Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Matematica

Deep Learning and Nonlinear PDEs in High-Dimensional Spaces

Tesi di Laurea in Equazioni Differenziali Stocastiche

Relatore:
Chiar.mo Prof.
Andrea Pascucci

Presentata da:
Eugenio Rossini

Correlatore:
Chiar.mo Prof.
Renato Campanini

IV Sessione
Anno Accademico 2017/2018

Contents

List of Figures

Introduction

In this thesis we present a method to solve high-dimensional nonlinear PDEs by using the Deep Learning theory. The algorithm was introduced by Weinan, Han and Jentzen in 2017 [WHJ17] and it can produce a solution to semilinear PDEs even in dimension one hundred. This method allows to overcome the major drawbacks of working in high-dimensional spaces. As we know the most evident difficulty lies in the "curse of dimensionality", namely, as the dimensionality grows the algorithm complexity exponentially grows too. There is a limited number of methods that can solve high-dimensional PDEs. For linear parabolic PDEs, one can use the Feynman-Kac formula and Monte Carlo methods to develop algorithms to evaluate solutions at any given space-time location. In [HJK17] it was developed a quite efficient algorithm to approximate the solution to nonlinear parabolic PDEs based on the nonlinear Feynman-Kac formula and the multilevel Picard technique. The complexity of this algorithm is shown to be $\mathcal{O}(d\epsilon^{-4})$ for semilinear heat equations, where d is the dimensionality of the problem and ϵ is the required accuracy.

In recent years, Deep Learning techniques have emerged in a wide variety of different topics. The main aim of Machine Learning is to solve problems with a large number of data or features, for example; computer vision, natural language processing, time series analysis etc. This change of programming paradigm has assisted the Machine Learning to reach its potential in both statistics and computer science. Many papers have been published with the aim of improving theoretical and empirical knowledge. This success stimulates speculations that Deep Learning might hold the key to solve the curse of dimensionality problem.

This thesis is structured as follows.

- **Chapter 1.** We introduce some results about the theory of Backward Stochastic Differential Equations and we prove the nonlinear version of the Feynman-Kac formula. This formula will be used to transform the nonlinear PDE solving problem into a BSDE solving problem.
- **Chapter 2.** We introduce the most important concepts of the Machine Learning theory. We define the Deep Learning problem and Multilayer Neural Networks. We show an example of Deep Neural Networks applied to an engineering problem.
- **Chapter 3.** We present the Deep Learning-based BSDE solver in order to approximate the solution to a semilinear parabolic PDE. After a brief introduction to the option pricing problem, we will apply the Neural Network algorithm to the 100-dimensional nonlinear Black-Scholes equation.
- **Appendixes.** We report some theorems about stochastic analysis and we introduce a modern approach to Deep Learning. This approach consists of giving theoretical results to explain why Deep Neural Networks work so well.

Chapter 1

Backward Stochastic Differential Equations and PDEs

The main aim of this chapter is to present the theory and results about the link between (nonlinear) Partial Differential Equations (PDEs) and Stochastic Differential Equations (SDEs). In particular, we consider the Feynman-Kac formula, which in its classical statement provides the solution to a PDE through probabilistic properties of stochastic processes. We will see a possible extension of this result in the case that the PDE is not linear.
In the first section we explore the Backward Stochastic Differential Equations (BSDEs) theory by giving proof of an existence and uniqueness theorem of BSDEs solution (([MMY99], [PR14], [Z17]). After that we present the nonlinear Feynman-Kac formula ([P98], [P15], [PP92]). Finally, we give a brief introduction of the main numerical methods to solve BSDEs.

1.1 Basic properties of BSDEs

We assume that $T \in (0, \infty)$. Let $W = (W_t)_{0 \leq t \leq T}$ be a $d-$dimensional Brownian motion on a filtered probability space $(\Omega, \mathcal{F}, \mathbb{F}, P)$, where $\mathbb{F} =$

$(\mathcal{F}_t)_{0\leq t\leq T}$ is the natural filtration associated to W. Furthermore, we define

$$\mathbb{F}^t := \{\mathcal{F}_s^t\}_{t\leq s\leq T}, \quad \mathcal{F}_s^t = \sigma(W_r - W_t, t \leq r \leq s)$$

with $0 \leq t \leq T$.

Definition 1.1. Let $\mathbb{S}^2(0,T)^k$ be the set of $\mathbb{R}^k$-valued stochastic processes Y, progressively measurable, such that

$$E\left[\sup_{0\leq t\leq T}|Y_t|^2\right] < \infty.$$

Definition 1.2. Let $\mathbb{H}^2(0,T)^d$ be the set of $\mathbb{R}^d$-valued stochastic processes Z, progressively measurable, such that

$$E\left[\int_0^T |Z_t|^2 dt\right] < \infty.$$

Stated differently, $\mathbb{H}^2(0,T)^d$ is the set of all progressively measurable processes, subset of $L^2([0,T]\times\Omega, dt\otimes dP; \mathbb{R}^d)$ (*i.e.*, for t fixed, the $\mathbb{R}^d$-valued square integrable process Z restrict to $[0,t]\times\Omega$ is $\mathcal{B}([0,t])\otimes\mathcal{F}_t$ -measurable). We define a pair of variables (ξ, f) and suppose that the following properties are valid:

(A) $\xi \in L^2(\Omega, \mathcal{F}_T, P; \mathbb{R}^k)$;

(B) $f : \Omega\times[0,T]\times\mathbb{R}^k\times\mathbb{R}^{k\times d} \to \mathbb{R}^k$ s.t.:

- $f(\cdot, t, y, z)$, abbreviate $f(t,y,z)$, is progressively measurable $\forall y, z$;
- $f(t,0,0) \in \mathbb{H}^2(0,T)^k$;
- f is uniformly Lipschitz in (y,z), *i.e.* $\exists$ C_f constant such that

$$|f(t,y_1,z_1) - f(t,y_2,z_2)| \leq C_f(|y_1-y_2| + \|z_1-z_2\|)$$

$$\forall y_1, y_2 \in \mathbb{R}^k, \ \forall z_1, z_2 \in \mathbb{R}^{k\times d}, \ dt\otimes dP \ a.s., \text{ with } \|z\| = [Tr(zz^T)]^{\frac{1}{2}};$$

A **solution** to the BSDE characterized by (ξ, f), is a pair of $\mathbb{R}^k\times\mathbb{R}^{k\times d}$-valued progressively measurable stochastic processes $\{(Y_t, Z_t);\ 0 \leq t \leq T\}$

such that $(Y, Z) \in \mathbb{S}^2(0,T)^k \times \mathbb{H}^2(0,T)^{k\times d}$ and the following equation holds:

$$-dY_t = f(t, Y_t, Z_t)dt - Z_t dW_t, \quad Y_T = \xi \tag{1.1}$$

Equivalently, the same BSDE can be stated as an integral stochastic equation

$$Y_t = \xi + \int_t^T f(s, Y_s, Z_s)ds - \int_t^T Z_s dW_s, \quad 0 \le t \le T.$$

We call (ξ, f), respectively, the **Terminal Condition** and the **Driver** (or Generator) of BSDE.
The next result is the existence and uniqueness theorem for BSDEs.

Theorem 1.1.1. *Let (ξ, f) be the Terminal Condition and the Driver of a BSDE, which satisfies the conditions (A) and (B). Then, exists the solution (Y, Z) to BSDE (1.1) and it is unique.*

Proof. The proof is based on the fixed-point method.
We consider a function Φ on $\mathbb{S}^2(0,T)^k \times \mathbb{H}^2(0,T)^{k\times d}$, mapping $(U, V) \in \mathbb{S}^2(0,T)^k \times \mathbb{H}^2(0,T)^{k\times d}$ to $(Y, Z) = \Phi(U, V)$ defined by

$$Y_t = \xi + \int_t^T f(s, U_s, V_s)ds - \int_t^T Z_s dW_s. \tag{1.2}$$

More precisely, we construct the stochastic processes (Y, Z) as follows: first we consider the martingale

$$M_t = E[\xi + \int_0^T f(s, U_s, V_s)ds | \mathcal{F}_t]$$

. By the conditions on (ξ, f), this is a d-dimensional square-integrable martingale. We can use the martingale representation theorem (see Appendix A, Theorem A.0.1), which allows us to represent a random variable by using the Itô's integral. This theorem gives us a proof of existence and uniqueness of the stochastic process $Z \in \mathbb{H}^2(0,T)^{k\times d}$ such that

$$M_t = M_0 + \int_0^t Z_s dW_s. \tag{1.3}$$

Now we define the process Y as follows

$$Y_t = E\Big[\xi + \int_t^T f(s, U_s, V_s)ds|\mathcal{F}_t\Big] = M_t - \int_0^t f(s, U_s, V_s)ds, \quad 0 \leq t \leq T.$$

We can replace the representation of M (1.3) in the previuos equation, and by noting that $Y_T = \xi$, we obtain the Equation (1.2). By Doob's inequality (see Appendix A, Theorem A.0.2) we observe that

$$E\Big[\sup_{0\leq t\leq T} |\int_t^T Z_s dW_s|^2\Big] \leq 4E\Big[\int_0^T |Z_s|^2 ds\Big] < \infty.$$

Hence, by the conditions on (ξ, f), we obtain that $Y \in \mathbb{S}^2(0,T)^k$. From this we deduce that Φ is a well-defined function from $\mathbb{S}^2(0,T)^k \times \mathbb{H}^2(0,T)^{k\times d}$ into itself. The next step is to show that the pair (Y, Z) is a solution to the BSDE (1.1) if and only if it is a fixed point of Φ.
Let (U, V), $(U', V') \in \mathbb{S}^2(0,T)^k \times \mathbb{H}^2(0,T)^{k\times d}$ and

$$(Y, Z) = \Phi(U, V), \;\; (Y', Z') = \Phi(U', V').$$

We set

$$(\bar{U}, \bar{V}) = (U' - U, V' - V), \;\; (\bar{Y}, \bar{Z}) = (Y - Y', Z - Z')$$

and

$$\bar{f}_t = f(t, U_t, V_t) - f(t, U'_t, V'_t).$$

We define $\beta > 0$, and apply the Itô's formula to the process $e^{\beta s}|\bar{Y}_s|^2$ between $s = 0$ and $s = T$:

$$|\bar{Y}_0|^2 = -\int_0^T e^{\beta s}(\beta|\bar{Y}_s|^2 - 2\bar{Y}_s\bar{f}_s)ds - \int_0^T e^{\beta s}|\bar{Z}_s|^2 ds - 2\int_0^T e^{\beta s}\bar{Y}_s^T\bar{Z}_s dW_s. \tag{1.4}$$

Notice that

$$E\Big[(\int_0^T e^{2\beta t}|Y_t|^2|Z_t|^2 dt)^{\frac{1}{2}}\Big] \leq \frac{e^{\beta T}}{2} E\Big[\sup_{0\leq t\leq T}|Y_t|^2 + \int_0^T |Z_t|^2 dt\Big] < \infty,$$

results from Burkholder-Davis-Gundy inequality (see Appendix A, Theorem A.0.3) and shows that the process

$$\int_0^t e^{\beta s}\bar{Y}_s^T \bar{Z}_s dW_s$$

is a uniformly integrable local martingale. We observe that the left-hand side (l.h.s.) of the previous inequality is the Quadratic Variation Process of the local martingale. The second inequality derives from $Y \in \mathbb{S}^2(0,T)^k$ and $Z \in \mathbb{H}^2(0,T)^{k\times d}$. By taking the expectation, the Equation 1.4 becomes:

$$E[|\bar{Y}_0|^2] + E\bigg[\int_0^T e^{\beta s}(\beta|\bar{Y}_s|^2 + |\bar{Z}_s|^2)ds\bigg] = 2E\bigg[\int_0^T e^{\beta s}\bar{Y}_s\bar{f}_s ds\bigg]$$

$$\leq 2C_f E\bigg[\int_0^T e^{\beta s}|\bar{Y}_s|(|\bar{U}_s| + |\bar{V}_s|)ds\bigg]$$

$$\leq 4C_f^2 E\bigg[\int_0^T e^{\beta s}|\bar{Y}_s|^2 ds\bigg] + \frac{1}{2}E\bigg[\int_0^T e^{\beta s}(|\bar{U}_s|^2 + |\bar{V}_s|^2)ds\bigg].$$

Here the first inequality is verified by the Lipschitz uniform condition on f and $\bar{f}$. We set $\beta = 1 + 4C_f^2$. Hence, by substituting in previous inequalities, we get

$$E\bigg[\int_0^T e^{\beta s}(|\bar{Y}_s|^2 + |\bar{Z}_s|^2)ds\bigg] \leq \frac{1}{2}E\bigg[\int_0^T e^{\beta s}(|\bar{U}_s|^2 + |\bar{V}_s|^2)ds\bigg].$$

We define the norm on the Banach space $\mathbb{S}^2(0,T) \times \mathbb{H}^2(0,T)^d$:

$$\|(Y,Z)\|_\beta = \bigg(E\bigg[\int_0^T e^{\beta s}(|Y_s|^2 + |Z_s|^2)ds\bigg]\bigg)^{\frac{1}{2}}.$$

From the above inequality:

$$\|\Phi(U,V) - \Phi(U',V')\|_\beta = \|(Y,Z) - (Y',Z')\|_\beta \leq \frac{1}{4}\|(U,V) - (U',V')\|_\beta.$$

So, Φ is a strict contraction mapping. Hence, we conclude that Φ, by using the contraction mapping theorem, admits a unique fixed point. This point

is the solution to the BSDE. □

1.1.1 Linear BSDEs

In this section we will focus on linear BSDEs, historically the first Backward Stochastic Equations that have been studied [B73]. The main result that we report provides us an important tool to obtain the solution to this type of BSDEs. Such proposition will be useful in the next section to prove more general conclusions.
Now we consider the case where the drive of BSDE f is a linear function in y and z. We can write the linear BSDE as follows

$$-dY_t = (A_tY_t + Z_tB_t + C_t)dt - Z_tdW_t, \quad Y_T = \xi \tag{1.5}$$

or equivalently

$$Y_t = \xi + \int_t^T [A_sY_s + Z_sB_s + C_s]ds - \int_t^T Z_sdW_s$$

where A and B are, respectively, two $\mathbb{R}^k$ and $\mathbb{R}^{k\times d}$-valued bounded and progressively measurable processes, and $C \in \mathbb{H}^2(0,T)^k$. Under these conditions we can solve explicitly the above BSDE. Note that such BSDE is well-defined by general theory in the previous section. Here, we only want to determine a representation formula for the solution.

Proposition 1.1.2. *Under appropriate regularity hypothesis on coefficients (see above), the unique solution* (Y, Z) *to the linear BSDE* (1.5) *is given by*

$$\Gamma_tY_t = E\left[\Gamma_T\xi + \int_t^T \Gamma_sC_sds|\mathcal{F}_t\right], \tag{1.6}$$

where Γ *is the solution process to the linear SDE*

$$d\Gamma_t = \Gamma_t(A_tdt + B_tdW_t), \quad \Gamma_0 = 1$$

or

$$\Gamma_t = \exp\left(\int_0^t B_s dW_s + \int_0^t \left[A_s - \frac{1}{2}|B_s|^2\right] ds\right)$$

Proof. The result follows applying Itô's formula to the process $\Gamma_t Y_t$. We then obtain

$$d(\Gamma_t Y_t) = -\Gamma_t C_t dt + \Gamma_t (Y_t B_t + Z_t) dW_t$$

hence

$$\Gamma_t Y_t + \int_0^t \Gamma_s C_s ds = Y_0 + \int_0^t \Gamma_s (Y_s B_s + Z_s) dW_s. \tag{1.7}$$

The fact that A and B are bounded processes combined with the definition of Γ, guarantees us that $E[\sup_t |\Gamma_t|^2] < \infty$. Denote by b_∞ the upper bound of B, then the following inequalities are true

$$E\left[\left(\int_0^T \Gamma_s^2 |Y_s B_s + Z_s|^2 ds\right)^{\frac{1}{2}}\right] \leq \frac{1}{2} E\left[\sup_t |\Gamma_t|^2 + 2\int_0^T |Z_t|^2 dt + 2b_\infty^2 \int_0^T |Y_t|^2 dt\right] < \infty$$

By the Burkholder-Davis-Gundy inequality we prove that the local martingale in (1.7) is uniformly integrable. Then, by taking the expectation,

$$\Gamma_t Y_t + \int_0^t \Gamma_s C_s ds = E\left[\Gamma_T Y_T + \int_0^T \Gamma_s C_s ds | \mathcal{F}_t\right] =$$

$$= E\left[\Gamma_T \xi + \int_0^T \Gamma_s C_s ds | \mathcal{F}_t\right]$$

and we obtain (1.6). By the martingale representation theorem on (1.7), we achieve the process Z (by considering the expectation of $\Gamma_t Y_t$ as a martingale). □

1.1.2 Comparison Principles of Solutions

In this section we present a comparison theorem, which allows us to compare solutions related to different BSDEs.

Theorem 1.1.3. *Let (ξ_1, f_1) and (ξ_2, f_2) be two pairs of Terminal Conditions and Drivers that satisfy the assumptions (A) and (B) (see Section 1.1). Let*

$(Y^1, Z^1), (Y^2, Z^2)$ be two solutions to BSDEs of the form (1.1) related to (ξ_1, f_1) and (ξ_2, f_2). Moreover, if the following conditions hold:

- $\xi^1 \leq \xi^2 \quad P-a.s.$
- $f^1(t, Y_t^1, Z_t^1) \leq f^2(t, Y_t^1, Z_t^1) \quad dt \otimes dP - a.s.$
- $f^2(t, Y_t^1, Z_t^1) \in \mathbb{H}^2(0,T)^{k\times d}$,

then $Y_t^1 \leq Y_t^2$ for all $0 \leq t \leq T$, $P-a.s.$
Furthermore, if $Y_0^2 \leq Y_0^1$, then $Y_t^1 = Y_t^2, \quad 0 \leq t \leq T$.
In particular, if $P(\xi^1 < \xi^2) > 0$ or $f^1(t,\cdot,\cdot) < f^2(t,\cdot,\cdot)$ on a set endowed with a strictly positive measure $dt \otimes dP$, then $Y_0^1 < Y_0^2$.

Proof. We set $\bar{Y} = Y^2 - Y^1, \bar{Z} = Z^2 - Z^1$. Moreover, let

$$\Delta_t^y = \frac{f^2(t, Y_t^2, Z_t^2) - f^2(t, Y_t^1, Z_t^2)}{Y_t^2 - Y_t^1} \mathbf{1}_{Y_t^2 - Y_t^1 \neq 0}$$

$$\Delta_t^z = \frac{f^2(t, Y_t^1, Z_t^2) - f^2(t, Y_t^1, Z_t^1)}{Y_t^2 - Y_t^1} \mathbf{1}_{Z_t^2 - Z_t^1 \neq 0}$$

$$\bar{f}_t = f^2(t, Y_t^1, Z_t^1) - f^1(t, Y_t^1, Z_t^1).$$

Then, $(\bar{Y}, \bar{Z})$ satisfy the linear BSDE

$$-d\bar{Y}_t = (\Delta_t^y \bar{Y}_t + \Delta_t^z \bar{Z}_t + \bar{f}_t)dt - \bar{Z}_t dW_t, \quad \bar{Y}_T = \xi^2 - \xi^1. \tag{1.8}$$

By the uniform Lipschitz continuity of f^2 in y and z, we obtain that Δ^y and Δ^z are bounded. Furthermore, $\bar{f}_t \in \mathbb{H}^2(0,T)^{k\times d}$. By using Proposition 1.1.2 it follows that $\bar{Y}$ is given by

$$\Gamma_t \bar{Y}_t = E\left[\Gamma_T(\xi^2 - \xi^1) + \int_t^T \Gamma_s \bar{f}_s ds | \mathcal{F}_t\right],$$

where Γ is a strictly positive process. From the hypothesis we can see that $\xi^2 - \xi^1 \geq 0$ and $\bar{f} \geq 0$. Therefore, we can conclude that $\bar{Y} \geq 0$. □

Remark 1. An important observation is that in the proof of Theorem 1.1.3, we have to impose the regularity condition only on the generator f^2. It is not necessary to impose the uniform Lipschitz condition on f^1.

Corollary 1.1.4. *If the pair* (ξ, f) *satisfies* $\xi \geq 0 \quad P-a.s.$ *and* $f(t,0,0) \geq 0 \quad dt \otimes dP - a.s.$, *then we have* $Y_t \geq 0, \quad 0 \leq t \leq T \quad P-a.s.$ *Moreover, if* $P(\xi > 0) > 0$ *or* $f(t,0,0) > 0 \quad dt \otimes dP - a.s,$ *then* $Y_0 > 0$.

Proof. The proof immediately follows by the comparison Theorem 1.1.3, when $(\xi^1, f^1) = (0,0)$. In this case, the solution to BSDE is definitely $(Y^1, Z^1) = (0,0)$. □

1.2 Feynman-Kac Formula and Its Extension to the Nonlinear Case

We introduce a central tool in the stochastic analysis of PDEs, the Feynman-Kac formula. After the classical formulation by Feynman and Kac, we give a possible extension to the nonlinear case. We consider the BSDE of the form:

$$\begin{cases} dX_s = b(s, X_s)ds + \sigma(s, X_s)dW_s, \quad X_0 = x \in \mathbb{R}^n \\ -dY_s = f(s, X_s, Y_s, Z_s)ds - Z_s dW_s, \quad Y_T = g(X_T) \end{cases} \tag{1.9}$$

where $(t,x) \in [0,T) \times \mathbb{R}^n$ and $x \in \mathbb{R}^n$.
In literature these kind of BSDEs are also called Forward BSDEs (FBSDEs), because of the presence of a forward stochastic differential equation in the system. We also prove, under certain regularity conditions on coefficients, that the solution to FBSDE (1.9) has the Markov property. This characteristic is crucial to determine stochastic processes, that are the solution to FBSDEs (1.9). We want to find the classical solution to a semilinear PDE of the form:

$$-\frac{\partial v}{\partial t}(t,x) - \mathcal{L}v(t,x) - f(t,x,v(t,x),\langle \sigma(t,x), D_x v(t,x)\rangle) = 0 \tag{1.10}$$

$$v(T,x) = g(x) \tag{1.11}$$

We also demonstrate that in certain cases the *vice versa* is true. It is possible to find the solution to a semilinear PDE by knowing the solution to the BSDE. However, in this case the solution to (1.10) - (1.11) does not satisfy

the regularity property as in classical solutions. Hence, we introduce the notion of **Viscosity Solution**.

First and foremost we recall the **linear** version of Feynman-Kac formula. This result allows us to express the solution to a parabolic PDE (with final condition, *i.e.* backward) of the form:

$$-\frac{\partial v}{\partial t} - \mathcal{L}v - f(t,x) = 0, \quad (t,x) \in [0,T) \times \mathbb{R}^n,$$

$$v(T,x) = g(x), \quad x \in \mathbb{R}^n,$$

using stochastic processes and probability theory

$$v(t,x) = E\bigg[\int_t^T f(s, X_s^{t,x})ds + g(X_T^{t,x})\bigg], \tag{1.12}$$

where $\{X_s^{t,x}, t \leq s \leq T\}$ is the solution to

$$dX_s = b(s, X_s)ds + \sigma(s, X_s)dW_s, \quad t \leq s \leq T, \quad X_t = x$$

where W is a d-dimensional Brownian motion and $\mathcal{L}$ is a second order differential operator:

$$\mathcal{L}v(t,x) = \big\langle b(t,x), D_x v(t,x)\big\rangle + \frac{1}{2}Tr(\sigma\sigma^T(t,x)D_{xx}^2 v(t,x)).$$

We would generalize this result when semilinear (nonlinear) PDEs are expressed in the form (1.10) - (1.11).

For the rest of the chapter we suppose that processes X, Y and Z, that form the solution of (1.9) are, respectively, $\mathbb{R}^n$, $\mathbb{R}^k$ and $\mathbb{R}^{k\times d}$-valued processes. Moreover, let W be a $\mathbb{R}^d$-valued Brownian motion. We assume some necessary conditions on terms of BSDE (1.9) and PDE (1.10)-(1.11). These conditions are summed up in the following remark.

Remark 2. **Assumptions**

(i) b, σ, f, g are, respectively, $\mathbb{R}^n$, $\mathbb{R}^{n\times d}$, $\mathbb{R}^k, \mathbb{R}^k$-valued deterministic function. Moreover, $b(\cdot,0), \sigma(\cdot,0), f(\cdot,0,0,0)$ and $g(0)$ are bounded.

(ii) b, σ, f, g are uniform Lipschitz continuous functions in (x, y, z) with Lipschitz constant L.

(iii) f is a continuous function on $[0, T] \times \mathbb{R}^n \times \mathbb{R}^k \times \mathbb{R}^{k \times d}$

(iv) f satisfies the linear growth condition in (x, y, z), *i.e.*

$$|f(t, x, y, z)| \leq K(1 + |x|^p + |y| + |z|)$$

with $K > 0$ and $(x, y, z) \in \mathbb{R}^n \times \mathbb{R}^k \times \mathbb{R}^{k \times d}$

(v) g is a continuous function that satisfies the linear growth condition

$$|g(x)| \leq K(1 + |x|^p), \quad K > 0$$

Remark 3. Equivalently, it is possible to define the FBSDE problem (1.9) by using integral stochastic equations

$$\begin{cases} X_t = x + \int_0^t b(s, X_s)ds + \int_0^t \sigma(s, X_s)dW_s \\ Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s dW_s \end{cases} \tag{1.13}$$

It is possible to prove that the Terminal Condition and the Generator of BSDE in (1.9) satisfy (A) and (B) (1.1).

1.2.1 Markov Property of the Solution

For all $(t, x) \in [0, T] \times \mathbb{R}^n$ and for any $\eta \in \mathbb{L}^2(\mathcal{F}_t)$, the process

$$\{X_s^{t,x}, t \leq s \leq T\}$$

is the solution to the SDE in (1.9), which starts from x at time t. Let $\{(Y_s^{t,x}, Z_s^{t,x}), t \leq s \leq T\}$ and $\{(\mathcal{Y}_s^{t,\eta}, \mathcal{Z}_s^{t,\eta}), t \leq s \leq T\}$ be the solution of the BSDE in (1.9) with $X_s = X_s^{t,x}$ and $X_s = \mathcal{X}_s^{t,\eta}$ where $t \leq s \leq T$. By the uniqueness of the solution to the BSDE, we have $(Y_s, Z_s) = (\mathcal{Y}_s^{t,X_t}, \mathcal{Z}_s^{t,X_t})$.

Theorem 1.2.1. *We assume $0 \leq t \leq T$. If the assumptions (i)-(v) in Remark 2 are valid, then*

- *exists a version of $(Y^{t,x}, Z^{t,x})$ for each x such that the mapping $(x, s, \omega) \mapsto (Y_s^{t,x}(\omega), Z_s^{t,x}(\omega))$ is $\mathbb{F}^t$-progressively measurable. Furthermore, $(Y^{t,x}, Z^{t,x})$ is indipendent of $\mathcal{F}_t$.*
- *For any $\eta \in \mathbb{L}^2(\mathcal{F}_t)$, we have*

$$(\mathcal{Y}_s^{t,\eta}(\omega), \mathcal{Z}_s^{t,\eta}(\omega)) = (Y_s^{t,\eta(\omega)}(\omega), Z_s^{t,\eta(\omega)}(\omega)), \quad ds \otimes dP - a.s.(s, \omega).$$

- *Consequently, (X, Y, Z) is Markov.*

For the detailed proof we refer to [Z17].
Now we define

$$v(t, x) := Y_t^{t,x} \tag{1.14}$$

Then $v(t, x)$ is both $\mathcal{F}_t$-measurable and independent of $\mathcal{F}_t$, and thus is deterministic. Since $Y_t = \mathcal{Y}_t^{t,X_t} = Y_t^{t,X_t}$, we have

$$Y_t = v(t, X_t), \quad 0 \leq t \leq T.$$

1.2.2 Nonlinear Feynman-Kac Formula

We introduce a fundamental result that allows us to extend the Feynman-Kac formula, that we saw at the beginning of this section, to a nonlinear framework. This shows how the classical solution of a semilinear PDE provides a process-solution to the associated BSDE.
We assume that the conditions in the Remark 2 are valid.

Proposition 1.2.2. *Let $v \in C^{1,2}([0, T) \times \mathbb{R}^n) \cap C^0([0, T] \times \mathbb{R}^n)$ a classical solution to the semilinear PDE (1.10)-(1.11). Let v satisfy the linear growth condition and, for some positive constants C and q, we have $|D_x v(t, x)| \leq C(1 + |x|^q)$ for all $x \in \mathbb{R}^n$. Then, the pair of stochastic processes*

$$Y_t = v(t, X_t), \quad Z_t = \langle \sigma(t, X_t), D_x v(t, X_t) \rangle, \quad 0 \leq t \leq T,$$

is a solution to the BSDE in (1.9).

Proof. The proof directly follows from the Itô's formula on $v(t, X_t)$. We observe that $(Y, Z) \in \mathbb{S}^2(0,T)^k \times \mathbb{H}^2(0,T)^{k\times d}$ in accord with the linear growth condition of v and $D_x v$. □

The Proposition 1.2.2 is theoretically important but in some circumstances the practical use could be more difficult. Often, the solution to PDE could not be $C^{1,2}$ or, in some cases, the PDE could not have any solution. Excluding the latter, it may be useful to find an "admissible" solution. These do not satisfy the regularity condition of the classical solution and are called **Viscosity Solutions**.

Definition 1.3. Let $v : [0,T] \times \mathbb{R}^n \to \mathbb{R}$ be a locally bounded function, then:

- $v \in C([0,T] \times \mathbb{R}^n)$ is called a **Viscosity Subsolution** of (1.10)-(1.11), if $v(T,x) \leq g(x), \quad x \in \mathbb{R}^n$ and $\forall\ \Phi \in C^{1,2}([0,T] \times \mathbb{R}^n)$ whenever the map $v - \Phi$ attains a local maximum at $(t,x) \in [0,T) \times \mathbb{R}^n$, it holds:

$$\partial_t \Phi(t,x) + \mathcal{L}\Phi(t,x) + f(t,x,v(t,x), \langle \sigma(t,x), D_x \Phi(t,x) \rangle) \geq 0$$

- $v \in C([0,T] \times \mathbb{R}^n)$ is called a **Viscosity Supersolution** of (1.10)-(1.11), if $v(T,x) \geq g(x), \quad x \in \mathbb{R}^n$ and $\forall\ \Phi \in C^{1,2}([0,T] \times \mathbb{R}^n)$ whenever the map $v - \Phi$ attains a local minimum at $(t,x) \in [0,T) \times \mathbb{R}^n$, it holds

$$\partial_t \Phi(t,x) + \mathcal{L}\Phi(t,x) + f(t,x,v(t,x), \langle \sigma(t,x), D_x \Phi(t,x) \rangle) \leq 0$$

- $v \in C([0,T] \times \mathbb{R}^n)$ is called a **Viscosity Solution** of (1.10)-(1.11) if it is both a *Viscosity Subsolution* and a *Viscosity Supersolution.*

We note that in the above definition, the only hypothesis made on v is the continuity. Thus, we can say that it is a solution to the differential equation, *without* its differentiability.
The following result can be interpreted as the reverse of the Proposition 1.2.2.

We prove that the solution to the BSDE (1.9) provides a Viscosity Solution to the PDE (1.10)-(1.11).

Theorem 1.2.3. *The function $v(t,x) = Y_t^{t,x}$ is continuous on $[0,T] \times \mathbb{R}^n$ and it is a Viscosity Solution to the PDE* (1.10)-(1.11)*.*

Proof. • First of all we have to prove that the function $v(t,x) = Y_t^{t,x}$ is continuous. Let us choose $(t_1, x_1), (t_2, x_2) \in [0,T] \times \mathbb{R}^n$, where $t_1 \leq t_2$. We denote $X_s^i = X_s^{t_i,x_i}, i = 1,2$ and, conventionally, assume that $X_s^2 = x_2$, with $t_1 \leq s \leq t_2$. We indicate with $(Y_s^i, Z_s^i) = (Y_s^{t_i,x_i}, Z_s^{t_i,x_i})$, $i = 1,2$, which is well defined for $t_1 \leq s \leq T$. Applying the Itô's formula to $|Y_s^1 - Y_s^2|^2$ between $s = t \in [t_1, T]$ and $s = T$, we have

$$|Y_t^1 - Y_t^2|^2 = |g(X_T^1) - g(X_T^2)|^2 - \int_t^T |Z_s^1 - Z_s^2|^2 ds$$

$$+2\int_t^T (Y_s^1 - Y_s^2)(f(s, X_s^1, Y_s^1, Z_s^1) - f(s, X_s^2, Y_s^2, Z_s^2))ds$$

$$-2\int_t^T (Y_s^1 - Y_s^2)'(Z_s^1 - Z_s^2)dW_s.$$

As in the Theorem 1.1.3, the local martingale

$$\int_t^s (Y_u^1 - Y_u^2)^T (Z_u^1 - Z_u^2)dW_u, \;\; t \leq s \leq T$$

is uniformely integrable. Hence, by taking the expectation in the above equation, we have

$$E[|Y_t^1 - Y_t^2|^2] + E\left[\int_t^T |Z_s^1 - Z_s^2|^2 ds\right]$$

$$= E[|g(X_T^1) - g(X_T^2)|^2]$$

$$+2E\left[\int_t^T (Y_s^1 - Y_s^2)(f(s, X_s^1, Y_s^1, Z_s^1) - f(s, X_s^2, Y_s^2, Z_s^2))ds\right]$$

$$\leq E[|g(X_T^1) - g(X_T^2)|^2]$$

$$+2E\left[\int_t^T |Y_s^1 - Y_s^2||f(s, X_s^1, Y_s^1, Z_s^1) - f(s, X_s^2, Y_s^1, Z_s^1)|ds\right]$$

$$+2C_f E\left[\int_t^T |Y_s^1 - Y_s^2|(|Y_s^1 - Y_s^2| + |Z_s^1 - Z_s^2|)ds\right]$$

$$\leq E[|g(X_T^1) - g(X_T^2)|^2]$$

$$+E\left[\int_t^T |f(s, X_s^1, Y_s^1, Z_s^1) - f(s, X_s^2, Y_s^1, Z_s^1)|^2 ds\right]$$

$$+(1 + 4C_f^2)E\left[\int_t^T |Y_s^1 - Y_s^2|^2 ds + \frac{1}{2}E\left[\int_t^T |Z_s^1 - Z_s^2|^2 ds\right]\right],$$

where C_f is the Lipschitz constant of f in y and z. So we have

$$E[|Y_t^1 - Y_t^2|^2] \leq E[|g(X_T^1) - g(X_T^2)|^2] + E\left[\int_t^T |f(s, X_s^1, Y_s^1, Z_s^1) - f(s, X_s^2, Y_s^1, Z_s^1)|^2 ds\right]$$

$$+(1 + 4C_f^2)E\left[\int_t^T |Y_s^1 - Y_s^2|^2 ds\right]$$

and, by the Gronwall's lemma, we then obtain:

$$E[|Y_t^1 - Y_t^2|^2] \leq C\Big\{E[|g(X_T^1) - g(X_T^2)|^2]$$

$$+E\left[\int_t^T |f(s, X_s^1, Y_s^1, Z_s^1) - f(s, X_s^2, Y_s^1, Z_s^1)|^2 ds\right]\Big\}.$$

This last inequality, combined with the continuity of f and g in x and with the continuity of $X^{t,x}$ in (t, x), allows us to prove the mean-square continuity of $\{Y_s^{t,x}, x \in \mathbb{R}^n, 0 \leq t \leq s \leq T\}$ (*i.e.* we say that a $\mathbb{R}^d$-valued stochastic process Z is *mean-square continuous* in $s \in \mathbb{R}^d$ if $E[|X_s|^2] < \infty$ and $\lim_{x \to s} E[|Z(x) - Z(s)|]^2 = 0$). It follows the continuity of $(t, x) \to v(t, x) = Y_t^{t,x}$. The final condition (1.11) is easily satisfied.

- Now, we have to prove that $v(t, x) = Y_t^{t,x}$ is a Viscosity Solution to the PDE (1.10). We only show the Viscosity Subsolution property, the Viscosity Supersolution property is similarly proved.
  Let Φ be a smooth test function and $(t, x) \in [0, T) \times \mathbb{R}^n$, such that (t, x) is a local maximum of $v - \Phi$. We suppose, without loss of generality,

that $v(t,x) = \Phi(t,x)$. We argue by contradiction by assuming that

$$-\frac{\partial \Phi}{\partial t}(t,x) - \mathcal{L}\Phi(t,x) - f(t,x,v(t,x),\langle \sigma(t,x), D_x\Phi(t,x)\rangle) > 0.$$

From the continuity of f, Φ and its derivatives, there exist $h, \epsilon > 0$ such that, for all $t \leq s \leq t+h, |x-y| \leq \epsilon$,

$$v(s,y) \leq \Phi(s,y) \tag{1.15}$$

$$-\frac{\partial \Phi}{\partial t}(s,y) - \mathcal{L}\Phi(s,y) - f(s,y,v(s,y),\langle \sigma(s,y), D_x\Phi(s,y)\rangle) > 0 \tag{1.16}$$

Let $\tau = \inf\{s \geq t : |X_s^{t,x} - x| \geq \epsilon\} \wedge (t+h)$, and consider the pair

$$(Y_s^1, Z_s^1) = (Y_{s\wedge\tau}^{t,x}, \mathbf{1}_{[0,\tau]}(s) Z_s^{t,x}), \quad t \leq s \leq t+h.$$

By concluding, (Y_s^1, Z_s^1) is a solution to the BSDE

$$-dY_s^1 = \mathbf{1}_{[0,\tau]}(s) f(s, X_s^{t,x}, u(s, X_s^{t,x}), Z_s^1)ds - Z_s^1 dW_s, \quad t \leq s \leq t+h,$$

$$Y_{t+h}^1 = v(\tau, X_\tau^{t,x}).$$

On the other hand, by the Itô's formula, the pair

$$(Y_s^2, Z_s^2) = (\Phi(s, X_{s\wedge\tau}^{t,x}), \mathbf{1}_{[0,\tau]}(s) \left\langle \sigma(s, X_s^{t,x}), D_x\Phi(s, X_s^{t,x})\right\rangle), \quad t \leq s \leq t+h,$$

satisfies the BSDE

$$-dY_s^2 = -\mathbf{1}_{[0,\tau]}(s)(\frac{\partial \Phi}{\partial t} + \mathcal{L}\Phi)(s, X_s^{t,x}) - Z_s^2 dW_s, \quad t \leq s \leq t+h,$$

$$Y_{t+h}^1 = \Phi(\tau, X_\tau^{t,x}).$$

From inequalities (1.15)-(1.16) and from the Theorem 1.1.3, we can conclude $Y_0^1 < Y_0^2$, *i.e.* $v(t,x) < \Phi(t,x)$, that is a contradiction.

□

1.3 Numerical Issues

In the previous section, we established a link between parabolic nonlinear PDEs and stochastic processes. We involved above all Backward Stochastic Differential Equations (BSDEs). The aim of this section is to introduce the main idea which stands behind classical numerical methods to approximate the solution to a BSDE. Once the solution is approximated and then applying the Feynman-Kac formula, it is possible to derive the solution of the associated PDE.
The first step to solve BSDE numerically is the temporal discretization:

- *Euler Scheme for SDEs (Forward Process).* We consider the temporal discretization of the interval $[0, T]$. The set $\pi = \{t_0 = 0 < t_1 < \cdots < t_n = T\}$, with $|\pi| := \max_{i=1,\cdots,n} \Delta t_i$, $\Delta t_i := t_{i+1} - t_i$, denotes the temporal partition. We approximate the forward diffusion process X of (1.9) by the following Euler Scheme X^π

$$X^\pi_{t_{i+1}} := X^\pi_{t_i} + b(X^\pi_{t_i})\Delta t_i + \sigma(X^\pi_{t_i})\Delta W_{t_i}, \quad i < n, \;\; X^\pi_0 = x$$

  where $\Delta W_{t_i} = W_{t_{i+1}} - W_{t_i}$.

- *Euler Scheme for BSDEs (Backward Process).* First, we approximate the terminal condition $Y_T = g(X_T)$ by substituting X with the Forward Euler Scheme: $Y_T \simeq g(X^\pi_T)$. Then we approximate the backward process of (1.9) with the following Euler Scheme:

$$\begin{aligned} Y_{t_i} &= Y_{t_{i+1}} + \int_{t_i}^{t_{i+1}} f(X_s, Y_s, Z_s)ds - \int_{t_i}^{t_{i+1}} Z_s dW_s \\ &\simeq Y_{t_{i+1}} + f(X^\pi_{t_i}, Y_{t_i}, Z_{t_i})\Delta t_i - Z_{t_i}\Delta W_{t_i}. \end{aligned}$$

We define the time discrete approximation of BSDE:

(1) by taking the conditional expectation with respect to $\mathcal{F}_{t_i}$,

$$Y_{t_i} \simeq E[Y_{t_{i+1}}|\mathcal{F}_{t_i}] + f(X^\pi_{t_i}, Y_{t_i}, Z_{t_i})\Delta t_i$$

(2) by multiplying both sides by ΔW_{t_i} and taking the conditional expectation,

$$0 \simeq E[Y_{t_{i+1}} \Delta W_{t_i} | \mathcal{F}_{t_i}] - Z \Delta t_i.$$

From (1)-(2) we obtain the Euler Scheme for the backward solution (Y^π, Z^π):

$$\begin{cases} Z^\pi_{t_i} = E\left[Y^\pi_{t_{i+1}} \frac{\Delta W_{t_i}}{\Delta t_i} | \mathcal{F}_{t_i} \right] \\ Y^\pi_{t_i} = E[Y^\pi_{t_{i+1}} | \mathcal{F}_{t_i}] + f(X^\pi_{t_i}, Y^\pi_{t_i}, Z^\pi_{t_i}) \Delta t_i, \quad i < n \end{cases} \tag{1.17}$$

with final condition $Y^\pi_{t_n} = g(X^\pi_{t_n})$.

Remark 4. The above schema is *implicit* since $Y^\pi_{t_i}$ appears in both sides of the Equation (1.17). By the Lipschitz condition of f, for Δt_i small enough, then the implicit scheme equation can be solved by a fixed point method, by substituting the second equation in (1.17) with

$$Y^\pi_{t_i} = E[Y^\pi_{t_{i+1}} + f(X^\pi_{t_i}, Y^\pi_{t_{i+1}}, Z^\pi_{t_i}) \Delta t_i | \mathcal{F}_{t_i}].$$

The rate of convergence is the same.

Remark 5. It is possible to prove, by Lipschitz regularity conditions on f and g, that the discrete time approximation error is

$$\mathcal{E}(\pi) \leq C |\pi|^{\frac{1}{2}}$$

with C independent of π.

Remark 6. The practical implementation of the numerical method in (1.17) requires to compute the conditional expectation with respect to $\mathcal{F}_{t_i}$. This calculation may be onerous in some cases. However, since we work into a Markovian framework, the expectations can be transorm in regressions:

$$E[Y^\pi_{t_{i+1}} | \mathcal{F}_{t_i}] = E[Y^\pi_{t_{i+1}} | X^\pi_{t_i}], \quad E[Y^\pi_{t_{i+1}} \Delta W_{t_i} | \mathcal{F}_{t_i}] = E[Y^\pi_{t_{i+1}} \Delta W_{t_i} | X^\pi_{t_i}]$$

These regressions can be approximate by statistical methods like *Least Squares Regression* ([LS01], [LGW06]), *Integration by Parts* ([BET04]) and *Quantization* ([PPP04]). The main advantage of these methods is that, using Monte

Carlo simulations, they are less affected by the "curse of dimensionality" problem, typical of deterministic methods.

In the next chapter we introduce some fundamental concepts about Artificial Neural Networks. Then we reformulate the BSDEs numerical approximation method in a Machine Learning framework. The numerical results will be compared with standard probabilistic algorithms. We highlight advantages and disadvantages of using this new method.

Chapter 2

An Introduction to Machine Learning

We are going to illustrate the principal characteristics and peculiarities of Machine Learning, a recent field in computer science that uses statistical techniques to give computer systems the ability to "learn" through data. The tools and knowledge that we present in this chapter will be useful to better understand the BSDE solver algorithm of the last chapter. This algorithm, by Deep Learning techniques, provides a numerical approximation method to solve high dimensional, nonlinear PDEs.
After a brief introduction to Machine Learning [SV08], we propose the main learning algorithms and consider some practical problems. Next, we present the Neural Network theory ([B06], [H09], [R13]) and Deep Learning theory [GBC16]. Finally, we expose the algorithm which stands behind the Neural Network learning techniques. We also report and analyze some optimization methods, that are used for training [GS03].

2.1 Basics and Applications

Machine Learning (ML) is a field of Artificial Intelligence (AI) that studies how computer systems "learn" using data. This can be interpreted as a different programming paradigm. Unlike *Procedural Programming*, a Machine

Learning algorithm does not required a sequence of detailed instructions. It learns from experience (data). The computer scientist Tom M. Mitchell (1997) provided a widely quoted definition of the algorithms studied in the ML field:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."[1]

Hence, the paradigm shift induces the programmer to implement an algorithm that allows the machine to develop its own "logic".

In the last few years, Machine Learning has had a wide impact of applications in every aspect of technology. Think of the so-called *Recommender Systems* (or *Recommendation Systems*), a subclass of information filtering systems that seeks to predict the preference a user would give to an item or a web service. Recommender Systems are utilized in a variety of fields and aspects of everyday life. Think of the most popular entertainment services such as, YouTube, Spotify or Netflix. These suggest to the user contents in line with the analyzed profile.

Also email spam filters are an implementation of ML techniques, they allow to avoid possible phishings or infected files.

Machine Learning is widely used in biomedical research [MP99]. These kind of algorithms support increasingly accurate predictions to avoid the outbreak of epidemics and are also used to detect tumors.

In the next sections we introduce three methods that are crucial for ML techniques.

2.1.1 Supervised Learning

The majority of Machine Learning applications use **Supervised Learning**. In this case, the machine receives a labeled dataset as input[2], this is

[1] [M97]

[2] Each Machine Learning system receives input data characterized by a collection of vectors (the so-called *feature vector*). A good choice of features could be very challenging. It is important to select which information is useful and which is redundant. This process (also called *feature extraction* or *feature selection*) permits to reduce dimensionality and optimizing system performance.

known as the **training set**. Supervised Learning requires that the algorithm's possible outcomes (ouputs or targets) are already known. Therefore, the data in the training set are already labeled with the correct answers. For example, this kind of learning is commonly used in email spam filters.
The training set used for classification problems, is represented like a sequence of pairs

$$\{(\boldsymbol{x_1}, y_1), \cdots, (\boldsymbol{x_n}, y_n)\}$$

where $\boldsymbol{x_i}$ stands for an input vector and y_i stands for the corresponding class label. Supervised Learning algorithms share the same crucial characteristics: training takes place through the minimization of a certain loss function. This function represents the error made by the system in classifying an input $\boldsymbol{x}_i$ with the label $\hat{y}_i$, instead the target y_i.
The widely used loss function is the so-called *Mean Squared Error*:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}.$$

MSE measures the average squared difference between the estimated and the correct values. With this measure, during the system training, the machine adjusts its own parameters (often called "weights") to reduce the MSE. The function that defines the Input-Output process of a ML system is completely defined by its parameters.
There are many optimization algorithms that are used for minimizing the loss function and finding the optimal values for system parameters. In the Neural Networks section we introduce some of these.
Once training is complete, it is appropriate to see if the model is correct, using **test** (or **evaluation**). Evaluation allows us to test our model against data that has never been used for training. This is meant to be representative of how the model performs well in the real world. A good rule of thumb for training-evaluation is to split the order of 80/20 or 70/30. Much of this depends on the size of the original source dataset[3].

[3]One of the most critical problems in Machine Learning theory is the training-test data split. Choosing the right ratio of the source dataset affects heavily the system performances. This topic is beyond the aim of this work. In merit of this we refer to some

2.1.2 Unsupervised Learning

On the other hand, **Unsupervised Learning** is when you only have input data without corresponding output labeled variables. The purpose of Unsupervised Learning is to model the underlying structure or distribution of the dataset in order to learn more about the experiments. Algorithms are left to their own to organize data. Looking for common features and identifying the internal structure in the data. That is to say, if the aim of Supervised Learning is to estimate the conditional probability distribution $p(y|\boldsymbol{x})$. Then the main purpose of Unsupervised Learning is to approximate the probability distribution $p(\boldsymbol{x})$ of the whole dataset. We notice that $p(y|\boldsymbol{x})$ represents the probability of the target y, given a features vector $\boldsymbol{x}$.
Therefore, some of the most used methods in Unsupervised Learning are *Kernel Density Estimation* and *Mixture Models*. However, a widely used technique is **Clustering**. In this case, the system splits the dataset into several groups (or clusters) of elements that share some common features.

2.1.3 Reinforcement Learning

In **Reinforcement Learning** the system does not receive a fixed dataset. Instead it receives a logical or real value after completion of a sequence. This defines whether the decision is correct or incorrect. In this case the result of the Machine Learning algorithm is conditioned by the enviroment. After each action of the model it receives a reward signal (called *reinforcement*) from the enviroment. If the action is correct the feedback will be positive and the algorithm continues to follow the same strategy. Otherwise, if the feedback is negative, the ML model is forced to search for an alternative startegy.
Reinforcement Learning is widely used where the system works in high-dimensional spaces. Atari games are a widely accepted benchmark for Reinforcement Learning. The implemented algorithm was able to play Atari games and learn the best strategy to win in a competitive time block. Using Supervised or Unsupervised Learning presents two main drawbacks. First,

results of Isabelle Guyon [G97] and [GMSV98].

the model receives high dimensional sensory input through RGB images (game screen). In return, the problem associated with the curse of dimensionality becomes unbearable. The other issue is the training complexity. With Supervised (or Unsupervised) Learning, the model needs millions of hours of play to give a correct response. Hence it requires a large quantity of memory and time. With Reinforcement Learning the machine training is quite simple, the model receives a reward every time one of its actions leads to winning the game.
This example explains the key idea behind the Reinforcement Learning. In order to study this topic in depth, we refer to the papers by Sutton and Barto [SB98] and by Bertsekas and Tsitsiklis [BT95].

2.1.4 Overfitting and Underfitting Issues

We have already mentioned that one of the dominant skills of Machine Learning algorithms is the "generalization capacity". The machine ability to generalize means solving new tasks by using a different dataset from the training set. We can define an error measure of training process. This is called *Training Error*. In the previous section we have seen that the problem of minimizing the Training Error can be transformed in an optimization problem. The difference between Machine Learning and Mathematical Programming can be represented by the introduction of the Generalization Error, also called *Test Error*. Generalization Error is defined by the expectation of the error on a new input, hence chosen from the test set.
Discriminant features that measure the ML algorithm performances are:

- The capacity to reduce Training Error.
- The capacity to minimize the increase of the Test Error with respect to the measure of the Training Error.

These aspects are related to two well-known Machine Learning problems: the **Underfitting** and the **Overfitting**. These issues are often the cause of poor performance in Machine Learning.
Underfitting refers to a model that can neither represents the training data

nor generalizes to new data. An underfit Machine Learning model is not suitable and will result in poor performances in the training data. Overfitting, however, refers to a model that represents the training data correctly, but has poor performances on new data. In other words, overfitting happens when an ML algorithm learns the details and noise in the training data, loosing the ability to generalize. In this case, the Error Test is greater than the Training Test. A measure that controls the inclination of a model towards Overfitting or Underfitting is its **Capacity**. Informally, a model with low Capacity does not accurately approximate data in the trainig set (this is Underfitting). Meanwhile, an algorithm with high Capacity goes towards Overfitting. It losses the ability to generalize, memorizing the training set patterns. The Vapnik-Chervonenkis (VC)[4] Dimension is a mathematically rigorous formulation of Capacity, but it is more on the theoretical end, and there can be a large gap between the VC Dimension and the model's actual Capacity. A very rough and easy way to estimate Capacity is to count the number of parameters. The more parameters, the higher the Capacity.
There are several tricks to avoid these issues. If we have a great number of data, then we obtain good performances in the training and in the test phase. Another way to avoid Overfitting is to limit the training time. If we choose a high number of iterations (*epochs*), the model will perfectly memorize the data training patterns, with poor performances in the test phase.

2.2 Artificial Neural Networks

An Artificial Neural Network (ANN) is a mathematical computing system vaguely inspired to the biological neural network that controls functionalities of the mammals brain. The most important component of these systems is the *interconnection* between elementary units of calculation (or nodes) called **neurons**. The network learns from input data through a training process (see Section 2.1). The information mined from the data is stored in the internal parameters of the network. These parameters are called **synaptic weights**

[4][V13]

and they refer to the amplitude of a connection between two nodes[5].
The Artificial Neural Network model has many semplifications compared to the biological network. Results obtained in an ANN framework cannot be transferred on real brain models. However, the use of this statistical-programming algorithm ranges from Voice Recognition problem to Data Mining and from Face Detection to Financial Engineering.
A Neural Network can be mathematically formalized by a directed graph. Here neurons are the nodes of this graph. Nodes receive signals from the external enviroment or from other neurons. They usually transform and transmit information like complex elaboration units. Inside each neuron there is an **activation function** which regulates and propagates signals throughout the network[6]. As we mentioned above, the main aim of a Machine Learning algorithm is to approximate an Input-Output function. Also, when we train a Neural Network there are several parameters that allow us to improve the Input-Output function approximation. We indicate some of these parameters:

- **Neurons**, or elementary units are characterized by a more or less complex internal structure and by their activation function.
- The **Neural Network architecture** is defined by the number of neurons, the structure and the directing of synaptic connections.
- **Synaptic weights** represent the internal parameters of a Neural Network and they are adjusted by specific learning techniques. We will describe learning algorithms for Neural Networks in the next sections.

We distinguish different kinds of Neural Networks by varying the above parameters. The connections topology is a central feature. A **Feedforward Neural Network** is a network wherein the connective structure can be represented by a directed acyclic graph. The data elaboration takes place from the starting point to the end and there is no loop inside the network. A

[5]In biology it corresponds to the ammount of the firing effect that one neuron has on another.
[6]The *Threshold Potential* is the neurological counterpart of activation function.

Recurrent Neural Network has feedback connections and cycles between units. This kind of Artificial Neural Network is more complex but it owns fascinating qualities. Feedforward Networks are widely used to predict outcomes and classify different items. Recurrents are used to simulate systems that own "internal memory". They are applied to forecast stock market prices, speech recognition and handwriting recognition.
There are other features to characterize the nature of a neural network. These refer to the way in which the system receives the training set data.

- **On-Line Learning:** Items are sequentially provided to the system. The machine uses these data to improve the performances of the next iteration.

- **Batch Learning:** We assume that the entire dataset is available before the training phase starts. In this case the training takes place once on the whole set of elements.
  There is a variation of this learning, named **Mini-Batch Learning**. In this case the system is trained on a fixed size subset of data[7].

For more details we refer to the essays by Bishop [B06] and Haykin [H09].

2.2.1 The Perceptron

The first Neural Network model was proposed in the 1960s by the American psychologist Frank Rosenblatt and it took the name of **Rosenblatt Perceptron** (a.k.a. Formal Neuron). The Perceptron is characterized by a neuron local memory that consists of a vector of weights. Input data are multiplied by these weights, which represent the strength of connections. The weighted algebraic sum obtained is compared with a threshold value by using the activation function. If the result is greater than the threshold value, then the output is equal to 1 and the signal is transmitted along the network. Otherwise, the signal is inhibited and it cannot participate in the

[7]The choice of correct batch size is a necessary step to improve the network performances, in Appendix B we describe a fundamental approach which gives theoretical basis to the Neural Network learning.

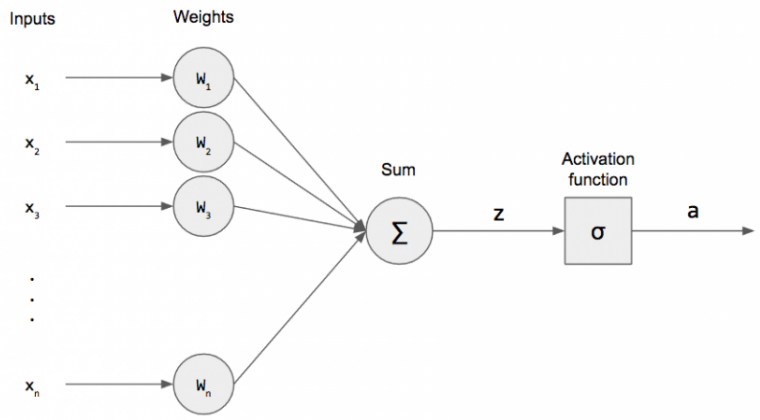


Figure 2.1: The Rosenblatt Perceptron architecture

final output. In this case the activation threshold value is equal to -1 (or 0). Let us try to formalize this concept. We assume that $x \in \mathbb{R}^n$ is the input vector, we denote by $w \in \mathbb{R}^n$ the vector of synaptic weights and by $\theta \in \mathbb{R}$ the threshold value (or *bias*). Let $y \in \{-1, 1\}$ be the output of the Perceptron and g be the activation function. Hence we set

$$y(x) = g\left(\sum_{i=1}^{n} w_i x_i - \theta\right) = g(w'x - \theta).$$

Two widely used activation functions in Perceptron applications are the *sign function* and the *Heaviside step*.

The Figure 2.1 shows a simple Formal Neuron scheme.

A basic result for Perceptron theory states that, with an appropriate choice of weights and threshold, it is possible to approximate the main logical functions, like NOT, AND and OR. The Perceptron model fits as a classifier of linearly separable data.

As we have just seen, the Perceptron architecture is quite simple and its practical implementation is very easy. However, it has numerous limitations. The American cognitive scientist Marvin L. Minsky and the mathematician Seymour Papert pointed out the major drawbacks in the Rosenblatt model ([MP70]). After the training, the Perceptron approximates only linearly separable functions. For example, it fails when it tries to estimate the logical

function XOR (exclusive disjunction). In the 1970s, after the Minsky and Papert essay, the interest in Neural Networks drastically decreased.

2.2.2 Deep Neural Networks

Immediately after the paper by Minsky and Papert, Machine Learning scientists hypothesized that a multilayer Perceptron network could overcome the drawbacks of a single Perceptron. However, the high computational cost made the implementation infeasible. Only in the late 1980s, thanks to the development of more advanced technologies, it was possible to construct the first Multilayer Feedforward Neural Network. This kind of network consists of a sequence of neuron layers connected in cascade. This is a part of a larger family of Neural Networks, the **Deep Neural Networks**. Deep Neural Networks can approximate any kind of continuous function on a compact set, simply by adjusting the weights and the activation functions of the model. This kind of Machine Learning structure allows us to solve nonlinear classification problems. The appropriate choice of vector of weights can be seen as a nonlinear optimization problem. Generally speaking, Deep Neural Network learning is more complex than the Perceptron learning, due to the nonlinear optimization. Nevertheless, the potential of Neural Networks with more than one layer cannot be compared with Perceptron features. We will present some important theorems of Deep Neural Networks in the next section. Deep Neural Networks can cluster several millions of data, by identifying similar features in different elements (let us think of the *Smart-Photo Album*). This kind of network can also perform an authomatic feature extraction process, unlike the other machine learning algorithms. Intuitively, during the training, the network recognizes the correlations among relevant features and it optimizes the result.
In the next paragraph we are going to explore the structure and learning algorithms of Feedforward Deep Neural Networks (or Feedforward Neural Networks). Let us make a clarification, the theory of Recurrent Neural Networks is little different from the Feedforwards and we will not consider it in this thesis.

Deep Neural Networks Architecture

The architecture of a Deep Neural Network is generally characterized by the following structure:

- An **input layer** consists of n units without elaboration capacity, where n is the number of network entrances. We notice that n is also the dimensionality of the vector of features, which describes the data characteristics.
- A set of neurons which are split into $L \geq 2$ **layers** where:
  - $L - 1$ consist of neurons whose outputs are connected with the inputs of the successive layer. These layers are called **hidden layers**.
  - The last layer consists of $K \geq 1$ neurons whose outputs correspond to the network outputs. The network output is $y \in \mathbb{R}^K$ and this layer is called the **output layer**.
- A set of directed and weighted edges which represent all the possible synaptic connections among the hidden layers and input and output layers. Let us suppose that there is no connection among neurons of the same layer and there is no feedback loop between the inputs of one layer and the outputs of the previous one.

An example of Multilayer Deep Neural Network is shown in Figure 2.2. Each neuron is characterized by an activation function $g_j^{(l)} : \mathbb{R} \to \mathbb{R}$, where j represents the neuron index into the l-th layer and $l = 1, \cdots, L$. This function acts on a weighted combination of the vector of inputs and threshold value $w_{j0}^{(l)}$, let $a_j^{(l)}$ indicate this sum. If we denote the output of a single neuron by $\hat{y}_j^{(l)}$, then we have:

$$a_j^{(1)} = \sum_{i=1}^{n} w_{ji}^{(1)} x_i - w_{j0}^{(1)}, \quad \hat{y}^{(1)} = g_j^{(1)}(a_j^{(1)}).$$

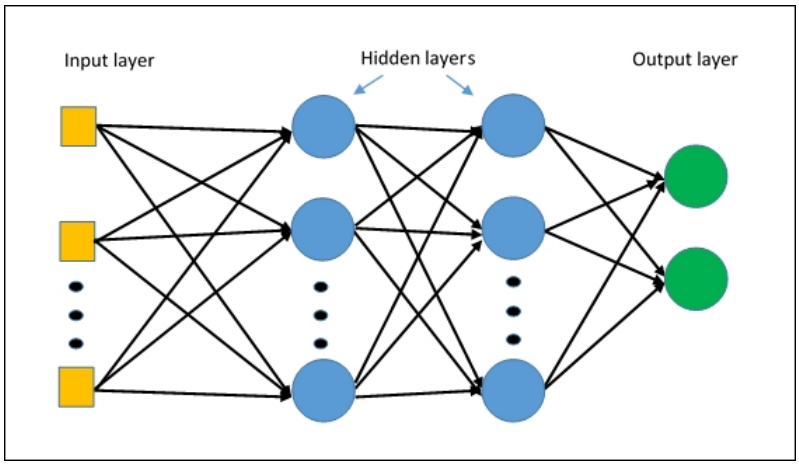


Figure 2.2: A simple Deep Neural Network structure

Let $N^{(l)}$ be the number of neurons in the l-th layer. For the generic j-th neuron of the $l > 1$-th layer, we have:

$$a_j^{(l)} = \sum_{i=1}^{N^{(l-1)}} w_{ji}^{(l)} z_i^{(l-1)} - w_{j0}^{(l)}, \quad \hat{y}^{(l)} = g_j^{(l)}(a_j^{(l)})$$

Figure 2.3 shows activation functions that are frequently used in Neural Network applications.

Approximation Properties for Deep Neural Networks

In this section we introduce some central results about the *approximation theory* for Deep Neural Networks[8]. Let

$$\mathcal{M}(g) = span\{g(\langle w, x\rangle - \theta), \ \ \theta \in \mathbb{R}, \ \ w \in \mathbb{R}^n\}$$

be the set of all linear combinations from a set of activation functions applied over an affine transformation of x. This transformation is defined by w and θ.

[8] [P99]

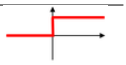
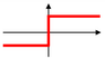
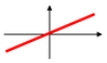
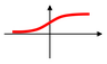
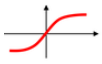
Activation Function	Equation	Example of Use	Plot
Heaviside Step	$\phi(x) = \begin{cases} 0, & x < 0 \\ 0.5, & x = 0 \\ 1, & x > 0 \end{cases}$	Perceptron	
Sign	$\phi(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases}$	Perceptron	
Linear	$\phi(x) = x$	Linear Regression	
Logistic (Sigmoid)	$\phi(x) = \frac{1}{1+e^{-x}}$	Logistic Regression, Deep NN	
Tangente Iperbolica	$\phi(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	Multi-Layer NN	
Rectifier (ReLu)	$\phi(x) = \max(0, x)$	Deep NN	

Figure 2.3: Activation functions for Artificial Neural Networks

Theorem 2.2.1 (Pinkus, 1996)**.**
Let $g \in C(\mathbb{R})$. In the topology of the uniform convergence on compact sets, the set $\mathcal{M}(g)$ is dense in $C(\mathbb{R}^n)$ if and only if g is not a polynomial.

The corollary follows:

Corollary 2.2.2. *Let $\Omega \subset \mathbb{R}^n$ be a compact set, let us assume $\epsilon > 0$ and g is an activation function (continuous and not polynomial). Given a function $f \in C(\mathbb{R}^n)$, it is possible to construct a two-layer Neural Network (with an appropriate choice of number of neurons, vector of weights and biases), such that the Input-Output function $y \in \mathcal{M}(g)$ satisfies the condition:*

$$\max_{x \in \Omega} |f(x) - y(x)| < \epsilon.$$

In other words, each two-layer Neural Network, with 1 hidden layer, can approximate any continuous function on a compact subset of $\mathbb{R}^n$.
Two-layer Neural Networks allow us to construct data interpolating function.

Theorem 2.2.3 (Pinkus, 1999)**.**
Let $g \in C(\mathbb{R})$ and let g not be a polynomial. Given K distinct points $\{x^i\}_{i=1}^K \subset \mathbb{R}^n$ and K numbers $\{\alpha_i\}_{i=1}^K \subset \mathbb{R}$, then exist K vectors $\{w_j\}_{j=1}^K \subset$

$\mathbb{R}^n$ *and* $2K$ *numbers* $\{v_j\}_{j=1}^K$, $\{\theta_j\}_{j=1}^K \subset \mathbb{R}$ *such that*

$$\sum_{j=1}^{K} v_j g(\langle w_j, x^i \rangle - \theta_j) = \alpha_i, \quad i = 1, \cdots, K.$$

2.3 Training Algorithms for Neural Networks

Once we have chosen the network architecture (number of hidden layers and number of neurons for each layer), we must adapt the optimal weights $w \in \mathbb{R}^n$, by training the learning system. In this section we introduce the most widely used method, the **backpropagation algorithm**[9]. Applying an optimization algorithm like Gradient, Conjugate Gradient or a Quasi-Newton method to Neural Networks it can be quite difficult, especially when the network is very deep (it owns many hidden layers). To solve this drawback, in the mid-1980s the backpropagation algorithm was developed. This is a sort of Gradient Descent method which it was built *ad hoc* for Neural Networks.

2.3.1 Differentiable Activation Function

The backpropagation algorithm looks for the minimum of the loss function in the weight space using the Gradient Descent method. Since this algorithm requires computation of the gradient of the loss function at each iteration, it is important to guarantee its **continuity** and **differentiability**. The Heaviside Step function, that is used in the training of the Perceptron, does not satisfy continuity and differentiability conditions. By the Figure 2.3 it results that the most common used function in Neural Networks learning is the real-valued **Logistic Function** (**Sigmoid**), $s_c : \mathbb{R} \to (0, 1)$ which is defined by

$$s_c(x) = \frac{1}{1 + e^{-cx}}$$

The constant c characterizes the Sigmoid form, higher values of c bring the shape of the Sigmoid closer to the Heaviside function. In the limit $c \to \infty$ the Sigmoid converges to a Step function at the origin. To simplify the notation,

[9] [RHW86]

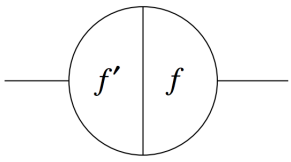


Figure 2.4: Internal structure of a neuron

we assume that $c = 1$.
We consider a Deep Neural Network with n input and m output. Let

$$\{(\boldsymbol{x_1}, \boldsymbol{y_1}), \cdots, (\boldsymbol{x_p}, \boldsymbol{y_p})\}$$

be the training set. This consists of p ordered pairs of vectors in $\mathbb{R}^n \times \mathbb{R}^m$. We denote by $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$, $i = 1, \cdots, p$ the set of network outcomes with respect to the training set elements. The aim is to minimize the loss function

$$E = \frac{1}{2p} \sum_{i=1}^{p} \|\boldsymbol{y_i} - \hat{\boldsymbol{y}}_{\boldsymbol{i}}\|^2.$$

The backpropagation algorithm is used to seek the local minimum of this function. Since the neural network is comparable with a complex chain made by composing functions, we expect that the central idea of this algorithm is the **chain rule**. We use this rule to compute the derivative of the composition of two or more functions.

Each neuron in the network has a composite structure, as it is shown in Figure 2.4. This depiction is called **B-Diagram** (Backpropagation Diagram). The right side is used to compute the Input-Output function of the neuron, while the left side deals with the derivative of the same function. Both functions are computed at each element of the same input dataset. We report the two main steps of backpropagation algorithm. In the next section we will describe the details of those steps.

(1) **Feedforward phase**: the training set elements are passed by the network (there are three ways to do it: *on-line*, *batch* or *mini-batch*). The direction is from the left side to the right side. Input-Output function

and its derivative are both computed at each element. This information about Input-Output function is stored into the corresponding neuron. In this phase, only the right side is used to transmit information to the next neuron.

(2) **Backpropagation phase**: this consists of the backpropagation of signal error along the network. In this phase we only use the left side of each neuron. The informations coming from the right side of the network are added together and they are multiplied by the derivative, which is contained in the left side of the neuron.

2.3.2 Backpropagation Algorithm

At first, synaptic weights of the network are randomly chosen. For the sake of simplicity, we consider a Neural Netwok with only one hidden layer. The backpropagation algorithm can be naturally extended to the multilayer case.
However, there are 4 steps:

(i) Feedforward computation

(ii) Backpropagation to the output layer

(iii) Backpropagation to the hidden layer

(iv) Weights update

As in optimization theory, the **early stopping** methods are concerned with the problem of choosing a time to stop the process. In order to maximize an expected reward or minimize an expected cost.
Here are some examples:

- The sequence is interrupted when the last value is in the neighborhood of a local minimum. In other words, when the Euclidean norm of the function gradient is less than a fixed threshold value.

- When the error variation percentage between two consecutive epochs is sufficiently small.

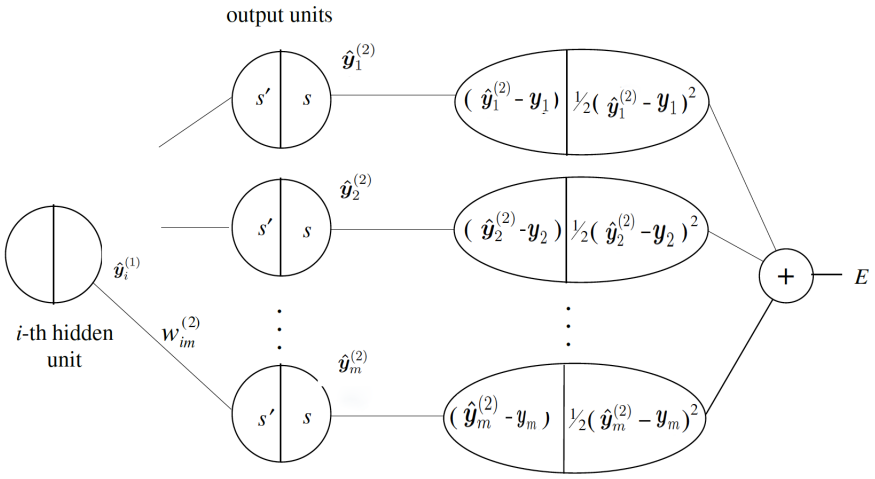


Figure 2.5: Extended multilayer network for the computation of E

- The learning algorithm is stopped when it reaches the maximum number of iterations.

Feedforward computation

The input vector $\boldsymbol{x}$ is presented to the network. The vectors $\hat{\boldsymbol{y}}^{(1)}$ and $\hat{\boldsymbol{y}}^{(2)}$ are, respectively, the output vector produced by the first layer and the output vector produced by the second layer. They are computed and stored, as we see in Figure 2.5. The evaluated derivatives of the activation functions are also stored in each neuron.

Backpropagation to the output layer

We are looking for the value of partial derivatives $\partial E/\partial w_{ij}^{(2)}$. We denote by $w_{ij}^{(2)}$ the weight of the synaptic connection between the i-th neuron of the hidden layer and the j-th neuron of the output layer. By the definition of derivative of sigmoid function, we have $\dot{s}_j = \hat{y}_j^{(2)}(1 - \hat{y}_j^{(2)})$. By multiplying the terms in the left side of each unit we obtain that the backpropagated error in this step is:

$$\delta_j^{(2)} = \hat{y}_j^{(2)}(1 - \hat{y}_j^{(2)})(\hat{y}_j^{(2)} - y_j),$$

and the partial derivative we are looking for is

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = [\hat{y}_j^{(2)}(1-\hat{y}_j^{(2)})(\hat{y}_j^{(2)}-y_j)]o_i^{(1)} = \delta_j^{(2)}\hat{y}_i^{(1)}.$$

Backpropagation to the hidden layer

Now we want to compute the partial derivatives $\partial E/\partial w_{ij}^{(1)}$. Each neuron j in the hidden layer is connected to each unit q in the output layer, with an edge of weight $w_{jq}^{(2)}$, for $q = 1, \cdots, m$. The backpropagated error is

$$\delta_j^{(1)} = \hat{y}_j^{(1)}(1-\hat{y}_j^{(1)})\sum_{q=1}^{m} w_{jq}^{(2)}\delta_q^{(2)}.$$

Therefore, the partial derivative is

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \delta_j^{(1)} o_i.$$

Weights update

After computing all partial derivatives the network weights are updated by using a Gradient Descent method. A constant γ defines the step length of the correction. The weights update is given by:

$$\Delta w_{ij}^{(2)} = -\gamma \hat{y}_i^{(1)}\delta_j^{(2)}, \quad for\ \ i = 1, \cdots, k+1;\ \ j = 1, \cdots, m,$$

and

$$\Delta w_{ij}^{(1)} = -\gamma \hat{y}_i \delta_j^{(1)} \quad for\ \ i = 1, \cdots, n+1;\ \ j = 1, \cdots, k,$$

The step length γ is also called the **learning rate**. A correct choice of this parameter fundamental for the convergence of the algorithm. The learning rate can be fixed or can be adaptive, in order to improve the algorithm's performances.
In the next section we will introduce some optimization algorithms that are used in Neural Network applications, in combination with the backpropagation method.

2.3.3 Optimization Algorithms

In this section we present some of the most used optimization methods within Neural Networks. For each algorithm we will explain briefly the main features.
Let $\{(x^{(i)}, y^{(i)})\}$ be the input-target set of pairs and $f(x; w)$ be the Input-Output function of the neural network, which depends on the input values and the synaptic weights.

Stochastic Gradient Descent (SGD)

The Stochastic Gradient Descent algorithm[10] is one of the most used methods in practical applications. It is very simple to implement and the computational cost is quite low. Like every Gradient Descent method, the weight correction takes place along the negative direction of the gradient g. We consider a subset (*mini-batch*) of the dataset of size m, it follows that

$$g = \frac{1}{m} \sum_{i=1}^{m} \nabla_w E(f(x^{(i)}; w), y^{(i)})$$

$$\Delta w = -\gamma g$$

The objective function is the loss function E which is the difference between estimated and true values for a sample of data. The learning rate is heuristically fixed at 0.01. We observe that, if the step length is too big ($\gamma \gg 0.01$), then the method may not be converge. Instead, a learning rate that is too small ($\gamma \ll 0.01$) leads to slow convergence.

Momentum

SGD has trouble descending ravines, *i.e.* areas where the surface curves much more steeply in one dimension than in another, that are common around local minima. In this scenario, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local minimum. The momentum method accelerates SGD in the relevant

[10] [R16]

direction and dampens oscillations. The method uses the momentum α, which depends on previous iterations. Let g_t be the gradient of the objective function at iteration t.

$$v_{t+1} = \alpha v_t - \gamma g_t$$

$$w_{t+1} = w_t + v_{t+1}.$$

Usually, α is equal to 0.5 or 0.9.

RMSProp

The Root Mean Square Propagation method (RMSProp)[11] is an **adaptive** algorithm. Hence, the learning rate is adapted for each of the parameters. It provides good performance in practice. The running average is calculated in terms of mean squared,

$$E[g^2]_t = \eta E[g^2]_{t-1} + (1-\eta)\,\langle g, g\rangle$$

where $\eta \in [0, 1]$ is the exponential decaying factor (*forgetting* factor). Usually, $\eta = 0.9$. Intuitively, the choice of η defines how the previous iteration memory is important in the running average computation. The weights update is

$$\Delta w_t = -\frac{\gamma g_t}{\sqrt{E[g^2]_t + \epsilon}}.$$

We observe that the root square to the denominator indicates the mean square (RMS, root mean square). In this case the learning rate γ is dynamically controlled by the root mean square of the gradient norm. It has been added to the denominator the factor ϵ, in order to prevent it from tending to 0.

Adam

The Adaptive Moment Estimation method (Adam)[12] is the most popular today and it can be seen as a combination of RMSProp and Momentum

[11] [TH12]
[12] [KB14]

method. Adam uses the running average of the objective function gradient and its second momentum. The parameters update follows the below scheme:

$$M_{t+1} = \beta_1 M_t + (1 - \beta_1) g_t$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2) \langle g_t, g_t \rangle$$

and the bias correction

$$\hat{M} = \frac{M_{t+1}}{1 - (\beta_1)^{t+1}}$$

$$\hat{v} = \frac{v_{t+1}}{1 - (\beta_2)^{t+1}}.$$

The weight correction is

$$w_{t+1} = w_t - \gamma \frac{\hat{M}}{\sqrt{\hat{v} + \epsilon}}.$$

The term ϵ is used to ensure numerical stability. The parameters β_1 and β_2 are used to control the exponential decay of the gradient and its second momentum. Usually we set $\epsilon = 10^{-8}$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

2.4 Comparing Methods

We now compare the performances of the three methods that we have previously seen: Stochastic Gradient Descent (SGD), RMSProp and Adam. These algorithms have been applied to a practical problem. Localizing and detecting impacts on a metal plate equipped with piezoelectric sensors using an Artificial Neural Network. It is obvious how important this problem is. For instance, the fuselages and the wings of airplanes, as well as some parts of ships are continuously monitored by this kind of sensors. This topic has been studied by several engineering research groups, including the Bologna division of CNAF, the National Computing Center of INFN (Italian Institute for Nuclear Physics) in collaboration with ARCES (Advanced Research Cen-

Optimizer: SGD Activation Function: Sigmoid (Logistic)

Test Size	Batch Size	Neurons	Epoch	Loss	Time [ms]	Early Stop
35	1	4	10	0.026939	1484.68	No
25	1	4	50	0.031312	3107.15	No
35	1	4	50	0.031956	2265.69	No
25	25	4	50	0.033919	995.13	No
25	1	4	10	0.035291	1122	No

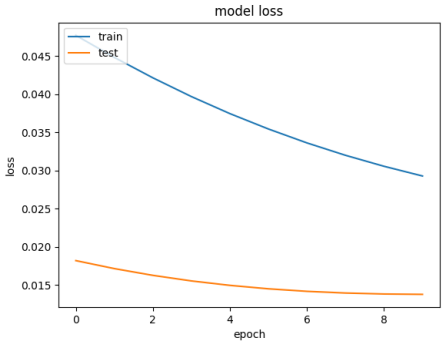


Figure 2.6: Table of results and model loss plot of SGD

ter on Electronic System) of Alma Mater Studiorum University of Bologna[13]. We used the same network architecture in all of the examples: 1 input layer consisting of 4 units (in this case the features correspond to the 4 angles of incidence that the impact forms with the 4 sensors), 1 hidden layer (with a number of neurons that can vary) and 1 output layer consisting of 2 units (the impact point coordinates). Other varying parameters are: the test set size, the batch size for training and the number of epochs. We reported the results related to some experiments in Figures 2.6, 2.7 and 2.8.

The Early Stop column refers to the stopping method. It can be the number of iterations before it reduced the error variation, or the fixed maximum number of iterations. As we can see the same activation function (Sigmoid) is used in all three examples.

The tables in Figures 2.6, 2.7 and 2.8 are listed in increasing order with respect to the loss. From these, the best methods are Adam and RMSProp. The latter reaches the minimum in a few iterations. SGD is fast but inaccurate, the loss is better than the RMSProp algorithm, despite that the

[13]The candidate has carried out a period of apprenticeship at CNAF-INFN of Bologna, where he had the opportunity to deepen the neural networks theory and their application in physics and engineering problems.

Optimizer: RMSProp Activation Function: Sigmoid (Logistic)

Test Size	Batch Size	Neurons	Epoch	Loss	Time [ms]	Early Stop
35	1	4	10	0.019360	1677.61	9
30	1	4	50	0.024174	2826.11	No
25	1	4	50	0.025436	1867.11	21
25	1	4	10	0.030676	995.13	No
25	25	4	50	0.031823	1402	No

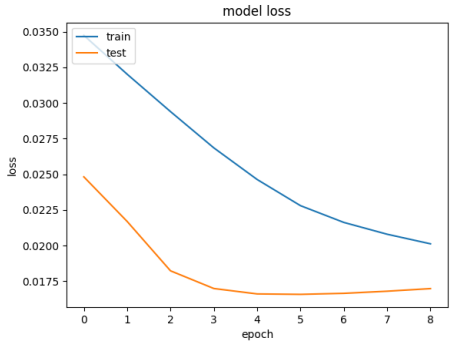


Figure 2.7: Table of results and model loss plot of RMSProp

Optimizer: Adam Activation Function: Sigmoid (Logistic)

Test Size	Batch Size	Neurons	Epoch	Loss	Time [ms]	Early Stop
35	1	4	50	0.019570	2801.97	38
30	1	4	50	0.020769	2334.97	No
30	15	4	50	0.022412	1626.56	No
30	1	4	10	0.027391	2953.65	No
25	1	4	50	0.027817	2067.44	20

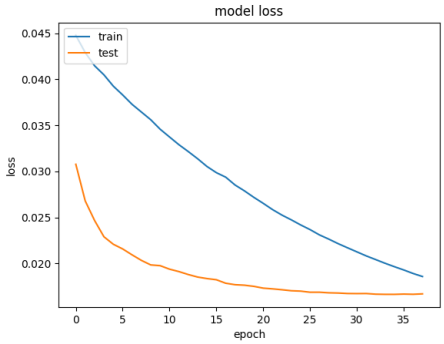


Figure 2.8: Table of results and model loss plot of Adam

parameters are equal.
We use Python and Keras (the TensorFlow API that is used for Deep Learning). All the numerical examples are run on a MacBook Pro with a 2.2 GHz Intel Core i7 processor and 16 Gb of memory.
In the next chapter we will introduce an algorithm that produces an approximation of a solution to a high-dimensional nonlinear PDE, by using BSDE theory and Deep Neural Networks.

Chapter 3

Deep Neural Network-Based BSDE Solver

In this chapter we provide an algorithm that solves high-dimensional nonlinear PDEs, by combining the BSDE stochastic theory (see Chapter 1) with the power of Machine Learning (see Chapter 2).
Partial Differential Equations are among the most popular tools used in modeling phenomena problems. The most important models are formulated as PDEs in high-dimensional spaces. For example, let us think of the Schrödinger equation in quantum many-body problems. In this case the dimensionality of the PDE is three times the number of electrons or quantum particles within the system. The practical use of these models can be very limited due to the "curse of dimensionality" problem. The computational cost of solving them grows exponentially with the dimensionality. Systems characterized by a high number of parameters are closely related to the real phenomenon but they are often impossible to solve. Sometimes, when we can find an approximation for the solution, the computational cost to obtain it is unbearable.
Another area where the curse of dimensionality plays a fundamental role is Machine Learning and Data Analysis. In Deep Neural Network applications the drawback is to balance the trade-off between the number of characteristics and the computational cost for training the network.

The algorithm was introduced by Weinan, Han and Jentzen [WHJ17]. This is formulated from the results of Chapter 1. We will transform the problem of solving a PDE into a BSDE (utilizing the nonlinear Feynman-Kac formula, Section 1.2). In this framework, the BSDE is solved by using a Deep Neural Network. The associated learning algorithm resembles the *reinforcement learning*, but built *ad hoc* specifically for this method.
Firstly, we describe the details of the algorithm. This can be used to solve a wide variety of problems that are based on high-dimensional nonlinear PDEs (*e.g.* the Schrödinger equation, the Hamilton-Jacobi-Bellman equation in dynamic programming and the Allen-Cahn equation)[1]. After that we concentrate on a practical financial problem, the *option pricing problem*. The goal is to determine a fair price for a derivative by using nonlinear Black-Scholes equation with default risk. We compare the results of this method with some classical algorithms based on Monte Carlo simulations. We will talk about the advantages of using Deep Neural Network and we will analyze possible improvements.

3.1 The Algorithm

The **Neural Network BSDE Solver** algorithm is used on a particular family of Partial Differential Equations, the semilinear parabolic PDEs. These PDEs can be represented as follows:

$$-\frac{\partial v}{\partial t}(t,x) - \mathcal{L}v(t,x) - f(t,x,v(t,x),\langle\sigma(t,x),D_x v(t,x)\rangle) = 0 \qquad (3.1)$$

where

$$\mathcal{L}v(t,x) = \big\langle b(t,x), D_x v(t,x)\big\rangle + \frac{1}{2}Tr(\sigma\sigma^T(t,x)D^2_{xx}v(t,x))$$

and with some specified terminal condition $v(T,x) = g(x)$.
In this case, t and x represent the time and the d-dimensional space variable. The term b is a known $\mathbb{R}^d$-valued function, σ is a known $\mathbb{R}^{d\times d}$-valued function

[1] [WHJ17]

and σ^T denotes the transpose associated to σ. We denote the gradient and the Hessian of the function v respect to x by $D_x v$ and $D^2_{xx} v$, respectively. $Tr(\cdot)$ denotes the trace operator of a matrix and f is a known nonlinear function. The algorithm allows us to find the solution v to the PDE (3.1) at $t = 0$, $x = \xi$, for some vector $\xi \in \mathbb{R}^d$.

3.1.1 BSDE Reformulation of the Problem

In Chapter 1 we presented the nonlinear Feynman-Kac formula (Section 1.2). This formula, under some conditions, permits us to represent the solution to a semilinear parabolic PDE as a Markov solution to the corresponding BSDE. We now briefly recall the main results of Chapter 1.
Let $(\Omega, \mathcal{F}, P)$ be a probability space and $W : [0,T] \times \Omega \to \mathbb{R}^d$ be a d-dimensional Brownian motion. $\mathbb{F} = (\mathcal{F}_t)_{0 \le t \le T}$ denotes the natural filtration on $(\Omega, \mathcal{F}, P)$ associated to $(W_t)_{0 \le t \le T}$. We consider the FBSDE[2]

$$\begin{cases} X_t = \xi + \int_0^t b(s, X_s) ds + \int_0^t \sigma(s, X_s) dW_s \\ Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s) ds - \int_t^T Z_s dW_s \end{cases} \tag{3.2}$$

We are looking for a $\mathbb{F}$-adapted solution process $\{(X_t, Y_t, Z_t)\}_{0 \le t \le T}$ with value in $\mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d$. Under suitable regularity assumptions on the coefficient functions b, σ and f, it can be proved the existence and up-to-indistinguishability uniqueness of the solution theorem (see Theorem 1.1.1). Moreover, by the nonlinear Feynman-Kac formula, the solution to the PDE (3.1) is related to the solution to the BSDE (3.2). Hence, for all $t \in [0,T]$ it holds P-a.s. that

$$Y_t = v(t, X_t) \qquad \textit{and} \qquad Z_t = \sigma^T(t, X_t) \nabla v(t, X_t), \tag{3.3}$$

Therefore, we can compute the value $v(0, X_0)$ associated to the PDE through Y_0 by solving BSDE. We plug the identities (3.3) in the second equation of

[2]We recall that FBSDE indicates a pair of differential stochastic equations. This pair consists of a *forward* and a *backward* SDE. The BSDE depends on the FSDE unknown variable. In this framework we talk about "decoupled" BSDE. The reverse is the "coupled" BSDE but it is not very common in applications, so we will not analyze it.

(3.2) and we obtain

$$\begin{aligned} Y_t = g(X_T) \ + \ \int_t^T f(s, X_s, v(s, X_s), \sigma^T(s, X_s)\nabla v(s, X_s))ds \\ + \int_t^T \sigma^T(s, X_s)\nabla v(s, X_s)dW_s \end{aligned} \tag{3.4}$$

In particular, for any $t_1, t_2 \in [0, T]$ with $t_1 \leq t_2$, it holds P-a.s. that

$$\begin{aligned} Y_{t_2} = Y_{t_1} \ - \ \int_{t_1}^{t_2} f(s, X_s, v(s, X_s), \sigma^T(s, X_s)\nabla v(s, X_s))ds \\ + \int_{t_1}^{t_2} \sigma^T(s, X_s)\nabla v(s, X_s)dW_s \end{aligned} \tag{3.5}$$

Next, we apply a time discretization to (3.5). More specifically, let $N \in \mathbb{N}$ and let $t_0, t_1, \cdots, t_N \in [0, T]$ be real numbers that satisfy

$$0 = t_0 < t_1 < \cdots < t_N = T.$$

For $N \in \mathbb{N}$ sufficiently large the Equation (3.5), combined with (3.3), provides an incremental law to compute the solution on the time interval nodes:

$$\begin{aligned} v(t_{n+1}, X_{t_{n+1}}) \approx v(t_n, X_{t_n}) - f(t_n, X_{t_n}, v(t_n, X_{t_n}), \sigma^T(t_n, X_{t_n})\nabla v(t_n, X_{t_n}))(t_{n+1} - t_n) \\ +\sigma^T(t_n, X_{t_n})\nabla v(t_n, X_{t_n})(W_{t_{n+1}} - W_{t_n}) \end{aligned} \tag{3.6}$$

From the first equation in (3.2) we obtain

$$X_{t_{n+1}} \approx X_{t_n} + b(t_n, X_{t_n})(t_{n+1} - t_n) + \sigma(t_n, X_{t_n})(W_{t_{n+1}} - W_{t_n}) \tag{3.7}$$

Each term in the r.h.s. of (3.6) and (3.7) approximations are known (the difference between two consecutive independent Brownian motions is normally distributed with mean 0 and variance $t_{n+1} - t_n$). The only unknown term is $\sigma^T \nabla v$. Next, we try to approximate the gradient of the solution by using Machine Learning techniques.

3.1.2 Deep Neural Network Approximation

For the sake of simplicity, we assume that the diffusion coefficient σ in (3.1) is the identity matrix, *i.e.* for all $x \in \mathbb{R}^d$ it holds that $\sigma(x) = Id_{\mathbb{R}^d}$.
From the Equation (3.6) we have to approximate

$$(\nabla_x v)(t_n, x) \in \mathbb{R}^d \tag{3.8}$$

with $x \in \mathbb{R}^d$, $n \in \{0, 1, \cdots, N\}$. This approximation takes place by using Feedforward Deep Neural Networks. We notice that in this framework Machine Learning is used to approximate the gradient of the solution function instead of the solution itself. The value v will be obtained by using the Equation (3.6).
In other words, we think of $\rho \in \mathbb{N}$ as the number of parameters in the Neural Network. Let $\theta \in \mathbb{R}^\rho$ be the vector of parameters and $\mathcal{V}^\theta$ be a suitable approximation of the solution at $t = 0$ and $x = \xi$,

$$\mathcal{V}^\theta \approx v(0, \xi)$$

for all appropriate parameters vector $\theta \in \mathbb{R}^\rho$. Let $\mathcal{G}_n^\theta : \mathbb{R}^d \to \mathbb{R}^d$ be a family of continuous functions, with $\theta \in \mathbb{R}^\rho$ and $n \in \{0, 1, \cdots, N-1\}$. For all appropriate values of $\theta \in \mathbb{R}^\rho$, $x \in \mathbb{R}^d$ and $n \in \{0, 1, \cdots, N-1\}$, the function $\mathcal{G}_n^\theta$ approximates $(\nabla_x v)(t_n, x)$. Hence,

$$\mathcal{G}_n^\theta \approx (\nabla_x v)(t_n, x).$$

Let $\mathcal{X} : \{0, 1, \cdots, N\} \times \Omega \to \mathbb{R}^d$ and $\mathcal{Y}^\theta : \{0, 1, \cdots, N\} \times \Omega \to \mathbb{R}$ be two stochastic processes. They satisfy, for all $\theta \in \mathbb{R}^\rho$,

$$\mathcal{Y}_0^\theta = \mathcal{V}^\theta, \quad \mathcal{X}_0 = \xi$$

and

$$\mathcal{X}_{n+1} = \Upsilon(t_n, t_{n+1}, \mathcal{X}_n, W_{t_{n+1}} - W_{t_n}),$$

$$\mathcal{Y}_{n+1}^\theta = \mathcal{Y}_n^\theta - f(t_n, \mathcal{X}_n, \mathcal{Y}_n^\theta, \mathcal{G}_n^\theta(\mathcal{X}_n))(t_{n+1} - t_n) + \mathcal{G}_n^\theta(\mathcal{X}_n)(W_{t_{n+1}} - W_{t_n})$$

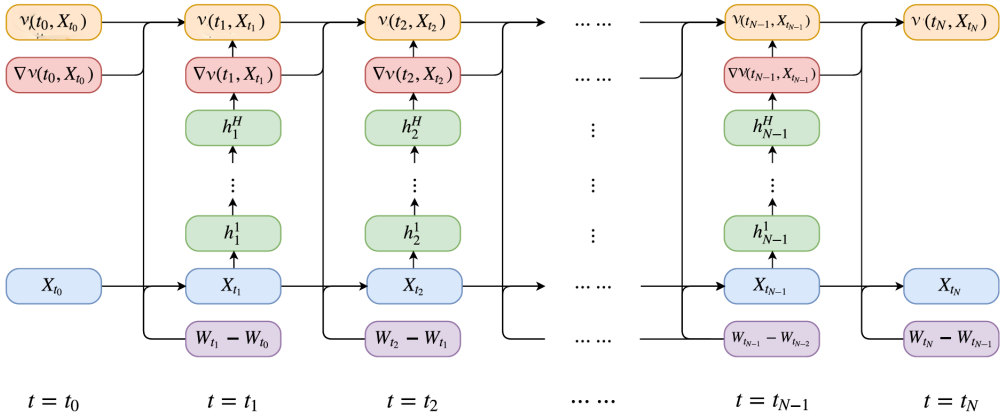


Figure 3.1: Neural Network architecture for the BSDE solver

where $\Upsilon : [0,T]^2 \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is a function that represents the incremental law in (3.7).

3.1.3 Neural Network Architecture

Figure 3.1 shows the Multilayer Neural Network for the BSDE solver algorithm. We notice that in Figure 3.1 the value $\nabla v(t_n, X_{t_n})$ is directly approximated by the network, while the solution $v(t_n, X_{t_n})$ is computed recursively.

The graph consists of a total of $N-1$ multilayer sub-network, one for each internal node of the time partition. Each sub-network has H hidden layers $h_n^1, \cdots, h_n^H$. Therefore, the whole network has $(H+2)(N-1)$ layers in total. Summing up, there are three types of connections in this network:

(i) $N-1$ Feedforward Multilayer Neural Networks $X_{t_n} \to h_n^1 \to h_n^2 \to \cdots \to h_n^N \to \nabla v(t_n, X_{t_n})$. They approximate the spatial gradient of the solution at $t = t_n$. The parameters of these networks are the weights of the synaptic connections θ_n, with $n \in \{1, \cdots, N-1\}$. These parameters are adjusted by using appropriate training algorithms (optimization algorithms).

(ii) The connection $(v(t_n, X_{t_n}), \nabla v(t_n, X_{t_n}), W_{t_{n+1}} - W_{t_n}) \to v(t_{n+1}, X_{t_{n+1}})$ represents the forward iteration. This is characterized by the Equation (3.6). It allows us to compute the final network output $v(t_N, X_{t_N})$. There are no parameters to be optimized in this network.

(iii) $(X_{t_n}, W_{t_{n+1}} - W_{t_n}) \to X_{t_{n+1}}$ represents the connection between different time blocks. This is characterized by (3.7). Also in this case there are no parameters to be optimized.

3.1.4 Neural Network Training

As we have seen in Chapter 2, the aim of the Neural Networks training is to adjust synaptic weights in order to minimize the loss function. Hence, the training problem is converted into an optimization problem. There are many algorithms to compute the optimum θ. The most used are based on the Stochastic Gradient Descent method (see Chapter 2, Section 2.3.3). We consider the Mean Square Error between the real and the approximate terminal condition of the BSDE (3.2). We use this MSE to define the expected loss function

$$\mathbb{R}^\rho \ni \theta \mapsto E[|\mathcal{Y}_N^\theta - g(\mathcal{X}_N)|^2] \in [0, \infty] \tag{3.9}$$

We assume that the function in (3.9) has a unique global minimum and let $\Lambda \in \mathbb{R}^\rho$ be the real vector for which the function in (3.9) is minimal. The minimizing loss function is inspired by the fact that

$$E[|Y_T - g(X_T)|^2] = 0$$

according to the BSDE (3.2).
The total set of Deep Neural Network parameters is $\theta = \{\theta_{v_0}, \theta_{\nabla v_0}, \theta_1, \cdots, \theta_{N-1}\}$. Weights $\{\theta_{v_0}, \theta_{\nabla v_0}\}$ characterize two additional Neural Networks that are used to approximate the functions $x \mapsto v(0, x) \in \mathbb{R}$ and $x \mapsto \nabla v(0, x) \in \mathbb{R}^d$ respectively. Therefore, under appropriate regularity conditions, the algorithm can estimate the vector $\Lambda \in \mathbb{R}^\rho$ by using Stochastic Gradient Descent methods. We denote the sequence of approximations of Λ by $\Theta : \mathbb{N}_0 \times \Omega \to$

$\mathbb{R}^\rho$. This is obtained by the following formula

$$\Theta_m = \Theta_{m-1} - \gamma \Phi^m(\Theta_{m-1})$$

where $m \in \mathbb{N}_0$. In this case Φ is the approximation of the loss function gradient in (3.9) and γ is the learning rate. For sufficiently large $\rho, N, m \in \mathbb{N}$ and sufficiently small $\gamma \in (0, \infty)$, we obtain an approximation of the PDE (3.1) at $t = 0$ and $x = \xi$,

$$\mathcal{V}^{\Theta_m} \approx v(0, \xi).$$

Of course, the above iterative formula to compute the sequence of Θ is generic and it is only esplicative of the method. In the next sections we will specify the various optimization methods that we are going to use (Adam, RMSProp). We will test the performances of all of them.
Therefore, the Deep Neural Network BSDE solver algorithm can be used to approximate the solution to the semilinear parabolic PDE that appears in many real problems, from quantum physics to optimal control theory. In the next section we are going to present a financial problem.
We will apply the above algorithm to solve the nonlinear PDE that control the option pricing in dimension one hundred. We use it in a financial mathematics context.

3.2 Black-Scholes Option Pricing Problem

For a better understanding of the application of the Deep Neural Network BSDE solver algorithm we have to introduce a digression about a classical financial mathematics problem.
The problem consists of assigning a fair price to a financial derivative (in our case an **option**). This issue becomes more complex when we transfer the model into the real market framework. Let us go step by step.
The Market Exchange [3] is an example of complex system. In the market, the apparently random price fluctuations are the result of the combination

[3]The Exchange or Bourse is a highly organized market where *brokers* and *traders* sold and bought tradable securities, commodities, foreign exchange and option contracts.

of different responses by speculators and traders. These responses are often highly correlated. The difficulty of forecasting traders' behavior led to develop stochastic models, in order to simulate the random nature of these behaviours. In this framework we can contextualize the financial-mathematical models to price options, derivatives and other financial instruments.
We start with the classical Black-Scholes equation [BS73] and we will conclude with an introduction to a more realistic model, by including the default risk [CGGN13].

3.2.1 European Call Option

A simple financial derivative is the **European Call Option** [4]. We suppose that at a certain time $t = 0$ an agent decides to make a contract with a seller (a bank, for instance). The call option contract gives the owner the right, but not the obligation, to buy an action at a specific price, the **strike price** K, on a certain expiration date $t = T$. If the share price at time $t = T$, $x(T)$, is larger than the strike price K, then the owner can exercise the option right and buy the share at the agreed price. The profit, by immediately selling the shares on the market, is $x(T) - K$. In contrast, if $x(T) < K$, then the option owner can decide not to buy the share. The profit in this case is equal to zero. Hence, the **payoff** of an European Call Option is:

$$(x(T) - K)^+ = \max\{x(T) - K, 0\}.$$

Symmetrically, an European Put Option gives the owner the right, but not the obligation, to sell a share at the strike price K, on the expiration date T. In this case the payoff is

$$(K - x(T))^+ = \max\{K - x(T), 0\}.$$

[4] "European" options are contracts that give the owner the right (but not the obligation) to buy or sell the underlying security at a specific price, only on the option's expiration date. They are different from the "American" counterpart, which gives the owner the right to buy or sell in any time between the purchase and the expiration date.

These type of contracts have their own price of activation. Assigning the fair price to these derivatives is called the **option pricing** problem. The option pricing depends on several variables[5].

- **The Price of the Underlying Asset:** when the underlying asset increases (resp. decreases), then the value of a call option increases (resp. decreases). Although, the value of a put option decreases (resp. increases).

- **The Strike Price:** This is the price of the underlying asset that agents agree when they make the contract. Usually, it is assigned by the seller and it can be lower, equal or higher than the current price of the asset.

- **The Volatility of the Underlying Asset:** The volatility is a statistical measure of the *dispersion* of returns for a given security or market index. An asset with high volatility is subject to frequent and strong oscillations of its price. Commonly, the higher the volatility, the riskier the asset and the more likely the profit.

- **The Expiration Date:** If the expiration date T is close to the purchase date, then the temporal value component tends to be zero. Hence, under the same assumptions, the farther the expiration date, the higher the option price.

Now, we report a real example of an European Call Option. For the rest of the thesis we will concentrate on these kind of derivatives.

Example 3.1. *An investor purchases a* 90*-day European Call Option on a stock of* 1000 *Google's shares (GOOG) with a strike price of* 125€.
To purchase an option contract the buyer has to pay the seller an option price of 0.05€ *for each share. The total amount of the option price is* 50€ $(1000 \times 0.05€ = 50€)$. *On the expiration date there may be two scenarios:*

- ***The Price of the Underlying Asset is Increased***
  At expiration, the spot price of the stock GOOG is 130€. *Therefore,*

[5] [B01]

the current price is higher than the strike price.
In this case, the owner of the call option has the right to purchase the stock at 125€ *and exercises the option, making* 5€ *(or* 130€ − 125€*) for each share. The trader's profit margin is*

$$1000 \times (130€ - 125€) - (1000 \times 0.05€) = 4950€$$

- ***The Price of the Underlying Asset is Decreased***
  In this scenario, if the spot price of the stock GOOG is 120€ *at expiration. It does not make sense to exercise the option to purchase the stock at* 125€*. In this case, the payoff is* 0€ *and the buyer claims a loss of* 50€*, the option price.*

Why Use Options?

Derivatives like put or call options are frequently used for two purposes:

- As a financial coverage for medium/high risk investments (*hedging with options*);
- For the speculation;

For example, we can think of hedging as an insurance policy, just as we insure our house or car. Options can be used to insure our investments against a downturn. There is no doubt that hedging strategies can be useful, especially for large institutions.
Options can be used also for speculation purposes. Let us think of the strategy of a put option. In this case, the seller earns only when the value of the underlying asset decreases. Put options are the easiest way to earn a profit after a financial crash.
In the next section we will intoduce the Black-Scholes model. Firstly we will describe the one-dimensional equation and classical solver methods. After that, we will introduce the multi-dimensional model and the Black-Scholes equation with the default risk. For the latter we are going to apply the Deep Neural Network-based BSDE solver to approximate the solution of the model.

3.2.2 Black-Scholes Model

The Black-Scholes-Merton model initially appeared in the works by Merton [M73] and by Black and Scholes [BS73]. This is the standard method to attribute a fair price for financial derivatives such as options. In the next paragraph we are going to describe the main properties and the principal applications of this model.

Model Assumptions

We assume several "idealistic hypothesis" on the market and on the shares:

- The rate of return on the riskless asset is constant and thus called the *risk-free interest rate*.
- The underlying stock price is a random walk with drift, more precisely, it is a Brownian motion, and its drift and volatility are constant.
- It is allowed the short sale of underlying asset.
- There is no arbitrage opportunity (*i.e.* there is no way to make riskless profit).
- The option transactions do not incur any fees or costs.
- It is possible to buy and sell any amount, even fractional, of the stock.

The above conditions are an interesting topic to study but they are often far from the reality.

Model Derivation

In this section we derive the Black-Scholes model from the previous hypothesis and other financial-mathematical conditions[6]. Let us consider a financial derivative, whose price is indicated by $f(S_t, t)$, where t is the temporal variable and S_t is the underlying price. From the previous assumptions,

[6][G13], [P11]

we observe that the price S_t is a Brownian motion which satisfies the following Stochastic Differential Equation:

$$dS = rSdt + \sigma SdW_t, \tag{3.10}$$

where $r \in \mathbb{R}$ is the interest rate of the stock and $\sigma \in \mathbb{R}_{>0}$ is its volatility. Let us formulate a portfolio[7]

$$\pi = f - \frac{\partial f}{\partial S}S.$$

We notice that $\partial f/\partial S$ is the derivative price variation with respect to the underlying price[8]. Equation (3.10) is the key of the Black-Scholes model. Let us apply the Itô's Lemma and obtain the Stochastic Differential Equation that the portfolio π has to satisfy.

$$d\pi = df - \frac{\partial f}{\partial S}dS = \left(\frac{\partial f}{\partial S}rS + \frac{\partial f}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2}\right)dt + \frac{\partial f}{\partial S}\sigma SdW_t - \frac{\partial f}{\partial S}rSdt - \frac{\partial f}{\partial S}\sigma SdW_t$$

We have assumed that the portfolio is riskless in an infinitesimal time interval. Under the hypothesis of no arbitrage, we have

$$d\pi = r\left(f - \frac{\partial f}{\partial S}\right)dt.$$

By merging the two previous equations, we obtain

$$rS\frac{\partial f}{\partial S} + \frac{\partial f}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} - rf = 0 \tag{3.11}$$

This is a parabolic PDE and is called the **Black-Scholes equation**. The above PDE has to be satisfied by any derivative instrument, with no arbitrage opportunity.

Black-Scholes Pricing for European Call Option

If we give the boundary conditions for the Black-Scholes PDE (3.11), then it is possible to obtain a unique solution, in order to determine the option

[7] A portfolio, is a collection of investments held by an investor subject

[8] Usually, in literature we can find this quantity indicated by the Greek letter Δ.

price. The quantitative finance consultant P. Wilmott *et al.* [WHD95] were the first to formulate these conditions for the European Call Options. In the following examples we only consider these kind of options.
The payoff for European Call Options, as we have already seen, is defined by

$$f(S,T) = \max\{S-K, 0\} = (S-K)^+.$$

We notice that, by the Equation (3.10), we can conclude that if $S = 0$ also $dS = 0$. Hence, in this particular case, the underlying price is constant. Therefore, if $S = 0$ on the expiration date, then the call option is worthless,

$$f(0,T) = 0.$$

If the underlying price increases with no limits, then the owner will use the option right. In this case the strike price can be neglected. Hence,

$$f(S,t) \sim S, \qquad for \;\; S \to \infty$$

Once we have set these boundary conditions, we can find a unique solution to the PDE (3.11).

$$f(S,t) = SN(d_1) - Ke^{-r(T-t)}N(d_2) \tag{3.12}$$

where S is the underlying asset price at t. In the previous equation, K is the strike price, r is the (annualized) risk-free interest rate and $N(\cdot)$ is the Standard Normal Cumulative Distribution function

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}y^2} dy.$$

By d_1 and d_2 we denote

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

$$d_2 = d_1 - \sigma\sqrt{T-t}.$$

Here, σ is the (annualized) volatility of returns of the underlying asset.

3.3 Numerical Methods

In this section we present the most popular algorithm to compute the approximate solution for the Black-Scholes method. In this framework the Monte Carlo estimator [9] is the key idea of the approximation process.

3.3.1 Principles of Monte Carlo

In mathematics, Monte Carlo methods are a broad class of computational algorithms that are based on the analogy between volume and probability. Their essential idea is to solve deterministic problems by using randomness. The theory of measure formalizes the intuitive notion of probability. If we consider an event as a collection of different states or configurations, then the probability of the event is its volume or measure. This is relative to a set of all possible outcomes. Monte Carlo methods use this observation in reverse. We compute the measure of a set by interpreting the volume as a probability. We will briefly analyze the formal idea behind the Monte Carlo method.
Let X be a random variable and $f \in m\mathcal{B}$ (*i.e.* f is a Borel-measurable function). We also assume that $f(X) \in L^2(\Omega, P)$. We want to compute the expectation $E[f(X)]$. By the law of large numbers,

$$E[f(X)] \approx \frac{1}{n} \sum_{k=1}^{n} f(X^{(k)}), \quad n \gg 1$$

where $X^{(1)}, X^{(2)}, \cdots, X^{(n)}$ are independent realizations of the random variable X. The variable X may assume different probability distribution (*e.g.* Uniform, Gaussian, Exponential, etc.). The more recent numerical computing software provide the main kind of distributions. Therefore, when we know the probability distribution of a random variable, the Monte Carlo method

[9] [G13]

permits to approximate it. In the next paragraph we evaluate the efficiency of Monte Carlo algorithms with reference to an option pricing problem.

3.3.2 Pricing Options Using Monte Carlo Simulations

As we have seen in Section 3.2, the underlying price evolution for a particular financial derivative follows the stochastic differential equation (3.10). The solution to this equation is

$$S(T) = S(0)\exp[(r - \frac{1}{2}\sigma^2)T + \sigma W(T)]. \tag{3.13}$$

In the above expression, $S(0)$ is the known underlying price at time $t = 0$, σ is its volatility and r is the risk-free interest rate. $W(T)$ is a random variable, which is normally distributed with mean 0 and variance T. $W(T)$ can be manipulated to become the distribution $\sqrt{T}Z$, where Z is a standard normal random variable with mean 0 and variance 1. Substituting this back into the Equation (3.13), we have

$$S(T) = S(0)\exp[(r - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T}Z]. \tag{3.14}$$

Thus, we can compute the expected value of the discounted payoff[10]

$$E[e^{-rT}(S(T) - K)^+].$$

By using Monte Carlo simulations it is possible to approximate this expected value. It can be proved that there is a correlation between the discounted payoff and the option price, for more datails we refer to [G13].

Example 3.2. *By using the Equation* (3.12) *it is possible to obtain the exact price for a European Call Option with* $S(0) = 90$, $K = 100$, $r = 0.05$, $\sigma = 0.2$ *and* $T = 1$. *Therefore, the result is* $f(S, 0) = 5.0912$.

In the numerical simulations we used two different approaches to price the option. We reported the results in the tables in Figure 3.2. In the first implementation we directly compute the price at $T = 1$ *(the expiration date). In*

[10]It refers to the discount factor e^{-rT} which consider the interest rate.

Monte Carlo method without discretization			
Number of simulation	**Price**	**99% confidence interval**	**Computation time [s]**
1000	4.8121	[4.0128, 5.6115]	0.0007
10000	5.0518	[4.7921, 5.3115]	0.0009
100000	5.0790	[4.9969, 5.1611]	0.0144
1000000	5.0811	[5.0550, 5.1072]	0.0838
Monte Carlo with discretization, $\Delta t = 0.01$			
Number of Simulation	**Price**	**99% Confidence Interval**	**Computation Time [s]**
1000	4.6645	[3.8904, 5.4386]	0.3779
10000	5.0952	[4.8381, 5.3523]	3.7425
100000	5.0751	[4.9920, 5.1581]	37.1450
1000000	5.0911	[5.0650, 5.1172]	374.3505

Figure 3.2: Option pricing by using Monte Carlo

the second framework we discretized the time interval into 100 *sub-intervals of the same length. In each node we used a Monte Carlo simulation. We conclude that the Monte Carlo method with discretization provides better results, despite the high computational cost. In the tables we reported the* 99% *confidence interval. This confirms the convergence of the Monte Carlo method. By the law of large numbers, increasing the number of iterations, the confidence interval is going to reduce.*

Multi-Dimensional Case

Now, we want to apply the Monte Carlo algorithm to price an option with many underlying assets. We consider d assets, each of them with value $S_i(t),\ \ i = 1, \cdots, d$. The equation which describes the evolution of a single stock price is

$$dS_i(t) = S_i(t) r_i(S(t), t)dt + S_i(t)\sigma_i(S(t), t)^T dW(t) \tag{3.15}$$

W is a k-dimensional Brownian motion, each of σ_i is a $\mathbb{R}^k$-valued vector and r_i is a real-valued function. We assume that both σ_i and r_i are deterministic functions and they depend on $S(t) = (S_1(t), S_2(t), \cdots, S_d(t))^T$. Itô calculus gives us the solution to the Equation (3.15),

$$S_i(T) = S_i(0) \exp[(r - \frac{1}{2}\sigma_i^2)T + \sigma_i W_i(T)] \tag{3.16}$$

$W_i(T)$ is a k-dimensional Brownian motion for any $i = 1, \cdots, d$, with mean 0 and variance T. We ideally would use, as well as in the one-dimensional case, a standard Random Number Generation software to generate samples of each $W_i(T)$. However, there is a problem. In the multi-dimensional case different assets do not behave independently. On average, they tend to move up and down together. This is modeled by introducing the correlation between different Brownian motions. Hence,

$$E[W_i(T)W_j(T)] = \Omega_{i,j}T$$

where $\Omega_{i,j}$ is the correlation coefficient. Let us see how to create correlated normal random variables. Let x be a vector of independent $N(0,1)$ variables and we define a new vector $y = Lx$. Each element of y is normally distributed, with mean $E[y] = LE[x] = 0$ and variance

$$E[yy^T] = E[Lxx^TL^T] = LE[xx^T]L^T = LL^T.$$

Thus, in order to obtain $E[yy^T] = \Omega$, we need to know L, such that

$$LL^T = \Omega.$$

L is not uniquely defined, we could use the Cholesky decomposition of Ω to find it. Therefore, from a symmetric and positive-definite matrix (usually, these conditions are satisfied by a generic correlation matrix) we can obtain a lower-triangular matrix with a positive diagonal. Once we find L we can construct a vector of normally distributed correlated variables. Hence, we simulate the underlying Brownian motions and compute the expected value of the discounted payoff.
The main issue of this method is that, almost always, we do not know the correlation matrix of the assets. A way to solve this problem is to compute the empirical covariance matrix of underlying assets and then compute the associated correlation matrix. It is a time-consuming algorithm and the complexity grows exponentially with the number of underlying assets. Moreover, collecting and memorizing a massive number of data is quite difficult.

Monte Carlo method on an option with 10 underlying assets

Number of Simulation	Price	99% Confidence Interval	Computation Time [s]
1000	5.7451	[1.1120, 10.3781]	0.2670
10000	5.0669	[3.6317, 6.5022]	2.5625
100000	4.9806	[4.5218, 5.4393]	26.3427
1000000	4.8644	[4.7198, 5.0090]	259.4046

Figure 3.3: Multi-dimensional option pricing using Monte Carlo

In the next example we use a correlation matrix which is randomly generated[11]. This example is useful to explain the multi-dimensional Monte Carlo algorithm but it can not be considered as a real scenario.

Example 3.3. *We consider a European Call Option with* 10 *underlying assets. We assume that*

$$S(0) = (S_1(0), S_2(0), \cdots, S_{10}(0)), \;\; S_i(0) = 100, i = 1, \cdots, 10$$

and $\sigma = (0.4, 0.4, \cdots, 0.4)$. *The strike price is* $K = 150$ *and the interest rate is* $r = 0.05$. *The results are reported in the table in Figure 3.3. In this case, we do not know the exact option price. However, the law of large numbers ensures the convergence of the method. Of course, the computation time is much higher than in the one-dimensional case.*

One of the main drawbacks of this algorithm is the necessity to provide a correlation matrix *a priori*. Also the estimation of an empirical correlation matrix may be difficult with a large number of assets.

3.3.3 Pricing Options Using Deep Learning

Previously, the conditions imposed on the market and the nature of derivatives allowed us to obtain a solution to the Black-Scholes linear PDE. This model can be modified in order to simulate a more realistic evolution of the option price. The Black-Scholes model can be augmented for real market fundamental factors. These include defaultable securities, higher interest

[11]For this purpose we use NumPy and Pandas that are two fundamental packages for scientific computing with Python 3.

rates for borrowing and lending, transaction cost, etc. Each of these extensions return a nonlinear contribution in the final pricing model. Despite the difficulties in approximating a solution to this kind of PDE, these nonlinear integrations can be often indispensable in modeling real phenomena. In particular, the credit crisis and the European sovereign debt crisis have highlighted the basic risk that has been neglected in the classical Black-Scholes model, the *default risk*. Moreover, there is the "curse of dimensionality" problem. This is typical of the financial derivatives with many underlying assets. Therefore, there is no possibility to use standard numerical algorithms. Monte Carlo simulations are unworkable due to the nonlinearity of the model. The other numerical methods like FEM or Galerkin are impossible to use because of the high dimensionality.
In order to overcome these drawbacks we will use the Deep Learning-based technique that we have introduced in Section 3.1. In this section we are going to apply this new kind of method to a practical problem. We would like to price a European Call Option based on 100 underlying assets, conditioned to the default risk.
When default of the contract's issuer occurs, the contract's holder only receives a fraction of the current value $\delta \in [0, 1)$. In this case the (possible) default is modeled by the first jump time of a Poisson process with intensity Q. This is a decreasing function of the current value. In other words, the default becomes more likely when the option value is low. Hence, the value process can be modeled by (3.1) with the generator

$$f(t, x, v(t, x), \langle \sigma(t, x), D_x v(t, x) \rangle) = -(1 - \delta)Q(v(t, x))v(t, x) - rv(t, x)$$

where r is the risk-free interest rate of the assets. We assume that the underlying asset price moves as a geometric Brownian motion. We then select the intesity function Q as a piecewise-linear function of the current value within three different intervals ($w^h < w^l, \gamma^h > \gamma^l$):

$$Q(y) = \mathbf{1}_{(-\infty, w^h)}(y)\gamma^h + \mathbf{1}_{[w^l, \infty)}(y)\gamma^l + \mathbf{1}_{[w^h, w^l)}(y)[\frac{(\gamma^h - \gamma^l)}{(w^h - w^l)}(y - w^h) + \gamma^h]$$

The nonlinear Black-Scholes equation in $[0,T]\times\mathbb{R}^{100}$ becomes

$$\frac{\partial v}{\partial t}+\bar{b}x\cdot\nabla v(t,x)+\frac{\bar{\sigma}^2}{2}\sum_{i=1}^{d}|x_i|^2\frac{\partial^2 v}{\partial x_i^2}(t,x)$$
$$-(1-\delta-r)\min\{\gamma^h,\max\{\gamma^l,\frac{(\gamma^h-\gamma^l)}{(w^h-w^l)}(v(t,x)-w^h)+\gamma^h\}\}v(t,x)=0 \tag{3.17}$$

We choose $T=1,\ \delta=2/3,\ r=0.02,\ \bar{b}=0.02,\ \bar{\sigma}=0.2,\ w^h=50,\ w^l=70,\ \gamma^h=0.2,\ \gamma^l=0.02$ and terminal condition $g(x)=\min\{x_1,\cdots,x_{100}\}$ for $x=(x_1,\cdots,x_{100})\in\mathbb{R}^{100}$.

Regarding the neural network architecture we choose $H=4$. Hence, the number of hidden layers of each sub-network (*i.e.* the "vertical" neural networks in 3.1) is equal to 4. We notice that the first and the last layer of these sub-networks have the number of neurons equal to the dimensionality of the problem (100). Meanwhile, the second and the third layer have the number of units equal to 110 ($dim+10$). In this case, the learning rate is heuristically set to 0.008. The batch size is 64 and the time interval of one year is discretized into 40 equal sub-intervals.

In this framework the exact solution to the semilinear parabolic PDE is not known (during the training, the error function is minimized with respect to the known terminal condition). By using the Multilevel Picard Approximation method to estimate the solution to (3.17) at $t=0$ and $x=(100,\cdots,100)$, we obtain

$$v(t=0,x=(100,100,\cdots,100))\approx 57.300.$$

We use this approximation to compare the performances. The Multilevel Picard algorithm uses standard estimation processes, like Monte Carlo simulations (on different accuracy levels[12]) and the Picard iterations algorithm. The main difference between the Picard method and the other deterministic methods (FEM, Galerkin, Finite Differences, etc.) is the computational cost.

[12]The *multilevels*.

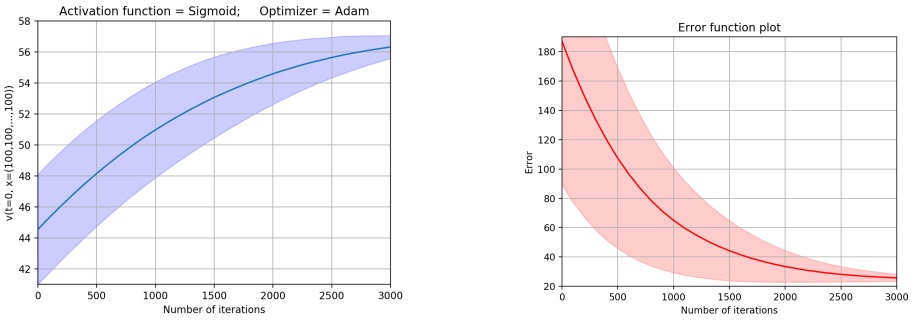


Figure 3.4: Activation Function: Sigmoid; Optimizer: Adam; Learning Rate = 0.008; $\Delta t = 0.03$; Maximum Number of Iterations = 3000; Batch Size = 64; $\bar{v}(0, (100, \cdots, 100)) \approx 56.3244$.

In the deterministic case the cost grows exponentially with the dimensionality. Instead, in the Picard method the growth is polynomial. For more details we refer to [HJK17].

Figure 3.4 shows the performances of the Deep Neural algorithm. The first is the plot of the mean value and the standard deviation of the price approximation for 5 independent runs. The second one is the error function plot with respect to the same runs. In this first example we used the Sigmoid as the activation function and the Adam algorithm to optimize the network parameters. The results are good and the average of the computational time is equal to 362.6 seconds. We compare this method with the RMSProp optimizer.

Figure 3.5 shows the results with reference to the RMSProp method. The average of the computational time is 346.8 seconds. Let us notice that, over 2000 iterations, the results do not vary and the approximation is close to the mean. The performances seem to be better than Adam.

We would like to reduce the computational cost. There are several ways to do it. For example, we can use less iterations or less units per layer. In this case the results may not be better. We try to reduce the number of time nodes (hence, we also reduce the number of layers). Figure 3.6 shows the results with reference to 5 independent runs on 10 time nodes. In this case the computational time is 84.4 seconds. The random component seems to be reduced. The standard deviation of the approximation and the error function

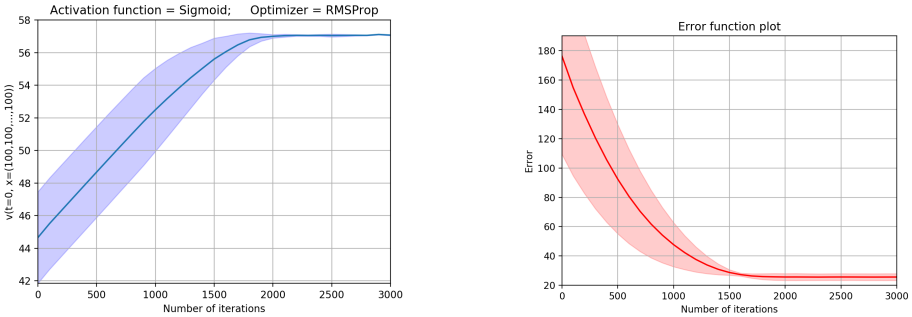


Figure 3.5: Activation Function: Sigmoid; Optimizer: RMSProp; Learning Rate = 0.008; $\Delta t = 0.03$; Maximum Number of Iterations= 3000; Batch Size = 64; $\bar{v}(0, (100, \cdots, 100)) \approx 57.0745$.

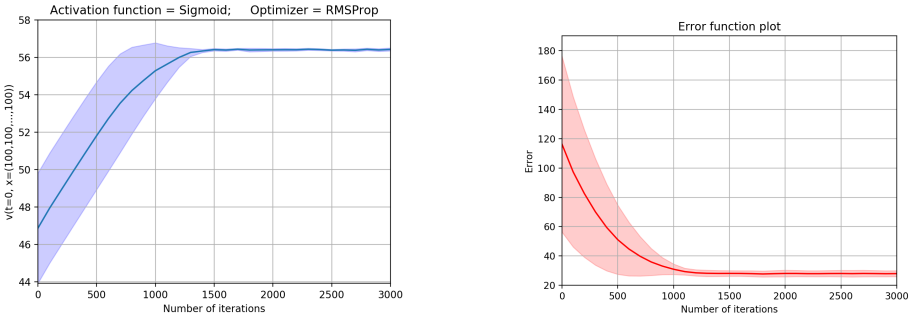


Figure 3.6: Activation Function: Sigmoid; Optimizer: RMSProp; Learning Rate = 0.008; $\Delta t = 0.1$; Maximum Number of Iterations = 3000; Batch Size = 64; $\bar{v}(0, (100, \cdots, 100)) \approx 56.4303$.

is minimized after 1000 iterations. The accuracy of the solution is reduced too. With 40 sub-networks the average approximation is $\bar{v}_{40} \approx 57.0745$. While, with 10 sub-networks is $\bar{v}_{10} \approx 56.4303$.
We notice that, without the default risk the option price is $v(t = 0, x = (100, 100, \cdots, 100)) \approx 60.781$. In this case, the Black-Scholes model is linear and it can be solved with a Monte Carlo method, as we have seen in Section 3.3.2. However, if we do not consider the default risk, the error could lead to serious consequences.
For the above experiments we used the programming-language Python, by using TensorFlow[13], the open-source software library for Deep Learning programming. All the numerical examples are run on a MacBook Pro with a 2.2 GHz Intel Core i7 processor and 16 Gb of memory.

[13]We used the implementation proposed by [WHJ17].

Conclusions

Partial Differential Equations (PDEs) are the most important tool used in modeling a large number of practical problems. From physics to financial mathematics, the evolution models are based on PDEs in high-dimensional spaces. However, solving these kind of PDEs is difficult due to the "curse of dimensionality" problem. For this reason, many deterministic algorithm (Finite Element Method or Finite Difference Method) are unfeasible. Moreover, if we apply the PDE-based model to a real phenomenon, then we must consider some nonlinear factors (in the thesis we saw the default risk for Black-Scholes equation). Therefore, we can not use probabilistic algorithms, like Monte Carlo simulation, due to the nonlinearity of the problem.
In order to overcome these issues, we have presented a new algorithm, introduced by [HJ17] and [WHJ17], that use Deep Learning techniques to solve these kind of problems. Numerical results suggest that the proposed algorithm is quite effective for a variety of real problems, especially in terms of accuracy. We can approximate the solution to a high-dimensional nonlinear PDE without knowing the correlation matrix.
However, there is some restrictions for this new algorithm due to the execution time and the computational cost. The most obvious drawback is that the number of parameters involved in the deep neural network grows with the number of points N, used to discretized time. This leads to an high computational cost. There are some improvements that can be implemented, for example we can use a different Neural Network architecture. Recurrent Neural Networks, attempting to imitate the brain's long and short-term memory, work well if we investigate the phenomena that have a temporal evolution.
Furthermore, the algorithm mentioned above can be extended to second order

nonlinear PDEs [BJ17]. In this case we use the Deep Learning to approximate the Hessian of the solution.
In recent years, Machine Learning techniques have had a big improvement both in theoretical and empirical aspects. By using these kind of algorithms we have the advantage of knowing that this topic is constantly growing.

Appendix A

Useful Results

Theorem A.0.1 (**Martingale Representation Theorem**).
Let W_t be a Brownian motion on a filtered probability sapce $(\Omega, \mathcal{F}, \mathbb{F}, P)$ and $\mathbb{F}$ be the natural filtration associated to W. Then, every square integrable martingale $(M_t)_{0\leq t\leq T}$ can be written in the form

$$M_t = M_0 + \int_0^t Z_s dW_s$$

with $(Z_t) \in \mathbb{H}^2(0,T)$ predictable process.

This theorem can be naturally extended to the case that W is a vectorial Wiener process.

Theorem A.0.2 (**Doob's Martingale Inequality**).
Let $M = (M_t)_{0\leq t\leq T}$ be a submartingale taking non-negative real values, either in continuous or discrete time. Then, for any constant $\lambda > 0$ and for all $p > 1$,

$$P\left[\sup_{0\leq t\leq T} |M_t| \geq \lambda\right] \leq \frac{E[|M_T|]}{\lambda}$$

$$E\left[\sup_{0\leq t\leq T} |M_t|^p\right] \leq \left(\frac{p}{p-1}\right)^p E[|M_T|^p].$$

Theorem A.0.3 (**Burkholder-Davis-Gundy Inequality**).
For all $p > 0$, there exist two positive constants c_p and C_p such that, for each local continuous martingale $M = (M_t)_{0 \le t \le T}$, it holds:

$$c_p E[\langle M \rangle_T^{p/2}] \le E\left[\sup_{0 \le t \le T} |M_t|\right]^p \le C_p E[\langle M \rangle_T^{p/2}].$$

Appendix B

Generalization Theory

In the last few years a new modern theory has been developed about Deep Learning. This tries to explain not "how" a Machine Learning algorithm works but "why". This is an interesting question also because these kind of models are similar to black-boxes and their capacity is often compared to an alchemical result. The **Generalization Theory** that explains why Deep Learning generalizes so well. In this section we will discuss most recent theoretical and empirical advances in this particular field.
The first topic of study was dealt within the context of Generalization Theory and it concerns the theoretical base for the problem of generalization accuracy. Why if we improve accuracy during the training, we have better performance in the testing?[1] A quantity that measures the difference between the training accuracy and the test accuracy is the **Generalization Error** or "Generalization Gap". More rigorously, Generalization Gap can be defined as

$$\mathcal{E}_{Gen} := \mathcal{R}[f_{\mathcal{A}(\mathcal{S}_p)}] - \hat{\mathcal{R}}_p[f_{\mathcal{A}(\mathcal{S}_p)}]$$

where $\mathcal{R}$ is the non-computable expected risk. This is the expectation of the loss function. Whereas $\hat{\mathcal{R}}$ is the computable empirical risk. Both risks refer to a function f on a dataset $\mathcal{S}_p$ given a learning algorith $\mathcal{A}$. Essentially, if we bound the Generalization Error with a small value it would guarantee that

[1] We recall that in a classification problem, the accuracy is a metric for evaluating the model. Informally, it is the percentage of data correctly classified on the total number of elements.

Deep Learning algorithm f generalizes well in practice. Multiple theoretical bounds exist for the Generalization Gap and they are based on model complexity, robustness and stability. For a classical approach we refer to the paper about Statistical Learning Theory [V13]. Informally, Statistical Learning Theory is a field of Machine Learning that uses statistical arguments to improve the automatic learning performances.
More recently a method that can analyzes the theoretical bound of the Generalization Gap has been developed, by using the optimization algorithms that we saw in Section 2.3.3. Previously, we have seen some variations of the Stochastic Gradient Descent (SGD) method. Now we analyze its generalization capacity.
In a recent paper [KL17] it has been proved that SGD method is an on-average stable[2] algorithm under some additional conditions on the loss function. These conditions are fulfilled in commonly used loss functions in Neural Networks, like Sigmoid or Hyperbolic Tangent . It was proved that the following inequality is true for non-convex functions as in Deep Neural Networks.

$$\begin{aligned} &E[\mathcal{R}[f_{\mathcal{A}(\mathcal{S}_p)}] - \hat{\mathcal{R}}_p[f_{\mathcal{A}(\mathcal{S}_p)}]] \\ &\leq \mathcal{O}\left(\frac{1+\frac{1}{c\xi}}{p} \cdot \max\left\{\left(E[\hat{\mathcal{R}}_p[f_{\mathcal{A}(\mathcal{S}_p)}]] \cdot N\right)^{\frac{c\xi}{1+c\xi}}, (\frac{N}{p})^{(c\xi)}\right\}\right). \end{aligned} \tag{B.1}$$

Where p is the training set size, N is the number of iterations and ξ characterizes how the curvature at the initialization point of the SGD method affects the stability. If ξ is small, then the algorithm stability is great. Thus, the SGD results are less affected by small perturbations in the training set. Therefore, we can reach a faster generalization. Moreover, the above inequality shows that the greater the training set size ($p \gg 1$), the smaller the generalization gap.
As we can see by (B.1), another important parameter is the batch size m. Informally, a small batch training introduces noise to the gradient and this noise drives the SGD away from sharp minima. Thus, enhancing general-

[2]Stability, in this case, means how sensitive is SGD to small perturbations in the training set.

ization. It was proved[3] that the optimum batch size is proportional to the learning rate and the training set size. Instead of decaying the learning rate, we can obtain the same results by increasing the batch size during the training. This procedure is successful for SGD, Momentum and Adam. In the SGD methods with momentum it is possible to express these observation by

$$\frac{\gamma p}{m(1-\alpha)} = constant$$

where γ is the learning rate, α is the momentum, p is the training set size and m is the batch size.
These results can confirm that Deep Learning is far from being called "Alchemy". The theoretical study that we have shown has to support the software development and empirical experience, in order to achieve better results with the best performances.

[3][SL18]

Bibliography

[B73] Bismut, J. M. (1973). Conjugate convex functions in optimal stochastic control. Journal of Mathematical Analysis and Applications, 44(2), 384-404.

[B01] Borsa Italiana (2001). Guida alle opzioni, aspetti teorici. Derivati Azionari, IDEM Mercato Italiano dei derivati.

[B06] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer-Verlag New York.

[BET04] Bouchard, B., Ekeland, I., Touzi, N. (2004). On the Malliavin approach to Monte Carlo approximation of conditional expectations. Finance and Stochastics, 8(1), 45-71.

[BJ17] Beck, C., Jentzen, A. (2017). Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. arXiv preprint arXiv:1709.05963.

[BS73] Black, F., Scholes, M. (1973). The pricing of options and corporate liabilities. Journal of political economy, 81(3), 637-654.

[BT95] Bertsekas, D. P., Tsitsiklis, J. N. (1995, December). Neuro-dynamic programming: an overview. In Proceedings of the 34th IEEE Conference on Decision and Control (Vol. 1, pp. 560-564). Piscataway, NJ: IEEE Publ.

[CGGN13] Crépey, S., Gerboud, R., Grbac, Z., Ngor, N. (2013). Counterparty risk and funding: The four wings of the TVA. International Journal of Theoretical and Applied Finance, 16(02), 1350006.

[G97] Guyon, I. (1997). A scaling law for the validation-set training-set size ratio. AT&T Bell Laboratories, 1-11.

[G13] Glasserman, P. (2013). Monte Carlo methods in financial engineering (Vol. 53). Springer Science & Business Media.

[GBC16] Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep learning. Cambridge: MIT press.

[GMSV98] Guyon, I., Makhoul, J., Schwartz, R., Vapnik, V. (1998). What size test set gives good error rate estimates?. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(1), 52-64.

[GS03] Grippo, L., Sciandrone, M. (2003). Metodi di ottimizzazione per le reti neurali. Rapporto Tecnico, 09-03.

[H09] Haykin, S.S.(2009). Neural networks and learning machines (Vol. 3). Upper Saddle River, NJ, USA:: Pearson.

[HJ17] Han, J., Jentzen, A. (2017). Solving high-dimensional partial differential equations using deep learning. arXiv preprint arXiv:1707.02568.

[KB14] Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[KL17] Kuzborskij, I., Lampert, C. H. (2017). Data-dependent stability of stochastic gradient descent. arXiv preprint arXiv:1703.01678.

[HJK17] Hutzenthaler, M., Jentzen, A., Kruse, T. (2017). On multilevel Picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations. arXiv preprint arXiv:1708.03223.

[LGW06] Lemor, J. P., Gobet, E., Warin, X. (2006). Rate of convergence of an empirical regression method for solving generalized backward stochastic differential equations. Bernoulli, 12(5), 889-916.

[LS01] Longstaff, F. A., Schwartz, E. S. (2001). Valuing American options by simulation: a simple least-squares approach. The review of financial studies, 14(1), 113-147.

[M73] Merton, R. C. (1973). Theory of rational option pricing. The Bell Journal of economics and management science, 141-183.

[M97] Mitchell, T. M. (1997). Machine learning (mcgraw-hill international editions computer science series).

[MMY99] Ma, J., Morel, J. M., Yong, J. (1999). Forward-backward stochastic differential equations and their applications (No. 1702). Springer Science & Business Media.

[MP70] Minsky, M., Papert, S. A. (1970). Perceptrons: An introduction to computational geometry. MIT press.

[MP99] Magoulas, G. D., Prentza, A. (1999). Machine learning in medical applications. In Advanced Course on Artificial Intelligence (pp. 300-307). Springer, Berlin, Heidelberg.

[P98] Pardoux, É. (1998). Backward stochastic differential equations and viscosity solutions of systems of semilinear parabolic and elliptic PDEs of second order. In Stochastic Analysis and Related Topics VI (pp. 79-127). Birkhäuser, Boston, MA.

[P99] Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. Acta numerica, 8, 143-195.

[P09] Pham, H. (2009). Continuous-time stochastic control and optimization with financial applications (Vol. 61). Springer Science & Business Media.

[Pe09] Perkowski, N. (2009). Markovian Case: FBSDEs and their Connection to PDEs.

[P11] Pascucci, A. (2011). PDE and martingale methods in option pricing. Springer Science & Business Media.

[P15] Pham, H. (2015). Feynman-Kac representation of fully nonlinear PDEs and applications. Acta Mathematica Vietnamica, 40(2), 255-269.

[PP90] Pardoux, E., Peng, S. (1990). Adapted solution of a backward stochastic differential equation. Systems & Control Letters, 14(1), 55-61.

[PP92] Pardoux, E., Peng, S. (1992). Backward stochastic differential equations and quasilinear parabolic partial differential equations. In Stochastic partial differential equations and their applications (pp. 200-217). Springer, Berlin, Heidelberg.

[PPP04] Pagès, G., Pham, H., Printems, J. (2004). Optimal quantization methods and applications to numerical problems in finance. In Handbook of computational and numerical methods in finance (pp. 253-297). Birkhäuser, Boston, MA.

[PR14] Pardoux, E., Râșcanu, A. (2014). Stochastic differential equations, Backward SDEs, Partial differential equations (Vol. 69). New York: Springer.

[R13] Rojas, R. (2013). Neural networks: a systematic introduction. Springer Science & Business Media.

[R16] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

[RHW86] Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533.

[SB98] Sutton, R. S., Barto, A. G. (1998). Reinforcement learning: An introduction. MIT press.

[SL18] Smith, S. L., Le, Q. V. (2018). A bayesian perspective on generalization and stochastic gradient descent.

[SV08] Smola, A., Vishwanathan, S. V. N. (2008). Introduction to machine learning. Cambridge University, UK, 32, 34.

[TH12] Tieleman, T., Hinton, G. (2012). Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26-31.

[V13] Vapnik, V. (2013). The nature of statistical learning theory. Springer science & business media.

[WHD95] Wilmott, P., Howison, S., Dewynne, J. (1995). The mathematics of financial derivatives: a student introduction. Cambridge university press.

[WHJ17] Weinan, E., Han, J., Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Communications in Mathematics and Statistics, 5(4), 349-380.

[Z17] Zhang, J. (2017). Backward Stochastic Differential Equations: From Linear to Fully Nonlinear Theory (Vol. 86). Springer.