx) < \infty$$

which is the criterion for finite activity.

Regularity Conditions

The finite activity processes have a finite number of jumps in each finite time interval. However, we can still allow the Lévy measure be infinite as long as $\nu(A)$ is finite for any

compact set A such that $0 \notin A$. In other words, we allow ν to blow up close to 0 such that the process have an infinite number of small jumps, where the convergence of the series of jumps relies on the following conditions:

$$\nu(\{0\}) = 0, \quad \int_{|x|>1} \nu(dx) < \infty, \quad \int_{|x|\leq 1} |x|^2 \nu(dx) < \infty. \tag{2.24}$$

Poisson Integral

We are not limited to have x as integrand in (2.23), we can define a stochastic process, $\{X_t(f)\}_{t\geq 0}$, with a more general integrand as

$$X_t(f) = \int_{[0,t]\times\mathbb{R}^d\setminus\{0\}} f(s,x) J_X(ds \times dx)$$

if

$$\int_{[0,t]\times\ \mathbb{R}^d\setminus\{0\}} |f(s,x)|\nu(ds)dx < \infty.$$

Compensated Measures

Similar to the finite activity compensated processes defined in (2.21) and (2.22), we can define a compensated jump measure for Lévy processes. For a jump measure J_X with intensity $\nu(dx)dt$, the compensated jump measure is

$$\widetilde{J}_X([t_1,t_2]\times A) = J_X([t_1,t_2]\times A) - \nu(A)(t_2 - t_1).$$

This allows us to express compensated Lévy processes as Poisson integrals,

$$\tilde{X}_t = \int_{[0,t]\times\mathbb{R}^d} x\widetilde{J}_X(ds \times dx).$$

Note that $\{\widetilde{X}_t\}_{t\geq 0}$ is a martingale.

Itô Isometry for Lévy Processes

An important property of the compensated Lévy processes is the Itô isometry. Let $\{X_t\}_{t\geq 0}$ be a d-dimensional Lévy process with jump measure J_X, which has intensity $\nu(dx)ds$. Then, if F satisfy

$$\int_{[0,T]\times A} \mathrm{E}\big|F(\omega,t,x)\big|^2 \nu(dx)ds < \infty,$$

for $A \subset \mathbb{R}^d$, then

$$\mathrm{E}\left[\Big|\int_{[0,T]\times A} F(\omega,s,x) J_X(ds\times dx)\Big|^2\right] = \int_{[0,T]\times A} \mathrm{E}\big|F(\omega,s,x)\big|^2 \nu(dx)ds.$$

Lévy-Itô Decomposition

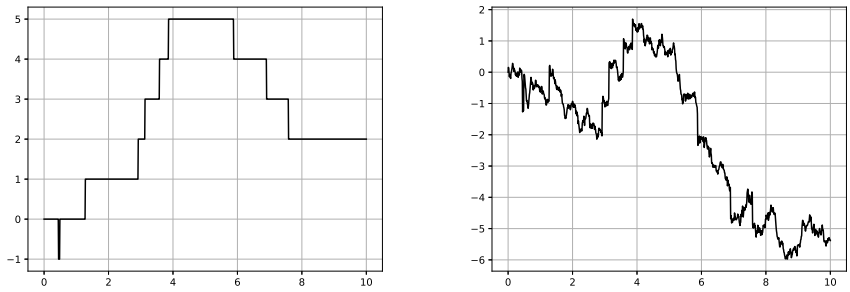
The sum of a Lévy process

$$X'_t = \int_{[0,t]\times\mathbb{R}^d} xJ_X(ds \times dx)$$

with intensity measure $\nu(dx)dt$ and a Brownian motion with drift and variance, $\{\mu t + AB_t\}_{t\geq 0}$, independent from $\{X'_t\}_{t\geq 0}$ defines another Lévy process

$$X_t = \mu t + AB_t + X'_t = \mu t + AB_t + \int_{[0,t]\times\mathbb{R}^d} xJ_X(ds \times dx) \tag{2.25}$$

where J_X has intensity $\nu(dx)dt$. **Figure 2.5(a)** displays a sample path of a compound Poisson process with $\lambda = 1$ and $\mathcal{U}\{-1, 1\}$ jump size distribution. **Figure 2.5(b)** shows the same sample path, but it is superpositioned with a sample path of a Brownian motion with $\mu = -1$ and $\sigma = 3$.



(a) Compound Poisson process with $\lambda = 1$ and $\mathcal{U}\{-1, 1\}$ jump size distribution.

(b) Superposition of compound Poisson process with $\lambda = 1$ and $\mathcal{U}\{-1, 1\}$ jump size distribution and Brownian motion with $\mu = -1$ and $\sigma = 3$.

Figure 2.5: Lévy process with and without Brownian motion component.

It turns out that every Lévy process can be expressed on a similar form as (2.25). Let $\{X_t\}_{t\geq 0}$ be a d-dimensional Lévy process. Then there exists a ν satisfying (2.24), a positive definite matrix $A \in \mathbb{R}^{d\times d}$ and a vector $\gamma \in \mathbb{R}^d$ such that

$$X_t = \gamma t + AB_t + \int_{|x|\in[1,\infty),s\in[0,t]} xJ_X(ds\times dx) + \int_{|x|\in(0,1),s\in[0,t]} x\widetilde{J}_X(ds\times dx). \tag{2.26}$$

Actually, the distribution of a Lévy process is uniquely determined by the triplet (A, ν, γ). This triplet is called the characteristic triplet or Lévy triplet.

This result implies that every Lévy process can be decomposed into a Brownian motion with drift, a compound Poisson process and an infinite superposition of independent compensated compound Poisson process.

Itô's formula for Lévy Driven Processes

Let $\{X_t\}_{t\geq 0}$ be a n-dimensional stochastic process driven by a Lévy process, which satisfy

$$X_t = \xi + \int_0^t \mu(s, X_s)ds + \int_0^t \sigma(s, X_s)dB_s + \int_{[0,t]\times E} \beta(X_{s-}, e)\widetilde{J}_X(ds \times de) \quad (2.27)$$

where $E = \mathbb{R}^d \setminus \{0\}$ and J_X has intensity $\nu(dx)ds$.

Then, for a function $u : [0, T] \times \mathbb{R}^d \to \mathbb{R}$ in $C^{1,2}$, we have that

$$\begin{aligned} u(t, X_t) =& u(T, X_T) - \int_t^T \nabla u(s, X_s)^\top \sigma(s, X_s)dB_s \\ &- \int_{[t,T]\times E} [u(s, X_{s-} + \beta(X_{s-}, e)) - u(s, X_{s-})]\tilde{J}_X(ds \times de) \\ &- \int_0^t \Big[\frac{\partial u}{\partial t}(s, X_s) + \frac{1}{2}\text{Tr}(\sigma(s, X_s)\sigma(s, X_s)^\top \nabla^2 u(s, X_s)) \\ &+ \int_E [u(s, X_s + \beta(X_s, e)) - u(s, X_s) - \nabla u(s, X_s)^\top \beta(X_s, e)]\nu(de)\Big]ds \end{aligned} \quad (2.28)$$

under certain regularity conditions as stated in Barles et al. (1997). Note that contrary to Itô 's lemma for drift-diffusion processes, we additionally have the compensated jump term on the last line of (2.28).

FBSDE for Lévy Driven Processes

Consider partial integro-differential equations (PIDEs) on the form

$$\frac{\partial u}{\partial t}(t, x) + \mathcal{L}u(t, x) + f(t, x, u, \sigma(t, x)^\top \nabla u(t, x)) = 0 \quad (2.29)$$

for $(t, x) \in [0, T) \times \mathbb{R}^n$ with terminal condition $u(T, x) = g(x)$. Here

$$\begin{aligned} \mathcal{L}u(t, x) =& \mu(t, x)^\top \nabla u(t, x) + \frac{1}{2}\text{Tr}(\sigma(t, x)\sigma(t, x)^\top \nabla^2 u(t, x)) \\ &+ \int_E [u(t, x + \beta(x, e)) - u(t, x) - \nabla u(t, x)^\top \beta(x, e))]\nu(de) \end{aligned}$$

There exists link between PIDEs as in (2.29) and Lévy driven stochastic processes, similar to the link between the semi-linear parabolic PDEs and Brownian motion driven FBSDEs. We apply Itô's lemma on (2.27) to get (2.28). We now have a set of forward and backward stochastic differential equations. Next step is to insert (2.29) and the terminal condition $u(T, x) = g(x)$ into (2.28) to get

$$\begin{aligned} u(t, X_t) =& g(X_T) - \int_t^T \nabla u(s, X_s)^\top \sigma(s, X_s)dB_s \\ &+ \int_t^T f(s, X_s, u(s, X_s), \sigma(s, X_s)^\top \nabla u(s, X_s))ds \\ &- \int_{[t,T]\times E} [u(s, X_{s-} + \beta(X_{s-}, e)) - u(s, X_{s-})]\widetilde{J}_X(ds \times de) \end{aligned} \quad (2.30)$$

and we can now solve (2.29) by solving the decoupled pair of equations (2.27) and (2.30). The solution at $u(0, \xi) = Y_0$ corresponds to solving (Y_t, Z_t, U_t) where

$$Y_t = u(t, X_t), \quad Z_t = \sigma(t, X_t)^\top \nabla u(t, X_t), \quad U_t = u(t, X_t + \beta(X_{t-}, e)) - u(t, X_t).$$

2.2.4 Generating Lévy Processes

The Lévy processes are numerically simulated for some cases in the following. The drift and the Brownian motion part in (2.26) are already covered. Our focus is limited to the jump component.

Finite Activity

We first consider the finite activity case, where

$$X_t = \int_{[0,t]\times\mathbb{R}^n} x J_X(ds \times dx) \tag{2.31}$$

is a n-dimensional Lévy process with intensity $\nu(dx)dt$ and finite activity. We can rewrite (2.31) as a compound Poisson process

$$X_t = \sum_{i=0}^{N_t} Y_i,$$

where $\{N_t\}_{t\geq 0}$ is a Poisson process with parameter $\int_{\mathbb{R}^n} \nu(dx) < \infty$ and Y_i are iid random variables with probability density function ν/λ.

Algorithm 2.1: Sampling a compound Poisson processes.

Input : T: The upper boundary of the desired time interval.
λ: The intensity of the compound Poisson Process.
f: The jump size distribution.

Output: $X_t(\omega_1)$: One sample path of the compound Poisson process on $[0, T]$.

$i \leftarrow 0$ /* Jump counter. */
$t_i \leftarrow 0$ /* Cumulative arrival times. */
/* Run while cumulative arrival times does not exceed max time. */
while $t_i < T$ **do**
  $i \leftarrow i + 1$
  $u \sim \mathcal{U}(0, 1)$ /* Sample uniform random variable. */
  $\tau_i \leftarrow -\frac{1}{\lambda}\ln(1 - u)$ /* Compute exponential random variable. */
  $t_i \leftarrow t_{i-1} + \tau_i$ /* Compute cumulative arrival time. */
  $Y_i \sim f$ /* Sample jump size. */
return $X_t = \sum_{\{i:t_i\leq t\}} Y_i$

Algorithm 2.1 describes how to generate sample paths of a compound Poisson process. We see that the difficulty of simulating such processes rely on the difficulty of simulating the jump sizes. In some cases, this is really straight forward and there exists numerous of numerical libraries capable to do this efficiently.

Infinite Activity

Since $\int_{\mathbb{R}^d} \nu(dx) = \infty$ in the infinite activity case, we can not proceed directly as above by reformulating the process to a compound Poisson process. We consider a Lévy process, $\{X_t\}_{t\geq 0}$ with the Lévy triplet $(0, \nu, 0)$, for a ν that satisfy $\int_{\mathbb{R}^d} \nu(dx) = \infty$ and (2.24), the process can be expressed as

$$X_t = \int_{|x|\in[1,\infty),s\in[0,t]} xJ_X(ds\times dx) + \int_{|x|\in(0,1),s\in[0,t]} x\widetilde{J}_X(ds\times dx) := X_t^{[1,\infty)} + \widetilde{X}_t^{(0,1)}. \tag{2.32}$$

The two terms in (2.32) are independent and can therefore be handled separately. The first term, $X_t^{[1,\infty)}$, consists of jump sizes with magnitude greater than 1 and has finite activity. It can therefore be turned into a compound Poisson process with intensity

$$\lambda_{[1,\infty)} = \int_{|x|\in[1,\infty)} \nu(dx)$$

and jump size distribution $\nu 1_{|x|\in[1,\infty)}/\lambda_{[1,\infty)}$.

The second term in (2.32) has infinite activity and therefore require some extra work. Asmussen and Glynn (2007) cover some alternatives on dealing with infinite activity processes. As previously stated, the compensated infinite activity process can be approximated arbitrarily well by a compensated finite activity process. The mean square error can be written as

$$\begin{aligned}
\mathrm{E}\big[|\widetilde{X}_t^{(0,1)} - \widetilde{X}_t^{[r,1)}|^2\big] &= \mathrm{E}\Big[\Big| \int_{|x|\in(0,r),s\in[0,t]} x\widetilde{J}_X(ds\times dx)\Big|^2\Big] \\
&= \int_{|x|\in(0,r),s\in[0,t]} x^\top x\nu(dx)ds \\
&= t\int_{|x|\in(0,1)} \underbrace{1_{|x|\in(0,r)}(x)x^\top x}_{F_r(x)}\,\nu(dx).
\end{aligned}$$

Since $|F_r(x)| \leq cx^\top x$ for some constant c, where

$$\int_{|x|\in(0,1)} |cx^\top x|\nu(dx) = |c|\int_{|x|\in(0,1)} x^\top x\nu(dx) < \infty$$

for all x satisfying $|x| \in (0,1)$ and $F_r(x) \to 0$ as $r \to 0$ we can use Lebesgue dominated convergence theorem, and conclude that

$$\mathrm{E}\big[|\widetilde{X}_t^{(0,1)} - \widetilde{X}_t^{[r,1)}|^2\big] \to 0$$

as $r \to 0$.

Ignore Small Jumps

The first strategy relies on simply ignoring the small jumps. That is, the infinite activity process is further decomposed into two independent parts,

$$\widetilde{X}_t^{(0,1)} = \widetilde{X}_t^{(0,r)} + \widetilde{X}_t^{[r,1)}, \quad r \in (0,1). \tag{2.33}$$

For a sufficiently small $r > 0$, $\widetilde{X}_t^{(0,r)}$ is negligible and can be ignored. Then

$$\widetilde{X}_t^{(0,1)} \approx \widetilde{X}_t^{[r,1)} = X_t^{[r,1)} - t \int_{|x| \in [r,1)} x\nu(dx)$$

which is a sum of a compound Poisson process, $X_t^{[r,1)}$, and a drift term. The compound Poisson process can be simulated by **Algorithm 2.1**.

Brownian Motion Approximation

A more refined strategy is rigorously covered in Asmussen and Rosiński (2001) and the multivariate case is covered in Cohen and Rosiński (2007). We again decompose as in (2.33), for the compensated process we have $\mathrm{E}[\widetilde{X}_t^{(0,r)}] = 0$ and

$$\Sigma_r = \mathrm{Var}[\widetilde{X}_1^{(0,r)}] = \mathrm{E}[\widetilde{X}_1^{(0,r)} \widetilde{X}_1^{(0,r)^\top}] = \int_{|x| \in (0,r)} xx^\top \nu(dx).$$

The isometry property has been used to compute Σ_r.

According to Cohen and Rosiński (2007), if Σ_r is non-singular and

$$\int_{x^\top \Sigma_r^{-1} x > k} x^\top \Sigma_r^{-1} x 1_{|x|<r}(x) \nu(dx) \to 0 \tag{2.34}$$

for all $k > 0$ when $r \to 0$, then

$$\Sigma_r^{-\frac{1}{2}} \widetilde{X}_t^{(0,r)} \xrightarrow{d} B_t$$

as $r \to 0$. Here $\Sigma_r^{-\frac{1}{2}}$ is the principal square root of Σ_r^{-1} and $\xrightarrow{d}$ denotes convergence in distribution.

An important class of Lévy processes are the processes where $\nu(dx)$ can be decomposed into a radial and angular component, $\nu(dx) = \phi(d\rho|u)\lambda(du)$ where λ is a finite measure on the $(d-1)$-dimensional unit sphere. For such processes the condition

$$\lim_{\epsilon \to 0} \frac{1}{\epsilon^2} \int_0^\epsilon \rho^2 \phi(d\rho|u) = \infty$$

implies that Σ_r is non-singular and (2.34) are satisfied.

This result is of importance since it allows us to approximate $\widetilde{X}_t^{(0,r)}$ by a Brownian motion which requires small computational effort to simulate. This approximation for a Lévy process with characteristic triplet (A, ν, γ) would be on the form

$$X_t \approx \Big(\gamma - \int_{|x|\in[r,1)} \nu(dx)\Big)t + AB_t + \Sigma_r^{\frac{1}{2}}\hat{B}_t + \sum_{i=1}^{N_t} Y_i$$

where B_t and $\hat{B}_t$ are independent Brownian motions, N_t is a Poisson process with rate $\lambda_{[r,\infty)} = \int_{|x|\in[r,\infty)} \nu(dx)$ and Y_i has distribution $\nu 1_{|x|\in[r,\infty)}/\lambda_{[r,\infty)}$. Alternatively, the two sum of the two Brownian motions can be written as

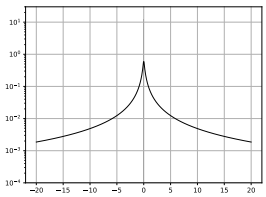
$$AB_t + \Sigma_r^{\frac{1}{2}}\widetilde{B}_t = (AA^\top + \Sigma_r)^{\frac{1}{2}} B_t^\star$$

where $B_t^\star$ is a Brownian motion independent of B_t and $\widetilde{B}_t$

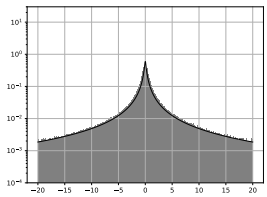
Since $\lambda_{[r,\infty)} \to \infty$ when $r \to 0$, the computational complexity for simulating the compound Poisson component is high when a small cut-off value r is chosen. The advantage of the Brownian motion approximation compared to ignoring small jumps, is that a higher cut-off value r can be chosen to achieve similar accuracy and hence reduce the computational complexity.

An Example: Simulating α-Stable Process

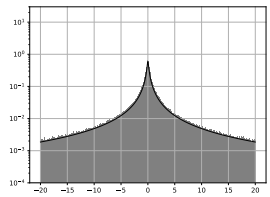
We consider the symmetric α-stable processes. That is, processes where the characteristic triplet is $(0, \nu, 0)$, where $\nu(dx) = c|x|^{-d-\alpha}dx$. Here d is the dimension of the process, $c > 0$ is a constant and $\alpha \in (0, 2)$. We simulate 10^6 realizations of X_1 for the two different strategies presented, either ignore jumps smaller than r, or approximate the small jumps as a Brownian motion (BM). The results are given in **Figure 2.6** for $\alpha = 0.5$ and in **Figure 2.7** for $\alpha = 1.5$. The solid line is the true probability density functions. The bars are normalized histograms of the simulated values. Note that the plots are log scaled.



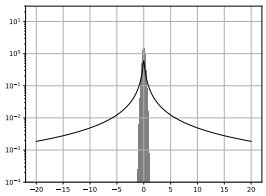
(a) Ignore. $r = 1$.



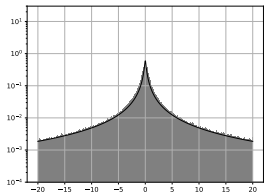
(b) Ignore. $r = 0.1$.



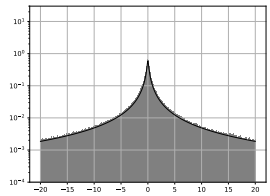
(c) Ignore. $r = 0.01$.



(d) BM. $r = 1$.

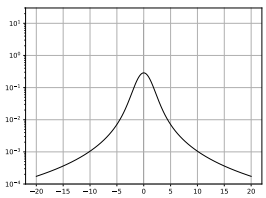


(e) BM. $r = 0.1$.

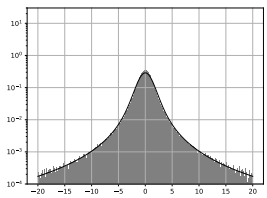


(f) BM. $r = 0.01$.

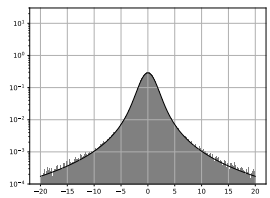
Figure 2.6: Histogram of α-stable process at $t = 1$ with $\alpha = 0.5$.



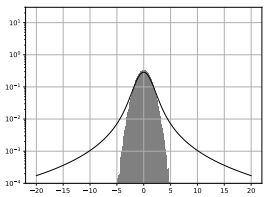
(a) Ignore. $r = 1$.



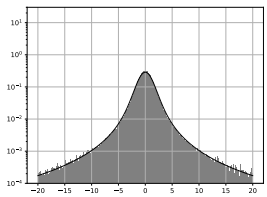
(b) Ignore. $r = 0.1$.



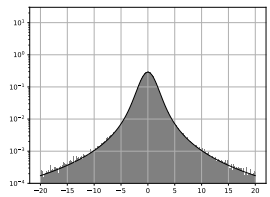
(c) Ignore. $r = 0.01$.



(d) BM. $r = 1$.



(e) BM. $r = 0.1$.



(f) BM. $r = 0.01$.

Figure 2.7: Histogram of α-stable process at $t = 1$ with $\alpha = 1.5$.

Chapter 3

Neural Networks

Neural networks play a key role as function approximators in the algorithms for solving PDEs to be presented. This chapter gives a brief introduction to neural networks mainly based on Goodfellow et al. (2016), which is a great, if not the best, introduction to deep learning.

The first section introduces a learning theoretical framework followed by an introduction to feedforward neural networks and activation functions. We then look into some basic concepts of optimization theory and lastly universal approximation theorems for neural networks are discussed.

3.1 Learning Theoretical Framework

In the general setting of supervised learning problems we have an input space $\mathcal{X}$ and an output space $\mathcal{Y}$. We assume that there is an unknown probability distribution $P(x, y)$ defined over $\mathcal{X} \times \mathcal{Y}$. The aim is to learn a function, often called hypothesis, $h : \mathcal{X} \to \mathcal{Y}$ based on independent samples from $P(x, y)$.

To be able to assess the goodness-of-fit of a hypothesis we need a loss function. A loss/cost function is a non-negative function, $\mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ where $\mathcal{Y}$ is the output space, such that

$$\ell(y, y) = 0 \text{ and } \ell(y, \hat{y}) > 0 \quad \forall \hat{y} \neq y.$$

One of the most common loss functions is the quadratic loss, $(\hat{y} - y)^2$ which arises in the ordinary linear regression problem.

The risk associated by the hypothesis h is defined as the expected value of a loss function

$$\mathcal{R}(h) = \mathrm{E}[\ell(y, h(x))] = \int_{\mathcal{X} \times \mathcal{Y}} \ell(y, h(x)) dP(x, y). \tag{3.1}$$

The goal is to minimize the risk, that is, make h be the best possible representation of $x_i \mapsto y_i$. We restrict ourselves to a fixed class of functions, $\mathcal{H}$ and solve

$$h^\star = \arg\min_{h \in \mathcal{H}} \mathcal{R}(h).$$

In practice $p(x, y)$ is rarely known and in some cases where $p(x, y)$ is known, the integral in (3.1) might not be possible to evaluate, so we typically estimate $\mathcal{R}(d)$ by Monte Carlo. Let $\mathcal{S}_N$ denote a random sample of N input-output pairs,

$$\mathcal{S}_N = \{(x_i, y_i) \sim P : i = 1, \ldots, N\}.$$

$\mathcal{S}_N$ is the so-called training set. We build a Monte-Carlo estimate of (3.1) based on the training set

$$\hat{\mathcal{R}}(h; \mathcal{S}_N) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, h(x_i)))$$

to achieve the empirical risk. The process of finding the minimizer of the empirical risk, i.e. solving

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \hat{\mathcal{R}}(h; \mathcal{S}_N).$$

is referred to as empirical risk minizimation in litterature.

3.2 Feedforward Neural Networks

Feedforward neural networks or multilayer perceptions aim to approximate some mapping $y = f^\star(x)$ by defining a mapping $y = f(x; \theta)$ and then learning the value of the parameter θ that gives the best function approximation. Such models are called feedforward because there is no feedback connection. That is, there is no connections that let the output of a model be fed back to itself. Extensions which allow for feedback connections are called recurrent neural networks.

Figure 3.1 gives an illustration of a fully connected neural network. This illustration consists of the input layer, one hidden layer and the output layer. The term *depth* refers to the amount of layers in the network, not including the input layer. In this case, the depth is 2. The term "deep learning" arises from the deep neural networks, where most literature consider neural networks with a depth larger than 3 to bee deep. Each layer consists of neurons, or nodes. The width of each layer is the amount of nodes in that layer. The illustrated network has a hidden layer with width p.

Further, each layer consists of an affine transformation and a non-linear function applied element-wise. The parameter θ consists of two components, W and b. Firstly, W, which is the linear transformation matrix, is called weight or kernel. Secondly, b is the translation term of the affine transformation. It is most frequently referred to as bias, but intercept is also common. In **Figure 3.1** we have $x \in \mathbb{R}^n$, $W^{(1)} \in \mathbb{R}^{p \times n}$ and $b^{(1)} \in \mathbb{R}^p$. This gives the affine transformation

$$u = W^{(1)}x + b^{(1)}$$

where $u \in \mathbb{R}^p$. Further, a non-linear function is applied elementwise u,

$$h_i = \phi^{(1)}(u_i) \quad \text{for } i = 1, \ldots, p.$$

This process is repeated as many times as the network is deep.

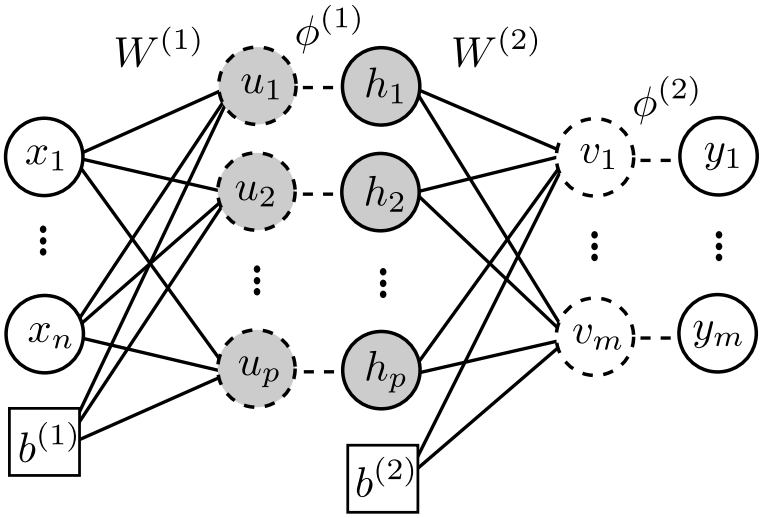


Figure 3.1: Fully connected feed forward neural network.

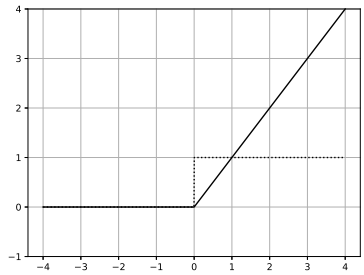
3.3 Activation Functions

	$\phi(x)$	$\phi'(x)$
Rectified Linear Unit (ReLU)	$\max(0, x)$	$\begin{cases} 0, & x < 0 \\ 1 & x > 0 \end{cases}$
Exponential Linear Unit (ELU)	$\begin{cases} \alpha(e^x - 1) & x < 0 \\ x & x > 0 \end{cases}$	$\begin{cases} \alpha e^x & x < 0 \\ 1 & x > 0 \end{cases}$
Hyperbolic Tangent	$\tanh(x)$	$\frac{1}{\cosh^2(x)}$
Sigmoid	$\frac{1}{1+e^{-x}}$	$\frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right)$

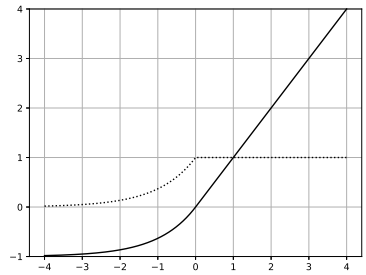
Table 3.1: Activation functions and their derivative.

As mentioned the neural networks consist of a non-linear activation function at each layer. If the activation function is linear, the network is not able to represent even simple functions as XOR, see Goodfellow et al. (2016). Common activation functions and their derivatives are given on **Table 3.1** and plotted on **Figure 3.2**. The activation function itself are the solid line, whereas the derivative is the dotted line.

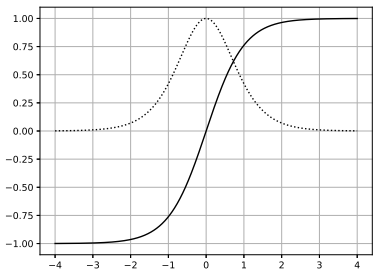
Sigmoid, and to some degree hyperbolic tangent, has traditionally been the most popular choice for hidden layers, but in recent years ReLU has gained popularity. There are mainly two reasons that ReLU has gained popularity. Firstly because both ReLU and the derivative of ReLU is extremely simple to compute. But most importantly, because of the phenomenon called vanishing gradient, discussed in Hochreiter (1991). The vanishing gradient problem causes the gradient to decrease exponentially as a function of the depth of the network and therefore slows down the training of the shallow layers.



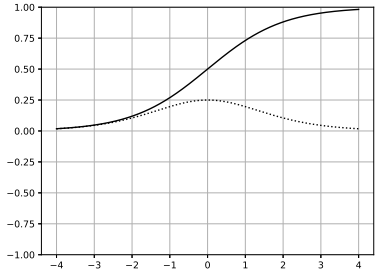
(a) Rectified linear unit.



(b) Exponential linear unit.



(c) Hyperbolic tangent.



(d) Sigmoid.

Figure 3.2: Activation functions and their derivative.

The choice of activation function on the output layer depends on the problem. If outputs on $\mathbb{R}^n$ is desired, the identity function could be used, whereas if outputs in $\mathbb{R}^n_+$ are desired, ReLU is a good choice. If the neural network approximates some classification task of two classes, it should produce the probability that the input is an instance of the classes. To satisfy Kolmogorovs axioms, the sigmoid should be used. For classification of more than two classes, the softmax is used. The softmax for N classes is defined as

$$\sigma_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}, \quad \text{for } i = 1, \ldots, n$$

3.4 Optimization

The optimization theory in the following is based on the lecture series of the "Deep Learning" course held at ETH Zürich, Pérez-Cruz (2018).

Gradient Descent

Gradient descent is the most basic minimization algorithm. We iteratively update $\hat{\mathcal{R}}(F_\theta; \mathcal{S}_N)$ based on the direction it is decreasing fastest as a function of θ. The direction of fastest decrease, is the direction of the negative gradient, so the algorithm is

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \hat{\mathcal{R}}(F_{\theta_n}; \mathcal{S}_N).$$

Here η is called the step size, or in the machine learning community, learning rate. θ_0 are set to a random value or set to a sensible value based on domain knowledge. The iterative updates are typically carried out until the updates $\nabla_\theta \hat{\mathcal{R}}(F_\theta; \mathcal{S}_N)_n$ are small, having reached a local minimum. Choosing the correct step size is important. A too small step size will make the algorithm converge slowly, and a too large step size will make the algorithm never converge. There exists conditions, such as the Wolfe conditions (see Nocedal and Wright (2006)) to select step size. Typically a binary search is carried out until a step size that satisfies the Wolfe conditions are found. This search requires several gradient and function evaluations, and are therefore expensive and rarely used in neural networks.

Stochastic Gradient Descent

As mentioned, computing the gradient of the risk is an expensive operation. We therefore resort to randomly splitting the full data set, $\mathcal{S}_N$ into N/K smaller batches of equal size K. We denote these smaller batches as $\mathcal{S}_{K,i}$. The size of the batches are frequently referred to as batch size. The stochastic term of stochastic gradient descent originates from the fact that the full data set is randomly partitioned. After partitioning, we proceed to update θ as follows:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \hat{\mathcal{R}}(F_{\theta_n}; \mathcal{S}_{K,i}),$$

which is equal to the non-stochastic gradient descent except that the gradient risk is only evaluated at the data pairs of $\mathcal{S}_{K,i}$. This process is executed for $i = 1, \ldots, N/K$, and we denote such a sweep through the full data set as one epoch. After the epoch is completed, the data is repartitioned into new randomly selected batches.

The notion behind stochastic gradient descent, is the fact that

$$\mathrm{E}\hat{\mathcal{R}}(F_\theta; \mathcal{S}_K) = \hat{\mathcal{R}}(F_\theta; \mathcal{S}_N)$$

allows us to update θ based on an unbiased estimate, without having to evaluate the derivative of the loss on the entire dataset.

(Stochastic) Gradient Descent with Momentum

The (stochastic) gradient descent algorithm can be extended to include momentum. The following algorithms will be described using the full data set $\mathcal{S}_N$, but could also be (and

are most frequently) executed by splitting the data set into batches. The notion behind introducing momentum is to avoid getting stuck in local minimums. We update the momentum:

$$m_{n+1} = \alpha m_n - (1-\alpha)\nabla_\theta \hat{\mathcal{R}}(F_{\theta_n}; \mathcal{S}_N)$$

with $\alpha \in [0, 1)$ and $m_0 = 0$. Then a correction for bias is performed:

$$\hat{m}_{n+1} = \frac{m_{n+1}}{1-\alpha^{n+1}}.$$

Finally we update θ,

$$\theta_{n+1} = \theta_n + \eta \hat{m}_{n+1}.$$

AdaGrad

The adaptive gradient algorithm, or AdaGrad, was proposed in Duchi et al. (2011) and "... dynamically incorporate knowledge of the geometry of the data observed in earlier iterations to perform more informative gradient-based learning". The algorithm is executed as follows:

$$g_{in} = \frac{\partial \hat{\mathcal{R}}(F_{\theta_n}; \mathcal{S}_N)}{\partial \theta_i}\Big|_{\theta=\theta_n}$$

and update

$$\theta_{i,n+1} = \theta_{i,n} - \frac{\eta}{\delta + \sqrt{\sum_{s=1}^{n} g_{is}^2}} \nabla_\theta \hat{\mathcal{R}}(F_{\theta_n}; \mathcal{S}_N)$$

where $\delta > 0$ is small.

ADAM

Adaptive moment estimation, or ADAM was proposed in Kingma and Ba (2014). ADAM aims to combine AdaGrad and the gradient descent with momentum. At first, we compute and store the gradient

$$g_{n+1} = \nabla_\theta \hat{\mathcal{R}}(F_{\theta_n}; \mathcal{S}_N)$$

and update the biased first and second raw moment estimate

$$\begin{aligned} m_{n+1} &= \beta_1 m_n + (1-\beta_1) g_{n+1} \\ v_{n+1} &= \beta_2 v_n + (1-\beta_2) g_{n+1} \odot g_{n+1} \end{aligned}$$

where $\odot$ denotes the elementwise multiplication. Further we correct both the moment estimates for bias

$$\begin{aligned} \hat{m}_{n+1} &= \frac{m_{n+1}}{1-\beta_1^{n+1}} \\ \hat{v}_{n+1} &= \frac{v_{n+1}}{1-\beta_2^{n+1}} \end{aligned}$$

where $m_0 = v_0 = 0$ and finally we update the parameters

$$\theta_{n+1} = \theta_n + \eta \frac{\hat{m}_{n+1}}{\sqrt{\hat{v}_{n+1}} + \epsilon}.$$

Here $\beta_1, \beta_2 \in [0, 1)$ and $\epsilon > 0$. Typical values are $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

3.5 Universal Approximation

The universal approximation theorem states that a feedforward neural network with a finite width can approximate continuous functions on a compact subset of $\mathbb{R}^n$, under some, relatively mild, assumptions on the activation function.

Classical results, such as Cybenko (1989), states that neural networks with sigmoidal activation functions and depth 2 are universal approximators on the unit hypercube. Hornik (1991) further extends the class of activation functions. Contrary to the depth-bounded universal approximation theorems, Lu et al. (2017) considers the ReLU activation function and provide a width-bounded universal approximation theorem.

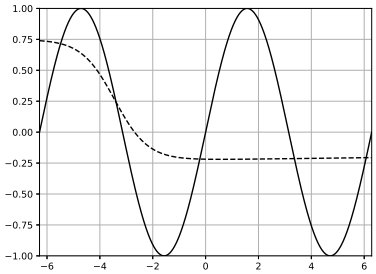
For any Lebesgue-integrable function $f : \mathbb{R}^n \to \mathbb{R}$ and any $\epsilon > 0$, there exists a fully-connected ReLU network with width $d_m \leq n + 4$, such that the function F_θ represented by this network satisfies

$$\int_{\mathbb{R}^n} |f(x) - F_\theta(x)| dx < \epsilon$$

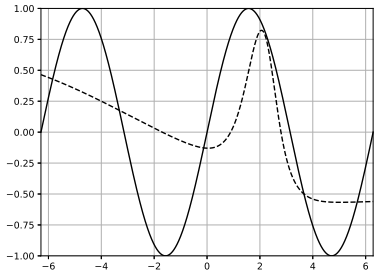
Figure 3.3 and **Figure 3.4** shows approximation of the sine wave for a few combinations of widths and depths. Hidden layer activation functions are sigmoid and ReLU respectively, whereas the identity function is used at the output layer. The dotted line is the neural net approximation whereas the solid line is the true value. The neural networks are trained using

$$\ell(y, \hat{y}) = |y - \hat{y}|^2 = |\sin(x) - F_\theta(x)|^2$$

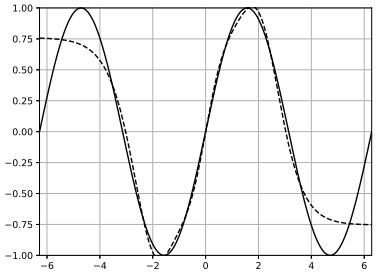
as the loss function with ADAM optimizer. The networks are trained in 10^5 iterations (which should be way more than sufficient), using a equidistant grid at $[-2\pi, 2\pi]$ with 10^5 points. It is evident that the shallow ReLU networks struggles to approximate the sinusoidal wave regardless of width, whereas it does a great job for a wide network width depth 3.



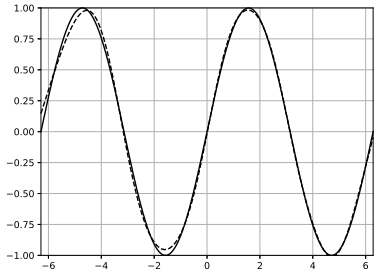
(a) Depth 2, width 2.



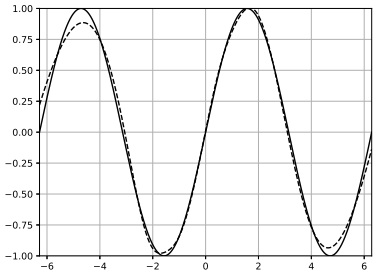
(b) Depth 3, width 2.



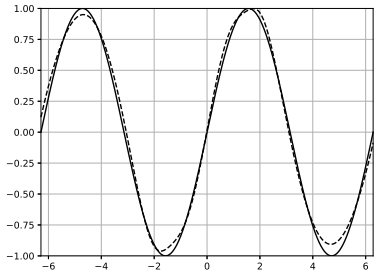
(c) Depth 2, width 8.



(d) Depth 3, width 8.

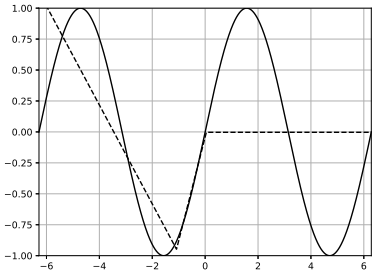


(e) Depth 2, width 128.

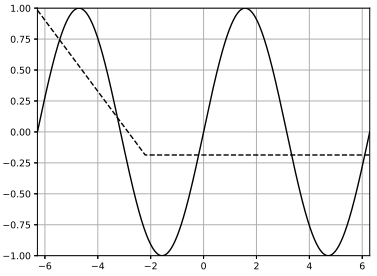


(f) Depth 3, width 128.

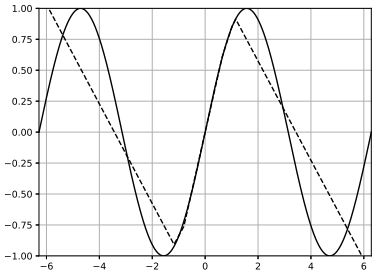
Figure 3.3: Neural network approximation of sin using sigmoid.



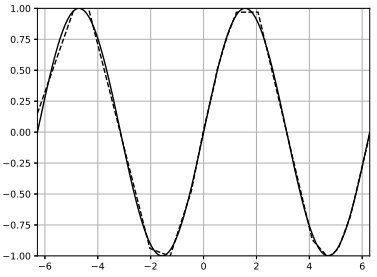
(a) Depth 2, width 2.



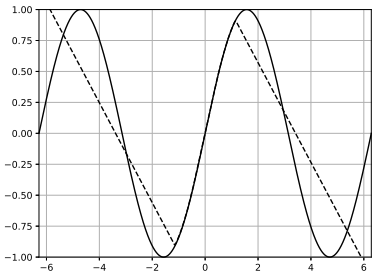
(b) Depth 3, width 2.



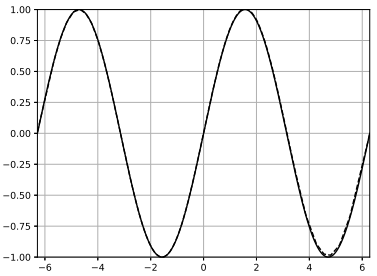
(c) Depth 2, width 8.



(d) Depth 3, width 8.



(e) Depth 2, width 128.



(f) Depth 3, width 128.

Figure 3.4: Neural network approximation of $\sin$ using ReLU.

Chapter 4

Deep Learning Algorithms for Solving Semi-Linear Parabolic PDEs

Recently, large developments have been made on solving semi-linear parabolic PDEs with deep learning, overcoming the curse of dimensionality. This chapter presents two of the pioneering approaches: The deep backward stochastic differential equation and the deep backward dynamic programming. The latter comes in two different versions. Additionally, the approaches are simulated and compared to each other.

The first section explains and presents each of the algorithms while the second section presents som numerical results for a selection of examples.

4.1 Presentation of Algorithms

Jiequn Han, Arnulf Jentzen and Weinan E present a pioneering algorithm to solve equations on the form (2.15) in E et al. (2017a), and in a more compact easier to read format in Han et al. (2017). The algorithm is named Deep BSDE. Huré et al. (2019) presents an approach similar to the Deep BSDE, called Deep Backward Dynamic Programming (DBDP). In this section, Deep BSDE will be presented as a brief summary of Han et al. (2017), followed by a presentation of two versions of the DBDP.

4.1.1 Deep BSDE

The aim is to compute $u(0, X_0)$ for some given $X_0 \in \mathbb{R}^d$ of (2.15). We discretize the time domain $[0, T] : 0 = t_0 < t_1 < ... < t_N = T$, and define X_t and $u(t, X_t)$ through Euler-Maruyama schemes based on (2.18) and (2.19) for $n = 0, \ldots, N-1$:

$$X_{t_{n+1}} \approx X_{t_n} + \mu(t_n, X_{t_n})\Delta t_n + \sigma(t_n, X_{t_n})\Delta B_n \tag{4.1}$$

and

$$u(t_{n+1}, X_{t_{n+1}}) \approx u(t_n, X_{t_n}) - f\big(t_n, X_{t_n}, u(t_n, X_{t_n}), \sigma(t_n, X_{t_n})^\top \nabla u(t_n, X_{t_n})\big)\Delta t_n + \nabla u(t_n, X_{t_n})^\top \sigma(t_n, X_{t_n}) \Delta B_n \tag{4.2}$$

where

$$\Delta t_n = t_{n+1} - t_n \quad \text{and} \quad \Delta B_n = B_{t_{t+1}} - B_{t_n} \sim \mathcal{N}(0, \Delta t_n).$$

The core of this algorithm is to turn the FBSDEs to a learning problem. This is done by parametrizing $x \mapsto \nabla u(t_n, x)^\top \sigma(t_n, x)$ for $n = 1, \ldots, N-1$ as neural networks

$$\nabla u(t_n, X_{t_n})^\top \sigma(t_n, X_{t_n}) = (\nabla u^\top \sigma)(t_n, X_{t_n}) \approx \mathcal{Z}(X_{t_n}; \theta_n).$$

Here, θ_n denotes the parameters of the neural network. These subnetworks, $\mathcal{Z}(.; \theta_n)$, are stacked based on (4.2) to form a deep neural network taking the sample paths $\{X_{t_n}\}_{n=0}^N$ and $\{B_{t_n}\}_{n=0}^N$ as input data and producing the final output $\hat{u}(\{X_{t_n}\}_{n=0}^N, \{B_{t_n}\}_{n=0}^N)$ as an approximation of $u(t_N, X_{t_N})$. Further, $u(0, X_0) \approx \theta_{u_0}$ and $\nabla u(0, X_0) \approx \theta_{\nabla u_0}$ are treated as parameters in the model. The total set of parameters is thus

$$\theta = \{\theta_{u_0}, \theta_{\nabla u_0}, \theta_1, \ldots, \theta_{N-1}\}.$$

We use the terminal condition evaluated at the last time step in (4.1) as our true outcome and use the quadratic loss. The risk, which we aim to minimize is then

$$\mathcal{R}(\hat{u}) = \mathrm{E}\Big[|g(X_{t_N}) - \hat{u}(\{X_{t_n}\}_{n=0}^N, \{B_{t_n}\}_{n=0}^N)|^2\Big]. \tag{4.3}$$

The algorithm is presented in **Algorithm 4.1**. In practice, (4.3) is minimized by a stochastic gradient descent-like algorithm.

Algorithm 4.1: Deep Backward Stochastic Differential Equation (Deep BSDE).

Input : X_0: Desired spatial point that the solution should be found at.
Output: Estimate of $u(t_0, X_0)$.
$\mathcal{U}_0 \leftarrow \theta_{u_0}$
$\mathcal{Z}(X_0; \theta_0) \leftarrow \theta_{\nabla u_0}$
for $n = 0, ..., N-1$ **do**
  $\Delta B_n \sim \mathcal{N}(0, \Delta t_n I)$
  $X_{t_{n+1}} \leftarrow X_{t_n} + \mu(t_n, X_{t_n})\Delta t_n + \sigma(t_n, X_{t_n})\Delta B_n$
  $\mathcal{U}_{n+1} \leftarrow \mathcal{U}_n + \mathcal{Z}(X_{t_n}; \theta_n)\Delta B_n - f(t_n, X_{t_n}, \mathcal{U}_n, \mathcal{Z}(X_{t_n}; \theta_n))\Delta t_n$
$\theta^\star \in \arg\min_\theta \mathrm{E}\Big[|\mathcal{U}_N - g(X_{t_N})|^2\Big]$
return $\theta^\star_{u_0}$

The algorithm can easily be extended to compute $u(t, X_0)$ for X_0 in some domain D. We let $u(t_0, .) = \mathcal{U}(.; \eta_0)$ be a neural network, and instead of considering a fixed X_0, we sample X_0 from some distribution in D and execute the rest of the algorithm as normal.

4.1.2 Deep Backward Dynamic Programming

Huré et al. (2019) presents two versions of an algorithm similar to the deep BSDE. The algorithm is called Deep Backward Dynamic Programming (DBDP).

Deep Backward Dynamic Programming 1

The first variant of the algorithms parametrizes $x \mapsto u(t_n, x)$ as a neural network $\mathcal{U}(x; \xi_n)$ for $n = 0, \ldots, N-1$, in addition to the parametrization of $\nabla u(t_n, x)^\top \sigma(t_n, x)$ as $\mathcal{Z}(x; \theta_n)$ familiar from Deep BSDE. Instead of optimizing the full set of neural network parameters simultaneously, the optimization problem is divided into easier to solve subproblems in a dynamic programming-like approach.

The algorithm iterates through the time steps backwards in time. Initially, we let $\mathcal{U}(.; \eta_N^\star) := g(.)$ and for each time step we minimize the L_2-norm, as in (2.1), of the Euler-Maruyama update:

$$\mathcal{R}_n(\eta_n, \theta_n) = \mathrm{E}\Big[|\mathcal{U}(X_{t_{n+1}}; \eta_{n+1}^\star) - \mathcal{U}(X_{t_n}; \eta_n) - \mathcal{Z}(X_{t_n}; \theta_n)\Delta B_n + f(t_n, X_{t_n}, \mathcal{U}(X_{t_n}; \eta_n), \mathcal{Z}(X_{t_n}; \theta_n))\Delta t_n|^2\Big].$$

with training data $(X_{t_n}, X_{t_{n+1}}, \Delta B_n)$. The algorithm is presented in **Algorithm 4.2**.

Algorithm 4.2: Deep Backward Dynamic Programming 1 (DBDP1)

Input : X_0: Desired spatial point that the solution should be found at.
Output: Estimate of $u(t_0, X_0)$.
$\mathcal{U}_N(.; \eta_N^\star) \leftarrow g(.)$
for $n = N-1, \ldots, 0$ **do**
  for $i = 0, \ldots, n$ **do**
    $\Delta B_i \sim \mathcal{N}(0, \Delta t_i I)$
    $X_{t_{i+1}} \leftarrow X_{t_i} + \mu(t_i, X_{t_i})\Delta t_i + \sigma(t_i, X_{t_i})\Delta B_i$
  $\mathcal{R}_n(\eta_n, \theta_n) \leftarrow \mathrm{E}\Big[|\mathcal{U}(X_{t_{n+1}}; \eta_{n+1}^\star) - \mathcal{U}(X_{t_n}; \eta_n) - \mathcal{Z}(X_{t_n}; \theta_n)\Delta B_n +$
  $f(t_n, X_{t_n}, \mathcal{U}(X_{t_n}; \eta_n), \mathcal{Z}(X_{t_n}; \theta_n))\ \Delta t_n|^2\Big]$
  $(\eta_n^\star, \theta_n^\star) \in \arg\min_{(\eta_n, \theta_n)} \mathcal{R}_n(\eta_n, \theta_n)$
return $\mathcal{U}(X_0;\ \eta_0^\star)$

Deep Backward Dynamic Programming 2

The next variant, DBDP2, is similar to DBDP1. The only difference is that $\sigma(t_n, x)^\top \nabla u(t_n, x)$ is no longer parametrized as a neural network, but as $\sigma(t_n, x)^\top \hat{\nabla} \mathcal{U}(x; \eta_n)$ where $\hat{\nabla}$ symbolizes numerical- or automatic differentiation.

4.2 Numerical Results

The implementation of Deep BSDE is available on GitHub, see Han (2019). The code for DBDP is not publicly available, so it has been implemented in Python using the TensorFlow 2.0 framework. All the numerical results have been produced on a MacBook Pro 2.2 GHz Intel Quad-Core i7 processor and 16GB of memory. This section presents the examples provided in Han et al. (2017) and E et al. (2017b) to show the diversity of equations that can be solved using the previously presented methods.

4.2.1 Test Cases

All the following results have a ReLU activation function and use the hyperparameters as provided in Han (2019) for the Deep BSDE method. A couple of days have been spent tweaking the hyperparameters of the two variants of DBDP to ensure a decent fit. Unless otherwise specified, the equations are solved in 100 spatial dimensions and the neural networks have two hidden layers, both with a width of 110. Further, the training batch is of size 64 and the validation batch sizes are 256. For all the results provided in the following, five independent runs are performed and the metrics are averaged over these five runs.

Hamilton-Jacobi-Bellman Linear Quadratic Control

The Hamilton-Jacobi-Bellmann (HJB) equations occurs in control theory, where the solution is the value function. By applying the general HJB equation for a classical linear-quadratic Gaussian control problem we get:

$$\frac{\partial u}{\partial t}(t,x) + \Delta u(t,x) - \lambda |\nabla u(t,x)|^2 = 0,$$

with some terminal condition $u(T,x) = g(x)$ and $\lambda > 0$. The explicit solution can be derived using Itô's formula,

$$u(t,x) = -\frac{1}{\lambda} \ln\Big(\mathrm{E}\big[\exp(-\lambda g(x + \sqrt{2}B_{T-t})\big]\Big),$$

where $\{B_t\}_{t\in[0,T]}$ is a Brownian motion. Consider now the terminal condition

$$g(x) = \ln\Big(\frac{1+|x|^2}{2}\Big),$$

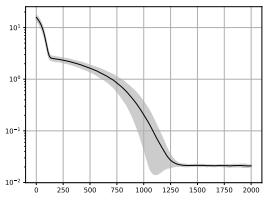
$\lambda = 1$ and $T = 1$. Then

$$u(0,0) = -\ln\Big(2\mathrm{E}\big[\frac{1}{1+2|B_1|^2}\big]\Big) \approx 4.5901,$$

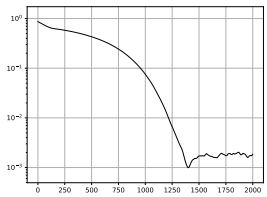
according to E et al. (2017a).

Numerical results are provided for a time discretization consisting of 20 time intervals. The results are shown in **Table 4.1** and **Figure 4.1**. Deep DBSE are trained using a learning rate of 0.01 over 2000 iterations. Both variants of DBDP are trained using 500 iterations

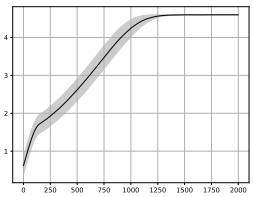
on the first and last time step and 200 iterations on the intermediate time steps. A decaying learning rate, starting as 10 and decreasing to 0.1 after 300 iterations is used. The vertical dotted lines in **Figure 4.1(d)** and **Figure 4.1(e)** display when a transition to a new time step is performed.



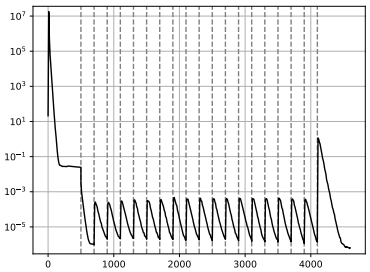
(a) Deep BSDE. Average loss $\pm$ standard error.



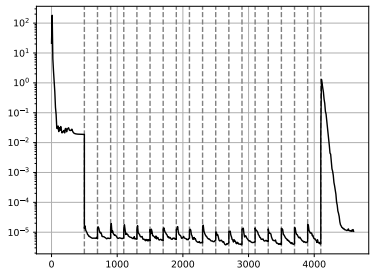
(b) Deep BSDE. Average relative approximation error.



(c) Deep BSDE. Average estimate of $u(0, X_0)$ $\pm$ standard error.



(d) DBDP1. Average loss.



(e) DBDP2. Average loss.

Figure 4.1: Hamilton-Jacobi-Bellman linear quadratic controller.

Algorithm	**Estimate**	**Standard error**	**Relative error**	**Time trained** (s)
Deep BSDE	4.59859	$1.57 \cdot 10^{-3}$	$1.85 \cdot 10^{-3}$	90.2
DBDP1	4.59568	$3.97 \cdot 10^{-3}$	$1.39 \cdot 10^{-3}$	296.2
DBDP2	4.60017	$1.91 \cdot 10^{-3}$	$2.20 \cdot 10^{-3}$	244.2

Table 4.1: Hamilton-Jacobi-Bellman linear quadratic controller.

Allen-Cahn

We consider the Allen-Cahn equation. The Allen-Cahn equation is a reaction-diffusion equation that is used in physics to model phase separation. The Allen-Cahn equation with the double-well potential is on the form

$$\frac{\partial u}{\partial t}(t,x) - \Delta u(t,x) - u(t,x) + [u(t,x)]^3 = 0, \quad t \in (0,T]. \tag{4.4}$$

It is actually an initial value problem with $u(x,0) = g(x)$, but by applying a time transformation $t \mapsto T - t$, we are able to turn it to a terminal value problem and can use the deep learning algorithms to solve it. After the time transformation, (4.4) reads

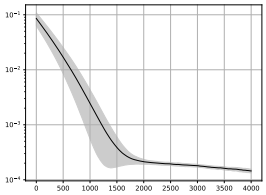
$$\frac{\partial u}{\partial t}(t,x) - \Delta u(t,x) + u(t,x) - [u(t,x)]^3, \quad t \in [0,T)$$

and $u(x,T) = g(x)$. We use

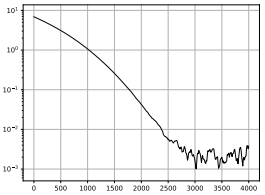
$$g(x) = \frac{1}{2 + \frac{2}{5}|x|^2},$$

$T = 0.3$ and $x \in \mathbb{R}^{100}$. Then, $u(0,0) \approx 0.052802$ which is computed by the Branching diffusion method in E et al. (2017a).

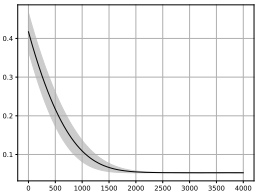
We consider a discretization of 20 time intervals. For Deep BSDE the learning rate is set to $5 \cdot 10^{-3}$ and the model is trained for 4000 iterations. For both variants of DBDP, 500 iterations are used to train on the $(N-1)$'th timestep, 400 iterations are used on the 0'th timestep and 200 iterations are used on the other time steps. A decaying learning rate is used, initially setting it to 1, reducing it to 0.1 after 200 iterations, then 0.01 after a total of 500 iterations and finally to 0.001 after a total of 1000 iterations. The results are displayed in **Table 4.2** and **Figure 4.2**.



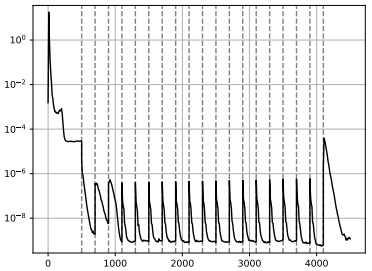
(a) Deep BSDE.
Average loss $\pm$ standard error.



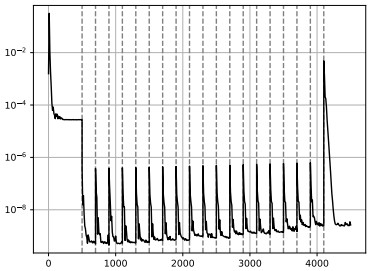
(b) Deep BSDE.
Average relative approximation error.



(c) Deep BSDE.
Average estimate of $u(0, X_0) \pm$ standard error.



(d) DBDP1. Average loss.



(e) DBDP2. Average loss.

Figure 4.2: Allen-Cahn.

Algorithm	**Estimate**	**Standard error**	**Relative error**	**Time trained** (s)
Deep BSDE	0.05275988	$2.45 \cdot 10^{-4}$	$3.40 \cdot 10^{-3}$	171.6
DBDP1	0.05283268	$2.38 \cdot 10^{-4}$	$4.45 \cdot 10^{-3}$	274.4
DBDP2	0.05292996	$2.61 \cdot 10^{-4}$	$4.82 \cdot 10^{-3}$	239.6

Table 4.2: Allen-Cahn.

Black-Scholes Equation with Default Risk

The Black-Scholes equation is used to price European options. We consider a variant of the Black-Scholes equation where the default risk is taken into account:

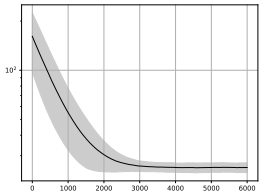
$$\frac{\partial u}{\partial t}(t,x)+\bar{\mu}x^\top \nabla u(t,x)+\frac{\bar{\sigma}^2}{2}\sum_{i=1}^{n} x_i^2 \frac{\partial^2 u}{\partial x_i^2}(t,x)-(1-\delta)Q(u(t,x))u(t,x)-Ru(t,x)=0,$$

where $\delta \in [0,1)$, $\bar{\mu} \in \mathbb{R}$, $\bar{\sigma}^2 \in (0,\infty)$ and $R \in \mathbb{R}$. Further,

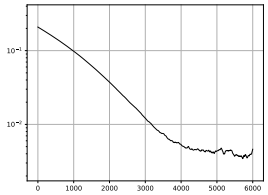
$$Q(y)=\begin{cases}\gamma^h, & -\infty < y < v^h \\ \frac{\gamma^h-\gamma^l}{v^h-v^l}(y-v^h)+\gamma^h, & v^h \le y < v^l \\ \gamma^l, & v^l \le y < \infty\end{cases}$$

where $v^h < v^l$ and $\gamma^h > \gamma^l$. The terminal condition is $g(x)=\min\{x_1,\dots,x_n\}$.

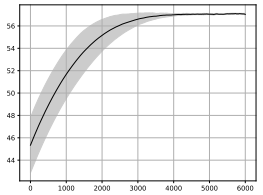
We perform numerical tests with $T=1$, $X_0=(100,\dots,100)$, $\delta=2/3$, $R=0.02$, $\bar{\mu}=0.02$, $\bar{\sigma}=0.2$, $v^h=50$, $v^l=70$, $\gamma^h=0.2$, $\gamma^l=0.02$ and a discretization of 40 time intervals. According to Han et al. (2017), $u(0,X_0)\approx 57.300$ Deep BSDE is trained with learning rate 0.008 for 6000 iterations. Both variants of DBDP are trained for 400 iterations on the last time step, 500 iterations on the first time step and 200 iterations on the intermediate time steps. A decaying learning rate is used, starting off with a learning rate of 100 and then decaying to 1 after 200 iterations. The results are displayed in **Table 4.3** and **Figure 4.3**.



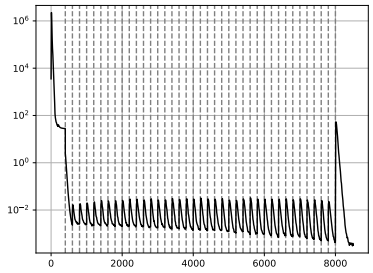
(a) Deep BSDE.
Average loss $\pm$ standard error.



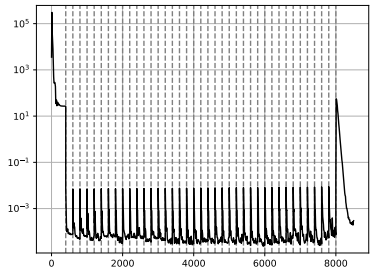
(b) Deep BSDE.
Average relative approximation error.



(c) Deep BSDE.
Average estimate of $u(0, X_0)$ $\pm$ standard error.



(d) DBDP1. Average loss.



(e) DBDP2. Average loss.

Figure 4.3: Black-Scholes equation with default risk.

Algorithm	**Estimate**	**Standard error**	**Relative error**	**Time trained** (s)
Deep BSDE	57.03612	$5.21 \cdot 10^{-2}$	$5.61 \cdot 10^{-3}$	508.8
DBDP1	57.09136	$10.51 \cdot 10^{-2}$	$3.64 \cdot 10^{-3}$	2779.8
DBDP2	57.04286	$12.05 \cdot 10^{-2}$	$4.49 \cdot 10^{-3}$	1448.8

Table 4.3: Black-Scholes equation with default risk.

Black-Scholes with Different Interest Rates for Borrowing and Lending

The standard variant of Black-Scholes assumes that the interest rate for borrowing and lending are equal. This is rarely the case. A variant which incorporates different interest rates for borrowing and lending is

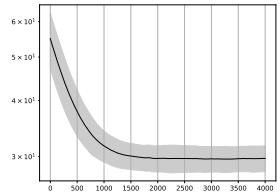
$$\frac{\partial u}{\partial t}(t,x) + \bar{\mu} x^\top \nabla u(t,x) + \frac{\bar{\sigma}^2}{2} \sum_{i=1}^{n} x_i^2 \frac{\partial^2 u}{\partial x_i^2}(t,x) - (\bar{\mu} - R^l) \sum_{i=1}^{n} x_i \frac{\partial u}{\partial x_i}(t,x) \\ - R^l u(t,x) + (R^b - R^l) \max\Big\{0, \sum_{i=1}^{n} x_i \frac{\partial u}{\partial x_i}(t,x) - u(t,x)\Big\} = 0$$

with terminal condition

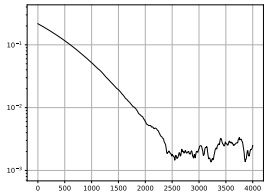
$$g(x) = \max\{\max_{1 \le i \le n} x_i - 120, 0\} - 2\max\{\max_{1 \le i \le n} x_i - 150, 0\}.$$

Numerical results are displayed in **Table 4.4** and **Figure 4.4** for $X_0 = (100, \ldots, 100)$, $T = 0.5$ and a discretization in time with 20 intervals. The results of Deep BSDE is produced by training for 4000 iterations with a learning rate of 0.005. DBDP is trained for 500 iterations on the first and the last time step and 200 iterations in the intermediate time steps. A decaying learning rate is used, initially 10, then decaying to 1 after 300 iterations and lastly decaying to 0.1 after a total of 750 iterations. According to E et al. (2017a), an approximate solution for $u(0, X_0)$ using multilevel Picard approximation methods is 21.299.

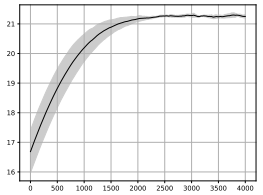
An important observation here is that both variants of DBDP performs really poor. Extensive tweaking of hyperparameters were performed, but DBDP still kept converging to the same, wrong, value.



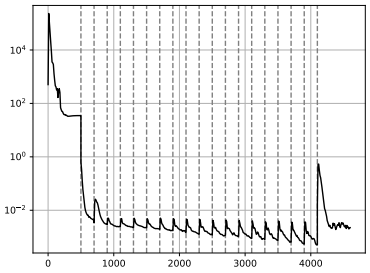
(a) Deep BSDE.
Average loss $\pm$ standard error.



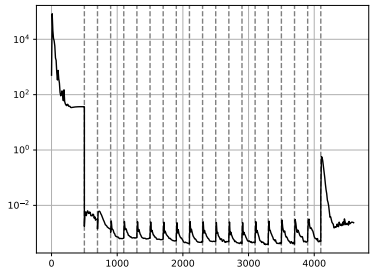
(b) Deep BSDE.
Average relative approximation error.



(c) Deep BSDE.
Average estimate of $u(0, X_0)$ $\pm$ standard error.



(d) DBDP1. Average loss.



(e) DBDP2. Average loss.

Figure 4.4: Black-Scholes equation with different interest rates.

Algorithm	**Estimate**	**Standard error**	**Relative error**	**Time trained** (s)
Deep BSDE	21.2518	$5.0 \cdot 10^{-2}$	$2.50 \cdot 10^{-3}$	188.6
DBDP1	19.0905	$2.9 \cdot 10^{-2}$	$104 \cdot 10^{-3}$	297.4
DBDP2	19.2671	$3.04 \cdot 10^{-2}$	$95 \cdot 10^{-3}$	284.6

Table 4.4: Black-Scholes eqution with different interest rates.

A PDE with Quadratically Growing Derivatives

We let

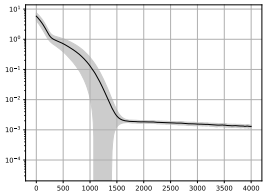
$$\psi(t,x) = \sin\left([T-t+\frac{1}{n}|x|^n]^{0.4}\right)$$

and consider the equation

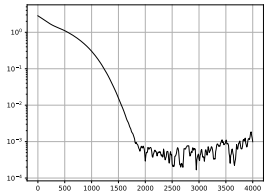
$$\frac{\partial u}{\partial t}(t,x) + |\nabla u(t,x)|^2 + \frac{1}{2}\Delta u(t,x) - \frac{\partial \phi}{\partial t}(t,x) + |\nabla \psi(t,x)|^2 + \frac{1}{2}\Delta \psi(t,x) = 0 \quad (4.5)$$

with $g(x) = \psi(x,T)$. Naturally, the analytical solution of (4.5) is $u(x,t) = \psi(x,t)$.

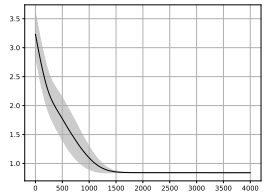
Numerical results are provided in **Table 4.5** and **Figure 4.5** for $X_0 = (0,\dots,0)$, $T = 1$ and a discretization of 30 intervals in time. Deep BSDE is trained using a learning rate of 0.005 over 4000 iterations. Both variants of DBDP use 500 iterations in the last time step, 300 iterations in the intermediate time steps and 700 iterations in the last time step. A decaying learning rate is used, initially at 10, then decaying to 1 after 300 iterations and to 0.1 after a total of 600 iterations.



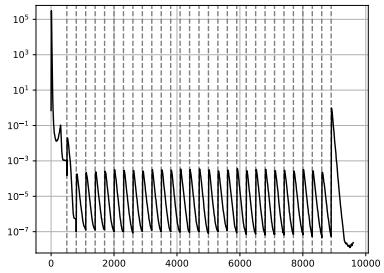
(a) Deep BSDE.
Average loss $\pm$ standard error.



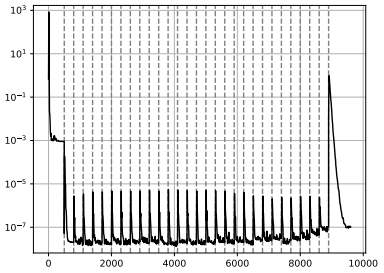
(b) Deep BSDE.
Average relative approximation error.



(c) Deep BSDE.
Average estimate of $u(0, X_0) \pm$ standard error.



(d) DBDP1. Average loss.



(e) DBDP2. Average loss.

Figure 4.5: PDE with quadratically growing derivative.

Algorithm	**Estimate**	**Standard error**	**Relative error**	**Time trained** (s)
Deep BSDE	0.8409392	$7.25 \cdot 10^{-4}$	$9.89 \cdot 10^{-4}$	268.6
DBDP1	0.8402152	$9.31 \cdot 10^{-4}$	$14.96 \cdot 10^{-4}$	985.8
DBDP2	0.8411672	$7.20 \cdot 10^{-4}$	$7.19 \cdot 10^{-4}$	608.2

Table 4.5: PDE with quadratically growing derivative.

A Time-Dependent Reaction-Diffusion-Type PDE with an Oscillating Solution

The equation

$$\frac{\partial u}{\partial t}(t,x)+\frac{1}{2}\Delta u(t,x)+\min\left\{1, u(t,x)-\left[\kappa-1-\sin\left(\lambda\sum_{i=1}^{n}x_i\right)\exp\left(\frac{\lambda^2 n(t-T)}{2}\right)\right]^2\right\}=0$$

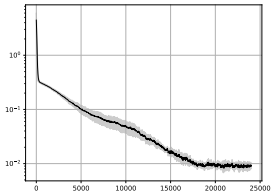
with terminal condition

$$g(x)=1+\kappa+\sin\left(\lambda\sum_{i=1}^{n}x_i\right)$$

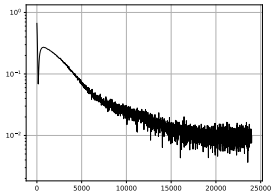
has explicit solution

$$u(t,x)=1+\kappa+\sin\left(\lambda\sum_{i=1}^{n}x_i\right)\exp\left(\frac{\lambda^2 n(t-T)}{2}\right).$$

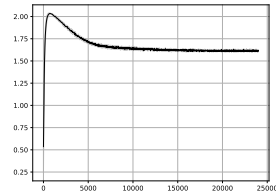
We run numerical test with $\lambda=1/\sqrt{n}$, $T=1$ and a time discretization of 30 intervals. Deep BSDE is trained for 2400 iterations with a learning rate of 0.01. Both variants of DBDP is trained for 1000 iterations on the last time step, 200 for the intermediate time steps and 400 iterations on the first time step. A decaying learning rate is used, starting with 10, then decaying to 1 after 100 iterations, to 0.1 after a total of 400 iterations and to 0.02 after a total of 700 iterations. The results are shown in **Table 4.6** and **Figure 4.6**.



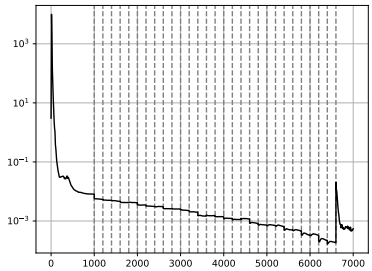
(a) Deep BSDE.
Average loss $\pm$ standard error.



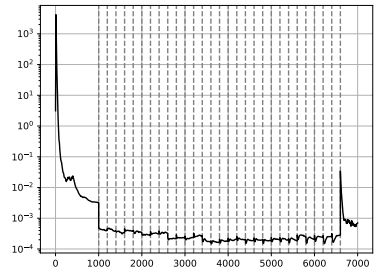
(b) Deep BSDE.
Average relative approximation error.



(c) Deep BSDE.
Average estimate of $u(0, X_0) \pm$ standard error.



(d) DBDP1. Average loss.



(e) DBDP2. Average loss.

Figure 4.6: Time-dependent reaction-diffusion-type PDE.

Algorithm	**Estimate**	**Standard error**	**Relative error**	**Time trained** (s)
Deep BSDE	1.61235	$4.92 \cdot 10^{-3}$	$7.72 \cdot 10^{-3}$	1510.2
DBDP1	1.61494	$10.65 \cdot 10^{-3}$	$9.34 \cdot 10^{-3}$	644.6
DBDP2	1.60723	$14.29 \cdot 10^{-3}$	$8.04 \cdot 10^{-3}$	477.6

Table 4.6: Time-dependent reaction-diffusion-type PDE.

4.2.2 Discussion

The authors of DBDP claim that the two variants are less prone to be trapped in local minima, contrary to Deep BSDE. While this might be true for some cases, the opposite can also occur, as seen in **Table 4.4**. The examples above might be unfair in favour of DBDP, since they have all been picked from articles discussing Deep BSDE.

A benefit of the DBDP method is that it enables the networks to be pre-trained; to initialize the neural networks with the same parameters as the preceding neural network. In theory, this trick allows more efficient training. Remember that we are iterating backwards in time. The neural networks of the last time step, $N-1$, should be trained for many iterations. Then, by using the pre-training trick, the preceding neural networks can be trained for way less iterations. This is displayed on the average loss plots in the previous section. The first time step, if $u(t_0, X_0)$ is not a neural network, but a parameter with X_0 constant, must however be trained for more iterations as shown in the previous section. In the case where X_0 in some region D is considered and $u(t_0, X_0)$ is approximated as a neural network, the issue with the 0'th step is not present. In practice having a different amount of iterations for the various time steps, adds more degrees of freedoms to the hyperparameters, making it increasingly difficult to tune.

Another advantage of the DBDP is the fact that it enables a finer discretization on the Euler-Maruyama schemes. Too many time steps in the Deep BSDE algorithm would cause memory issues since there would be too many neural network parameters to be trained simultaneously. DBDP on the other hand, would not run into the same issues, since the parameters of the neural network(s) of only one time step is optimized simultaneously. However, Deep BSDE have been able to solve Allen-Cahn with 200 time steps without issues.

Chapter 5

A Deep Learning Algorithm for Solving Fractional Laplace Equations

We turn our focus to an equation on the form (2.29), namely equations involving the fractional Laplace operator. This chapter proposes a numerical scheme to solve the equation.

The first section derives an approximation of the fractional Laplace operator and presents a numerical scheme similar to DBDP to solve such equations. The second section provides numerical results on a test example.

5.1 Presentation of Algorithm

We develop an algorithm that is mainly inspired by DBDP, in particular the second variant. The algorithm extends the ideas of DBDP to an algorithm that is capable of solving equations involving the fractional Laplace operator.

Fractional Laplace Operator

The fractional Laplace operator, $-(-\Delta)^{\alpha/2}$, in $\mathbb{R}^d$ with $\alpha \in (0,2)$, can be written by a variant of the singular integral definition, by Kwaśnicki (2015):

$$-(-\Delta)^{\alpha/2}u(t,x) = \int_{\mathbb{R}^d\setminus\{0\}} \Big[u(t,x+z) - u(t,x) - z^\top \nabla u(t,x) 1_{|z|<1}\Big]\nu_\alpha(z)dz$$

where

$$\nu_\alpha(dx) = \underbrace{\frac{2^\alpha \Gamma(\frac{d+\alpha}{2})}{\pi^{d/2}|\Gamma(-\frac{\alpha}{2})|}}_{c_{\alpha,d}} \frac{1}{|x|^{d+\alpha}}dx, \quad \alpha \in (0,2). \tag{5.1}$$

We have that

$$-(-\Delta)^{\alpha/2}u(t,x) = \int_{|z|<r}\Big[u(t,x+z)-u(t,x)-z^\top\nabla u(t,x)1_{|z|<1}\Big]\nu_\alpha(dx) + \int_{|z|>r}\Big[u(t,x+z)-u(t,x)-z^\top\nabla u(t,x)1_{|z|<1}\Big]\nu_\alpha(dz) \quad (5.2)$$

for some $r \in \mathbb{R}$. By assuming $r < 1$ we Taylor expand in $u(t, x + z)$ around x:

$$\begin{aligned} u(t,x+z) &= u(t,x) + z^\top\nabla u(x) + \int_x^{x+z}(x+z-y)^\top\nabla^2 u(t,y)dy \\ &= u(t,x) + z^\top\nabla u(t,x) + \frac{1}{2}z^\top\nabla^2 u(t,\xi)z, \quad \xi \in (x,x+z). \end{aligned} \quad (5.3)$$

We now insert (5.3) in (5.2), and get

$$\int_{|z|<r}\Big[u(t,x)-u(t,x)-z^\top\nabla u(t,x)1_{|z|<1}\Big]\nu_\alpha(dz) = \frac{1}{2}\int_{|z|<r} z^\top\nabla^2 u(t,\xi)z\nu_\alpha(dz), \quad (5.4)$$

for $\xi \in (x, x + z)$. We further assume $r \ll 1$ and hence $\nabla^2 u(t, \xi) \approx \nabla^2 u(t, x)$. We therefore approximate (5.4) as

$$\begin{aligned} \frac{1}{2}\int_{|z|<r} z^\top\nabla^2 u(t,x)z\nu_\alpha(dz) &= \frac{1}{2}\int_{|z|<r}\mathrm{Tr}\big(z^\top\nabla^2 u(t,x)z\big)\nu_\alpha(dz) \\ &= \frac{1}{2}\mathrm{Tr}\Big(\nabla^2 u(t,x)\int_{|z|<r}\frac{zz^\top}{|z|^{d+\alpha}}dz\Big) \\ &= \frac{1}{2}c_{\alpha,d}\underbrace{\frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}\frac{r^{2-\alpha}}{2-\alpha}}_{k_{\alpha,d,r}}\Delta u(t,x). \end{aligned}$$

Since ν_α is even, we have that

$$\int_{|z|>r} z^\top\nabla u(t,x)1_{|z|<1}\nu_\alpha(dz) = 0$$

such that the Fractional Laplace operator is approximated as

$$-(-\Delta)^{\alpha/2}u(t,x) \approx \int_{|z|>r}\Big[u(t,x+z)-u(t,x)\Big]\nu_\alpha(dx) + \frac{1}{2}c_{\alpha,d}k_{\alpha,d,r}\Delta u(t,x).$$

where we recognize the latter term as

$$\frac{1}{2}\mathrm{Tr}\Big(\sigma(t,x)^\top\sigma(t,x)\nabla^2 u(t,x)\Big)$$

for $\sigma(t,x) = \sqrt{c_{\alpha,d}k_{\alpha,d,r}}I$.

Numerical Scheme

We aim to find the solution $u(0, X_0)$, for some $X_0 \in \mathbb{R}^d$, for equations on the form

$$\frac{\partial u}{\partial t}(t,x) - (-\Delta)^{\alpha/2} u(t,x) = f(t,x,u(t,x)), \tag{5.5}$$

with some specified terminal condition $u(T,x) = g(x)$.

We develop a forward Euler-Maruyama-like scheme based on (2.27) on the time domain $[0,T]$, where $0 = t_0 < t_1 < \cdots < t_N < T$, by using the Brownian motion approximation on the small jumps:

$$X_{t_{n+1}} \approx X_{t_n} + \sqrt{c_{\alpha,d} k_{\alpha,d,r}} \Delta B_n + \sum_{s \in [t_n, t_{n+1})} \Delta X_s, \tag{5.6}$$

where the sum is over all the arrival times, s, in the interval $[t_n, t_{n+1})$ of the Poisson process with intensity $\lambda_{[r,\infty)} = \int_{|x| \in [r,\infty)} \nu(dx)$. ΔX_i are the jump sizes which are sampled from $\nu(dx) 1_{|x|>r} / \lambda_{|x|>r}$. The scheme for the corresponding BSDE reads:

$$\begin{aligned} u(t_{n+1}, X_{t_{n+1}}) \approx\ & u(t_n, X_{t_n}) - f(t_n, X_{t_n}, u(t_n, X_{t_n})) \Delta t_n \\ & + \sqrt{c_{\alpha,d} k_{\alpha,d,r}} \nabla u(t_n, X_{t_n})^\top \Delta B_n \\ & + \sum_{s \in [t_n, t_{n+1})} \Big[u(s, X_{s-} + \Delta X_s) - u(s, X_{s-}) \Big] \end{aligned} \tag{5.7}$$

Let $u(t_n, x)$ be approximated as a neural network $\mathcal{U}(x; \eta_n)$ as in DBDP. Further, automatic- or numerical differentiation applied to $\mathcal{U}(x; \eta_n)$, $\hat{\nabla}\mathcal{U}(x, \eta_n)$, is used to approximate $\nabla u(t_n, x)$. An issue with (5.7) is that we do not have a parametrization of $u(s,x)$ for $s \in (t_n, t_{n+1})$. We could therefore simply approximate $u(s,x)$ for $s \in (t_n, t_{n+1})$ as $\mathcal{U}(x, \eta_n)$ or $\mathcal{U}(x, \eta_{n+1})$. It is also possible to use a linear interpolation:

$$u(s,x) \approx \frac{s - t_n}{t_{n+1} - t_n} \mathcal{U}(x, \eta_{n+1}) + \frac{t_{n+1} - s}{t_{n+1} - t_n} \mathcal{U}(x,\ \eta_n).$$

The proposed algorithm is presented in **Algorithm 5.1**.

Algorithm 5.1: Solving fractional Laplace equations.

Input : X_0: Desired spatial point that the solution should be found at.
α: Index of fractional Laplace operator.
r: Cut-off value for the Brownian motion approximation.

Output: Estimate of $u(t_0, X_0)$.

$\lambda_{[r,\infty)} \leftarrow \int_{|x|\in[r,\infty)} \nu_\alpha(dx)$
$\mathcal{U}(.;\eta_N^\star) \leftarrow g(.)$
for $n = N-1, \dots, 0$ **do**
  /* Sample the forward process at time t_n. */
  $B_n \sim \mathcal{N}(0, t_n I)$
  $X_{t_n} \leftarrow \sqrt{c_{\alpha,d} k_{\alpha,d,r}} B_n$
  $P_n \sim \text{Poiss}(t_n \lambda_{[r,\infty)})$
  for $i = 0, \dots, P_n$ **do**
    $J \sim \nu 1_{|x|>r} / \lambda_{[r,\infty)}$
    $X_{t_n} \leftarrow X_{t_n} + J$
  $\Delta B_n \sim \mathcal{N}(0, \Delta t_n I)$
  $X_{t_{n+1}} \leftarrow X_{t_n} + \sqrt{c_{\alpha,d} k_{\alpha,d,r}} \Delta B_n$
  /* Sample the jumps and process in the interval $[t_n, t_{n+1})$. */
  $j \leftarrow 0$
  $s_0 \leftarrow t_n$
  repeat
    $\tau \sim \exp(\lambda_{[r,\infty)})$
    $\Delta X_j \sim \nu 1_{|x|>r} / \lambda_{[r,\infty)}$
    $s_{j+1} \leftarrow s_j + \tau$
    $\Delta B_j \sim \mathcal{N}(0, \tau I)$
    $X_{s_{j+1}} \leftarrow X_{s_j} + \Delta X_j + \sqrt{c_{\alpha,d} k_{\alpha,d,r}} \Delta B_j$
    $j \leftarrow j + 1$
  until $s_j < t_{n+1}$;
  $\Delta B' \sim \mathcal{N}(0, (t_n - s_{j-1}) I)$
  $X_{t_{n+1}} \leftarrow X_{s_{j-1}} + \sqrt{c_{\alpha,d} k_{\alpha,d,r}} \Delta B'$
  /* Compute the risk and optimize. */
  $\mathcal{R}_n(\eta_n) \leftarrow \mathrm{E}\Big[\big|\mathcal{U}(X_{t_{n+1}}; \eta_{n+1}^\star) - \mathcal{U}(X_{t_n}; \eta_n)$
  $\quad - \sum_{m=1}^{j-1} \big(\mathcal{U}(X_{s_m-} + \Delta X_m; \eta_n) - \mathcal{U}(X_{s_m-}; \eta_n)\big)$
  $\quad - \sqrt{c_{\alpha,d} k_{\alpha,d,r}} \hat{\nabla} \mathcal{U}(X_{t_n}; \eta_n) \Delta B_n$
  $\quad + f(t_n, X_{t_n}, u(t_n, X_{t_n}))\, \Delta t_n \big|^2 \Big]$
  $\eta_n^\star \in \arg\min_{\eta_n} \mathcal{R}_n(\eta_n)$
return $\mathcal{U}(X_0; \eta_0^\star)$

5.2 Numerical Results

The numerical scheme in **Algorithm 5.1** is implemented in Python using the TensorFlow 2.0 framework. All numerical results is produced on a MacBook Pro 2.2 GHz Intel Quad-Core i7 processor and 16GB of memory. The algorithm is carried out for a test case to solve the equation on the form (5.5). A discussion is included.

5.2.1 Test Case: An Equation with Sinusoidal Solution

We consider

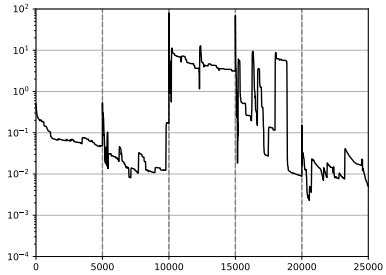
$$\frac{\partial u}{\partial t}(t,x) - (-\Delta)^{\alpha/2}u(t,x) - \sin(x)F'(t) - u(x,t)\int_{-\infty}^{\infty}(\cos(z)-1)\nu_\alpha(dz) = 0$$

with $g(x) = \sin(x)F(T)$ for some $F(t)$ in one dimension. The equation have the solution

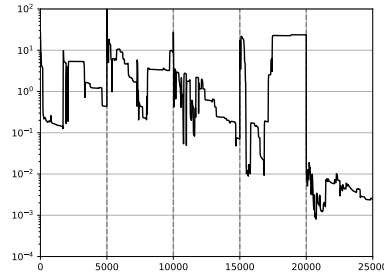
$$u(x,t) = \sin(x)F(t).$$

Let $F(t) = \exp(-t^2)$, $T = 0.1$ and $X_0 = -0.5$ such that the true solution is $u(0,-0.5) = \sin(-0.5) \approx -0.47942$. The equation is solved using 5 time intervals, trained for 5000 iterations on each step with a learning rate 0.001 without using pre-training. A validation set of size 128 and training set of size 64 is used. The neural network approximations of u have 2 hidden layers, both with a width of 10.

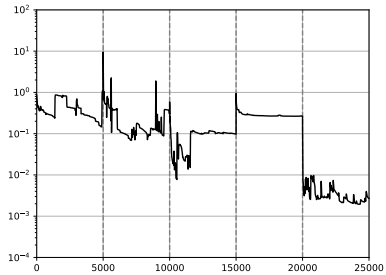
We consider $\alpha = 0.7$, $\alpha = 1.3$ and $\alpha = 1.8$ as test cases. For each of the three test cases, the algorithm is executed with three different cut-off values for the Brownian motion approximation. We let $r = 1/2$, $r = 1/20$ and $r = 1/200$. Each combination is trained 5 times and average values are reported. For $\alpha = 0.7$ the numerical results are provided in **Table 5.1**. **Figure 5.1** shows the validation loss averaged over the 5 simulations. The same results can be found in **Table 5.2** and **Figure 5.2** for $\alpha = 1.3$, and in **Table 5.3** and **Figure 5.3** for $\alpha = 1.8$.



(a) $r = 1/2$.



(b) $r = 1/20$.

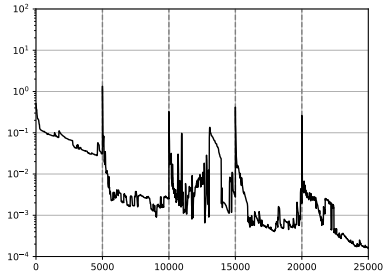


(c) $r = 1/200$.

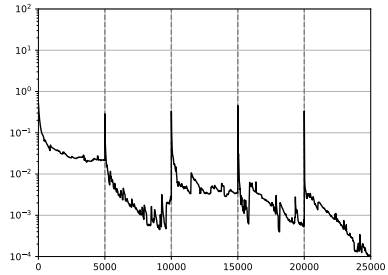
Figure 5.1: Validation loss for training of fractional Laplace algorithm with $\alpha = 0.7$.

	Estimate	**Standard error**	**Relative error**	**Time trained** (s)
$\mathbf{r = 1/2}$	-0.4765570	$34.4 \cdot 10^{-3}$	$60.6 \cdot 10^{-3}$	366.5
$\mathbf{r = 1/20}$	-0.2711255	$11.9 \cdot 10^{-3}$	$434 \cdot 10^{-3}$	315.8
$\mathbf{r = 1/200}$	-0.3437862	$19.9 \cdot 10^{-3}$	$32.9 \cdot 10^{-3}$	312.0

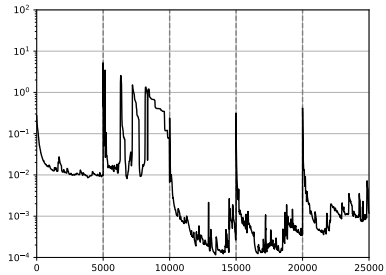
Table 5.1: Numerical results for fractional Laplace algorithm with $\alpha = 0.7$.



(a) $r = 1/2$.



(b) $r = 1/20$.

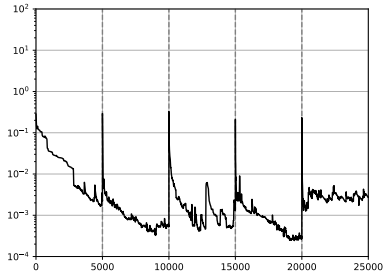


(c) $r = 1/200$.

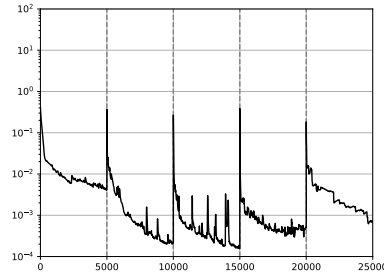
Figure 5.2: Validation loss for training of fractional Laplace algorithm with $\alpha = 1.3$.

	Estimate	**Standard error**	**Relative error**	**Time trained** (s)
$\mathbf{r = 1/2}$	-0.4854602	$3.45 \cdot 10^{-3}$	$12.6 \cdot 10^{-3}$	311.8
$\mathbf{r = 1/20}$	-0.5020119	$7.88 \cdot 10^{-3}$	$47.1 \cdot 10^{-3}$	320.6
$\mathbf{r = 1/200}$	-0.5059846	$1.14 \cdot 10^{-3}$	$55.4 \cdot 10^{-3}$	340.8

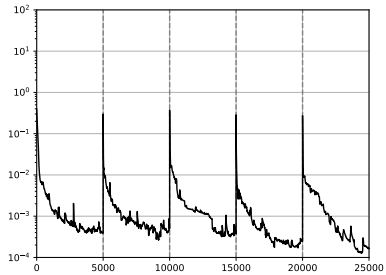
Table 5.2: Numerical results for fractional Laplace algorithm with $\alpha = 1.3$.



(a) $r = 1/2$.



(b) $r = 1/20$.



(c) $r = 1/200$.

Figure 5.3: Validation loss for training of fractional Laplace algorithm with $\alpha = 1.8$.

	Estimate	**Standard error**	**Relative error**	**Time trained** (s)
$\mathbf{r = 1/2}$	-0.4632650	$5.61 \cdot 10^{-3}$	$33.7 \cdot 10^{-3}$	290.8
$\mathbf{r = 1/20}$	-0.4795688	$2.06 \cdot 10^{-3}$	$3.95 \cdot 10^{-3}$	330.0
$\mathbf{r = 1/200}$	-0.4821258	$6.36 \cdot 10^{-3}$	$11.8 \cdot 10^{-3}$	451.5

Table 5.3: Numerical results for fractional Laplace algorithm with $\alpha = 1.8$.

5.2.2 Discussion

The first observation is that some of the results suffer from instability. We see it clearly on the validation loss plots, **Figure 5.1**, **Figure 5.2** and **Figure 5.3**, which are supposed to decay and exhibit a spike only after each 5000'th iteration, much like **Figure 5.3(c)**. The plots in **Figure 5.1** and **Figure 5.2** shows huge spikes on time steps where there ideally should not be spikes, confirming the instability. The results of **Algorithm 5.1** are increasingly good when α increases. The results for $\alpha = 0.7$ are of no worth, the results for $\alpha = 1.3$ are of very limited worth and the results for $\alpha = 1.8$ are of some worth. The decreasing stability with a decreasing α may suggest that the algorithm performs poorly when the forward process, see (5.6), are fat tailed.

The α-stable processes are notorious for their fat tails. The marginal distribution of the process even have an infinite mean for $\alpha \leq 1$ and an infinite variance for $\alpha < 2$. A different operator should therefore be considered for further testing. For instance, the operator

$$\mathcal{L}u(t,x) = \int_{\mathbb{R}\setminus\{0\}} \Big[u(t,x+z) - u(t,x) - z^\top \nabla u(t,x) 1_{|z|<1}\Big] \nu_{C,G,M,Y}(dx)$$

where

$$\nu_{C,G,M,Y}(dx) = \Big(\frac{C\exp(Gx)}{|x|^{1+Y}} 1_{x<0} + \frac{C\exp(-Mx)}{|x|^{1+Y}} 1_{x>0}\Big) dx. \tag{5.8}$$

This is the so-called CGMY process, as presented in Carr et al. (2003). We see that (5.8) decays a lot faster towards zero compared to (5.1), and it is therefore believed to be less prone to the instability.

We also observe that the effect of the cut-off value, r, does influence the results. We consider the case with $\alpha = 1.8$ and compare the relative error, given in **Table 5.3**, for the different cut-off values. The lowest relative error is attained for $r = 1/20$ and the highest relative error is attained for $r = 1/2$. One would expect that the error would strictly increase as a function of r. However, **Figure 2.6** and **Figure 2.7** shows limited visual difference between $r = 0.1$ and $r = 0.01$. So, $r = 1/20$ might be a sufficiently low cut-off for the fractional Laplace algorithm, allowing the total error to be dominated by other sources.

Chapter 6

Concluding Remarks

The thesis aims to verify the Deep BSDE and the two variants of DBDP, as well as an extension to solve equations involving the fractional Laplace. Both Deep BSDE and DBDP successfully avoids the curse of dimensionality by reformulating PDEs to learning problems.

A thorough presentation of the background theory is performed. The theory is important in order to understand the deep learning algorithms. Stochastic calculus is discussed, a deep dive is made into the Lévy processes and neural networks are introduced. We focus on two main methods for solving semi-linear parabolic PDEs, namely Deep BSDE and the two variants of DBDP. Both variants of DBDP is implemented in Python using the TensorFlow 2.0 framework and numerical results for several examples are provided. At last, the extension to solve equations involving the fractional Laplace is presented. This algorithm is also implemented using the TensorFlow 2.0 framework.

Both Deep BSDE and the two variants of DBDP successfully solves 100-dimensional semi-linear parabolic PDEs in most cases. Both variants of DBDP converge to the wrong value for only one of the test examples. Being able to solve such high dimensional PDEs is in practice not possible for previous methods. Unfortunately, the results of the fractional Laplace equation are somewhat disappointing. The algorithm gives some useful results in one dimension for $\alpha = 1.8$, but for other cases the results are of no use. The reason might be that the tails of the α-stable processes are simply too fat, and hence stable convergence is not possible. To confirm or discard this hypothesis, further testing should be done. For instance, the CGMY processes which are a class of Lévy processes where the Lévy measure decays exponentially and thus have thinner tails.

During the end of the work on this thesis Germain et al. (2020) was published. This article presents a new multistep method which is somewhat similar to DBDP. The article further shows that the multistep algorithm performs better, or similar to, Deep BSDE and DBDP. In particular it would be interesting to test such an algorithm on the fractional Laplace equation to see whether or not it would be an improvement.

Bibliography

Asmussen, S., Glynn, P.W., 2007. Stochastic Simulation: Algorithms and Analysis. Springer. doi:https://doi.org/10.1007/978-0-387-69033-9.

Asmussen, S., Rosiński, J., 2001. Approximations of small jumps of lévy processes with a view towards simulation. Journal of Applied Probability 38, 482–493. URL: http://www.jstor.org/stable/3215901.

Barles, G., Buckdahn, R., Pardoux, E., 1997. Backward stochastic differential equations and integral-partial differential equations. Stochastics and Stochastic Reports 60, 57–83. URL: https://doi.org/10.1080/17442509708834099, doi:10.1080/17442509708834099, arXiv:https://doi.org/10.1080/17442509708834099.

Carr, P.P., Geman, H., Madan, D.B., Yor, M., 2003. Stochastic Volatility for Levy Processes. Mathematical Finance , 345–382.URL: https://engineering.nyu.edu/sites/default/files/2019-03/Carr-stochastic-volatility-levy-processes.pdf, doi:http://dx.doi.org/10.2139/ssrn.314979.

Cohen, S., Rosiński, J., 2007. Gaussian approximation of multivariate lévy processes with applications to simulation of tempered stable processes. Bernoulli 13, 195–210. URL: https://doi.org/10.3150/07-BEJ6011, doi:10.3150/07-BEJ6011.

Cont, R., Tankov, P., 2004. Financial Modelling With Jump Processes. Chapman Hall/CRC: Financial MAthematics Series.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems 2, 303–314. URL: https://doi.org/10.1007/BF02551274, doi:10.1007/BF02551274.

Duchi, J., Hazan, E., Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. 12, 2121–2159.

E, W., Han, J., Jentzen, A., 2017a. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. arXiv e-prints , arXiv:1706.04702arXiv:1706.04702.

E, W., Han, J., Jentzen, A., 2017b. Deep Learning-Based Numerical Methods for High-Dimensional Parabolic Partial Differential Equations and Backward Stochastic Differential Equations. Communications in Mathematics and Statistics 5, 349–380. doi:https://doi.org/10.1007/s40304-017-0117-6.

Germain, M., Pham, H., Warin, X., 2020. Deep backward multistep schemes for nonlinear PDEs and approximation error analysis. arXiv e-prints , arXiv:2006.01496arXiv:2006.01496.

Glasserman, P., 2003. Monte Carlo Methods in Financial Engineering. Springer.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. The MIT Press.

Han, J., 2019. Deep BSDE Solver in TensorFlow (2.0). URL: https://github.com/frankhan91/DeepBSDE.

Han, J., Jentzen, A., E, W., 2017. Solving high-dimensional partial differential equations using deep learning. arXiv e-prints , arXiv:1707.02568arXiv:1707.02568.

Hochreiter, S., 1991. The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems .

Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. Neural Networks 4, 251 – 257. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T, doi:https://doi.org/10.1016/0893-6080(91)90009-T.

Huré, C., Pham, H., Warin, X., 2019. Deep backward schemes for high-dimensional nonlinear PDEs. arXiv e-prints , arXiv:1902.01599arXiv:1902.01599.

Kingma, D.P., Ba, J., 2014. Adam: A Method for Stochastic Optimization. arXiv e-prints , arXiv:1412.6980arXiv:1412.6980.

Kwaśnicki, M., 2015. Ten equivalent definitions of the fractional Laplace operator. arXiv e-prints , arXiv:1507.07356arXiv:1507.07356.

Lawler, G.F., 2014. Stochastic Calculus: An Introduction with Applications.

Lu, Z., Pu, H., Wang, F., Hu, Z., Wang, L., 2017. The expressive power of neural networks: A view from the width, in: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30. Curran Associates, Inc., pp. 6231–6239. URL: http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf.

Nocedal, J., Wright, S.J., 2006. Numerical Optimization. 2 ed., Springer.

Øksendal, B., 2013. Stochastic Differential Equations. 6 ed., Springer.

Pardoux, E., 1995. Backward stochastic differential equations and applications, in: Chatterji, S.D. (Ed.), Proceedings of the International Congress of Mathematicians, Birkhäuser Basel, Basel. pp. 1502–1510.

Pardoux, E., Răşcanu, A., 2014. Stochastic Differential Equations, Backward SDEs, Partial Differential Equations. Springer.

Pardoux, E., Tang, S., 1999. Forward-backward stochastic differential equations and quasilinear parabolic pdes. Probability Theory and Related Fields 114, 123–150. URL: https://doi.org/10.1007/s004409970001.

Pérez-Cruz, F., 2018. URL: http://www.da.inf.ethz.ch/teaching/2018/DeepLearning/.

NTNU

Norwegian University of
Science and Technology