1 2 9 0

UNIVERSIDADE Đ
COIMBRA

Andrei Fokin Teixeira

Fast Option Price Using Differential Machine Learning

Dissertation in the context of the Master in Engineering and Data Science, advised by Professor Nuno Lourenço and by Julien Dinh and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September, 2024

1 2 9 0

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Ð
COIMBRA

Andrei Fokin Teixeira

Fast Option Price Using Differential Machine Learning

Dissertation in the context of the Master in Engineering and Data Science, advised by Professor Nuno Lourenço and by Julien Dinh and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September, 2024

Acknowledgements

This work would not have been possible without the contribution of so many dear people around the globe.

I thank those who are in Brazil and have contributed, on an intellectual level, to my knowledge of Economics and Finance and, on a personal level, those who helped me and believed that I could take the step of studying, working and living outside my home country, especially my friends in Brazil, my former husband Vinicius and my immediate family (mother Katia, brother Luan, sister-in-law Kamila and future cousin Kauan, who I discovered in May will come here as soon as I present this thesis).

I also thank the professors, staff and fellow students who made me move forward in a universe that was not new for me, but that I had not fully explored, namely Data Science. I've always dreamed of working in this area, so when I see myself doing a master's degree at Portugal's oldest university, I realize what an immense privilege it is and that it will leave a deep mark on my life. The Master's course in Data Science is very recent and I am part of the third group, the last without a degree within the same department. I feel very fortunate to have had classes with professors who helped found not only MECD, but MEI and DEI itself.

Special thanks to Professors Ernesto Costa, Jorge Henriques, Catarina Silva, César Teixeira and Pedro Miguel Neves, who were always very open, patient and had a very horizontal dialog with someone living in a new country and studying a new area of knowledge. To professor and coordinator Paulo de Carvalho for being a backup for big problems and doubts, and to Márcia for her love and attention to the immediate day-to-day difficulties.

I want to thank my advisors Nuno Lourenço at the university and Manuel Coimbra at the bank, who helped me with constant feedback and gave me confidence that I was on the right track. I believe that this thesis has the potential to open doors to change my life in a way that I wouldn't have imagined 5 years ago. And thank you to my fellow colleagues with whom we exchanged doubts, anxieties and support for each other.

Abstract

Financial markets, as other economic activities, have demanded increased efficiency in its results and has turned to Data Science for this purpose. This dissertation explores the potential of Differential Machine Learning techniques to enhance the speed and efficiency of option pricing in financial markets, the potential of which has already been proved in the literature. Facing many analytical equations as theoretical models to run Monte Carlo simulations to generate future expected stock prices and many option instruments in the market to generate expected payoffs, differential learning optimizes financial institutions' predictions and helps improve their operational results. This document reflects a systematic progression from the initial definition of the problem and the research objectives, recalling the history of the main financial models and instruments, the introduction of computing in predicting financial assets and the emergence of alternative and fast methods for carrying out the pricing option. Based on the inaugural paper by Huge & Savine (2020), the practical content of this thesis proposes, initially, the translation of their code written in TensorFlow to PyTorch, a more flexible, well-documented language with a rapidly growing community. Then, we explore different architectural components of the neural network to see if each change is able to generate even greater improvements in differential learning. Three architectures, compared to Huge & Savine (2020) one, presented better RMSE values and higher percentage performance of differential learning compared to the standard. Also it was observed how sensitive some of the architectural components of a neural network are to the point where small changes made to the code in the first paper resulted in very high RMSE values. Overall, this dissertation aims to contribute to the advancement of option pricing studies by presenting a vast history of the development of the problem studied and by creating new benchmarks in terms of architecture for PyTorch code language and metrics of evaluation.

Keywords

Option Pricing - Financial Market - Differential Machine Learning

Resumo

O mercado financeiro, à semelhança de outras atividades económicas, tem demandado uma maior eficiência nos seus resultados e tem recorrido à Ciência dos Dados para esse efeito. Esta dissertação explora o potencial das técnicas de Differential Machine Learning para aumentar a rapidez e a eficiência do cálculo de preços de opções nos mercados financeiros, cujo potencial já foi comprovado na literatura. Perante muitas equações analíticas a partir modelos teóricos para executar simulações de Monte Carlo para gerar preços futuros esperados de ações e muitos instrumentos de opções para gerar payoffs esperados das mesmas, a aprendizagem diferencial otimiza as previsões das instituições financeiras e ajuda a melhorar os seus resultados operacionais. Este documento reflete uma progressão sistemática a partir da definição inicial do problema e dos objetivos de investigação, recordando a história dos principais modelos e instrumentos financeiros, a introdução da informática na previsão de ativos financeiros e o aparecimento de métodos alternativos e rápidos para realizar a precificação de opções. Com base no artigo inaugural de Huge & Savine (2020), o conteúdo prático desta tese propõe, inicialmente, a tradução de seu código escrito em TensorFlow para PyTorch, que é uma linguagem mais flexível, bem documentada e com uma comunidade em rápido crescimento. Em seguida, é realizada uma exploração de diferentes componentes arquiteturais da rede neural para ver se cada mudança é capaz de gerar melhorias ainda maiores no aprendizado diferencial. Três arquiteturas, em comparação com a apresentada por Huge & Savine (2020), apresentaram melhores valores de RMSE e melhor desempenho percentual da aprendizagem diferencial em relação à tradicional. Além disso, também foi observada que a sensibilidade de alguns dos componentes arquiteturais da rede é tal que pequenas alterações feitas resultam em valores mais elevados de RMSE. De uma forma geral, esta dissertação pretende contribuir para o avanço dos estudos de formação de preços de opções, apresentando uma vasta história do desenvolvimento do problema estudado e criando novos benchmarks em termos de arquitetura para a linguagem de código PyTorch e de métricas de avaliação.

Palavras-Chave

Precificação de Opções - Mercado de Opções - Differential Machine Learning

Contents

Acronyms

AAD Automatic Adjoint Differentiation.

AD Adjoint Differentiation.

AI Artificial Intelligence.

ARCH Autoregressive Conditional Heteroskedasticity Model.

ATM At the money.

backprop Backpropagation.

BD Big Data.

BOPM Binomial Options Pricing Model.

BSMM Black-Scholes-Merton Model.

CVA Counterparty Value Adjustment.

Diff NN Differential Neural Network.

Diff PCA Differential Principal Component Analysis.

Diff ML Differential Machine Learning.

DL Deep Learning.

FDM Finite Difference Method.

GARCH Generalized Autoregressive Conditional Heteroskedasticity Model.

GBM Geometric Brownian Motion.

HM Heston Model.

ITM In the money.

LSM Liquid State Model.

MC Monte Carlo Simulation.

ML Machine Learning.

MSE Minimum Sum of Errors.

OTM	Out of the money.
PCA	Principal Component Analysis.
PDE	Partial Differential Equation.
RNG	Random Number Generator.
SABR	Stochastic Alpha, Beta, Rho Model.
wrt	with respect to.

List of Figures

List of Tables

Chapter 1

Introduction

Technology impacts every aspect of human life, from infrastructure to communication, from health to entertainment and its presence improves processes of all kinds, as it makes possible to do things better and faster, with lower costs and fewer risks. In concrete terms, technology makes it more efficient for people and institutions to produce and access more things, new and better experiences, goods and services.

Economy, and especially Finance, are some of the areas affected by the advance of technology. Since the first modern computers appeared in the middle of the 20th century, the speed of technical changes has accelerated given the effects of Moore's Law. After the rise of the Internet, more people have come together and increased their interactions, and thanks to the ability of devices to store records of actions and turn them into data, new information and opportunities for new markets have emerged.

Intending to increase their business, financial institutions, on the one hand, want to increase the transactions carried out by existing clients and acquire new clients. On the other, manage large investment portfolios that will make them grow, whether they belong to third parties or themselves. Their investment sectors are responsible for a large part of the world's GDP and have a wide range of assets traded, from fixed to variable income, meeting diverse demands and offering solutions and alternatives for profitability by the laws of the market.

Within the variable income market there is a subset of derivative assets, which are financial instruments whose value is "derived" from prices, reference rates, or indices, whether of individual assets, groups of assets, or entire markets. “Underlying assets” are all these asset classes which options are dependent on. The following are the most common types of derivatives: forward contracts, futures contracts, swaps and options.

This work will focus on the options market set, which are assets translated as "the right to buy or sell an underlying asset at a fixed price and a future date". By "rights" we also mean "choice" or "option" - hence the name of the product - as this non-obligation is an advantage that exists, at least on the buyer's side, unlike other derivatives. Desjardins (2022) shows that the total derivative market cap was around US$ 600 trillion, a notional value two times bigger than global real estate market, six times bigger than global stock market and sixty times bigger than gold one.

In asset management, two main strategies can be used with derivatives and with options. The first strategy is hedging, i.e., protecting against negative trends about an expected future price. Hull (2003) points out that "hedging is therefore likely to be less expensive when carried out by the company compared to individual shareholders. Furthermore, by reducing risk, companies can focus on their main activities in which they have particular skills and expertise". The second strategy is making profits from trading using leverage and arbitrage. This strategy is possible because, in the market cycle, sometimes prices stretch so far or so depressed that changes in the trend can be predicted especially with the help of technical and fundamental analysis.

An intrinsic characteristic of any financial asset, given there is no complete information in the world, is that the price formation process is imperfect, i.e., although assets are under the same economic circumstances, the expectation that they will fluctuate to the same extent is not always complied with. Therefore, what is an advantage from the point of view of commercial gains, is also a challenge for hedging (at least on the options seller's side), since poor allocation decisions and unpleasant forecasts generate few deviations that nominally translate into million dollar losses.

Hull (2003) contextualizes the significance of Monte Carlo Simulation (MC) in option pricing, noting that since its introduction by Ulam & von Neumann (1949), it has become the most widely used technique for predicting future prices in options trading. MC is still relevant today and one of the most widely used in the treatment of options, as it is capable of modeling complex systems like economic's, with considerable statistical robustness and interaction with random variables. However, a great difficulty has arisen in multidimensional pricing prediction scenarios, where a large sample of paths is required to guarantee the accuracy of the results, resulting in longer processing times and greater use of computing resources.

Faced with this macro challenge, Huge and Savine (2020a) present an alternative that has proven to be effective: in a supervised learning problem, the introduction of a differential label label in the loss calculation increases training speed and with much higher predictions. The introduction of differentials opens up discussions and tests in search of validations, confirmations and discoveries in one of the frontiers of Machine Learning (ML), hence the name Differential Machine Learning (Diff ML) (Huge & Savine, 2020a). In the same way that in traditional neural networks dimensionality is reduced with the Principal Component Analysis (PCA) technique, in parallel, Diff ML introduces the notion of Differential Principal Component Analysis (Diff PCA), which has an even greater capacity to remove irrelevant factors.

Technology will continue to advance and so will the research area of Diff ML applied in the context of derivatives pricing, including options, mainly in the search for a model capable of generalizing different markets, each with its particularities at specific times of the economic cycle, all without losing the ability to bring good results so that investment portfolios, on the one hand, strengthen their hedging positions and, on the other, increase their return.

1.1 Main Drivers

In the continuous quest to advance science and its elements of technique and knowledge, it is hard to imagine that different areas of human knowledge don't cross paths. As the boundaries between disciplines become increasingly permeable, the emergence of innovations transcends the traditional limits of knowledge. And especially with the trend towards the computerization and digitization of the world, all aspects of human life, including finance, from individuals to companies, have created some convergence with computer science, robotics and other branches of technology. As technology continues to support economic progress, it provides more information from feedback machines.

In this context, we highlight two fundamental pillars that underpin this study: (i) the economic pillar, which brings the main object of discussion of this thesis; and (ii) the technological pillar, which will represent a way to finding solutions. Agents operating in the options market continue to seek to reduce costs and risks, aiming to improve their clients' investment experience and to ensure a better management of their own portfolios. Hence, the growing use of ML technologies to achieve these goals.

With regard to the pillars, below an explanation of their importance:

- Economic: the options market is part of the derivatives market and part of a large universe of investments called Finance; contrary to common sense, Economics and Finance are different things. While Economics is a social science that studies how societies allocate scarce resources for the production, distribution and consumption of goods and services based on individual and collective decisions, both in the public and private spheres and in a national and international scenario full of uncertainties and imperfections that affect such decisions, Finance is one of the reasons how to make allocation decisions. For this purpose, Finance studies the management of money and risks in order to maximize the value of assets and optimize the obtaining and offering of financing resources. It is clear that in an uncertain and incomplete information world, one of the main objectives of any options portfolio investor is to guarantee the best return/investment ratio. To this end, decisions are made guided by micro and macroeconomic information from the past, present and future. And like any individual economic agent, investors use the element of "expectation" using probabilistic economic scenarios as their tools to fill gaps in information.

- Technologic: all of humanity's achievements can be considered technologies. Focusing on computing, various calculating machines and automata have been observed throughout history. After the Second World War, electronic digital computers emerged with advances in miniaturization, network architectures, storage devices and data algorithms. In the latter group reside innovations grouped under areas of Artificial Intelligence (AI), ML and Deep Learning (DL), a continuum of computer science areas that are increasingly efficient and effective in dealing with regression, classification and clustering problems. In the 21st century, more information has become available with the transformation into data of all kinds of actions carried out on the Internet, the Big Data (BD) phenomenon. The volume of data and the processing capacity of these algorithms have such an impact on people's knowledge and decisions that they have been considered a powerful technique to solve problems. The increased confidence in machines happens because new ways of dealing with challenges are being tested and returning good outputs to society. Whether in the architecture of the networks, in the combination of distinct data types or adding new characteristics observed in the input data, such as derivatives, machine learning grows by creating new solutions for society.

The pillars of this work represent two major sciences that have evolved over centuries and branched out into different disciplines and areas of study over time. Their advances and subdivisions have provided a wide range of research and application paths. However, in the intersection of two of these paths there is a point of convergence around real problems related to financial markets and a power solution imported from the world of computer science.

When employing advanced differential machine learning techniques, Figure 1.1 illustrates the outcome of this convergence in enhancing the efficiency of rapid option pricing.

Thus, one of the intentions of this thesis is to contribute to the practical knowledge of Diff ML in the prediction of the options market price curve in order to provide insights and test hypotheses about this meeting of paths between Economics and Technology.

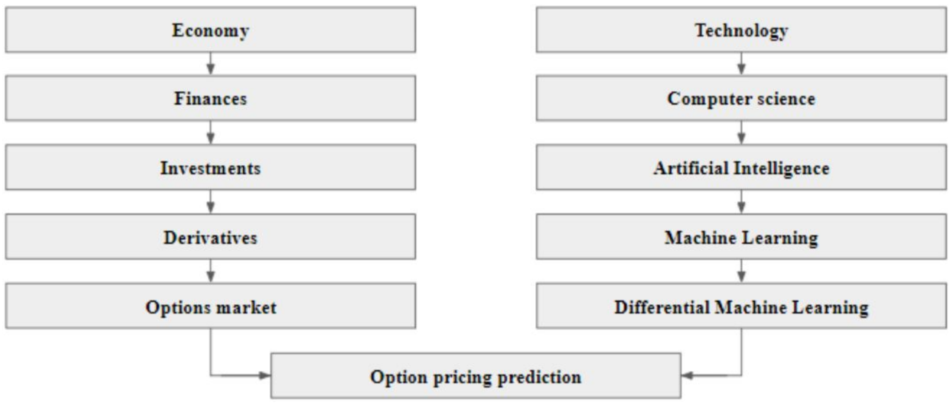


Figure 1.1: Research topic as a convergence of the main drivers.
Source: Author's own work.

1.2 Objective

Many applications are emerging in the options market thanks to the advancement of AI algorithms. One of the jobs of the data scientist is to provide insights to help companies improve product quality and make more business solutions available to their customers. Therefore, as there is room to explore so many techniques, this work will focus efforts around Diff ML algorithms, applying them to the financial market. Diff ML add relevant information in the loss without introducing bias, and carry out training with a small number of observations, resulting in better prediction metrics and with less resource consumption.

After present the evolution of options pricing models throughout history, argue the power of Diff ML in relation to standard ML and illustrate how similar it is compared with MC interactive model in terms of errors through subsequent chapters, the main objective of this thesis is to perform a supervised option pricing model for a regression problem by replacing the traditional Monte Carlo simulation by a Diff ML algorithm. This new algorithm is well known in the literature and can present good performances in different option types and many market conditions. The execution of the objective will involve translating the original code written in TensorFlow by Huge & Savine (2021a) to PyTorch. The code generates data from MC simulations and trains it in a Diff ML structure, composed of a specific architecture to deal with distinct option types and theoretical models.

Once the original code is translated to obtain a model that performs better than a standard ML model, a secondary objective is to list how changes in some restrictions affect the final result of the model and to obtain and validate the best of them. By evaluating different component arrangements, it will be possible to understand which components of a neural network impact positively in the prediction of payoffs.

To summarize, the list of tasks includes: (i) translation of all elements of Huge & Savine (2021a) code that executes the generation, preprocessing, training and testing of data; (ii) exploration of elements that allow the use the Diff ML model for different options, theoretical models and training types; (iii) evaluation of each architecture; and (iv) validade the robustness of the best architectures found. Details are shown in the methodology chapter.

By completing both objectives, both translation and validation of the best architectures, it will be possible to list the following contributions, respectively to the option pricing activity, create an architectural benchmark for future work using PyTorch as a base library and advance both theoretical knowledge of the impact of how each component impacts more or less on the differential neural network.

1.3 Outline

Below is how this document will be structured.

Chapter 2 presents essential concepts to advance in reading the thesis. Two are the sets of concepts: (i) the financial area, with an explanation of what options are, their types, how they work, some metrics and relationships between their price and economic variables; and (ii) the technological area, with the theory behind Diff ML, its practical operation, the step-by-step implementation and the advantages compared to standard ML.

Chapter 3 makes a brief review of the literature divided into two pillars: (i) in the first, a history of the main option pricing methods is made, from the first algebraic approaches with some attempt to bring economics interpretation, passing through iterative methods which uses some computation for scale gains, and opening the path for the well-known computational algorithms from the most traditional one to the recent innovation of Differential Neural Network (Diff NN); and (ii) in the second, papers are presented where Diff ML techniques are already being tested, with a first sub-block focused on option pricing, where the innovation came from and where the topic is most heated and another sub-block of applications in classic Data Science problems. The chapter intends to trace a progression in the way in which the price problem has been attacked and opens the doors to exemplify how it is currently solved more efficiently.

Chapter 4 provides the methodology divided into three main blocks. The first block details the code translation step from TensorFlow to PyTorch, explaining why to use this language and presenting the classes and methods. The second presents many scenarios where the Diff ML models can be trained and tested in terms of the type of option (economic design) and the architectural components of the model (technological design). The third block exposes the results of the metrics used to evaluate each architecture and choose the best Diff ML models that will be validated in the last and fourth block. This division intends to provide an overview of the logic sequence of the work done in partnership with BNP Paribas CIB.

Chapter 5 presents practical simulation results for each group of generated data and different architectures previously mentioned in Chapter 4. A group of them is considered the best and they are tested for different seeds to check how robust each one is.

The final chapter, the sixth, concludes this report by comparing all results observed in the previous one and examines the attainment of the proposed objectives. Additionally, the chapter outlines potential content for future work, encompassing both economic and technological dimensions.

Appendixes will provide details about some processes, mathematical issues, numbers and plots.

Chapter 2

Background

Before starting the literature review, it is important to present some concepts relating both to the consolidated options market and Diff ML. The concepts presented are crucial to create the basis for understanding the evolution of the main option pricing models and how different works inside and outside the financial area are applying Diff NNs.

The first subsection focuses on the economic pillar, presenting types of options, mechanisms of gains and losses, the problem of maximizing profit, economic variables that influence the current price of the derivative and how they achieve this influence, and some indicators widely used in the financial market. The second takes the paper by Huge & Savine (2020a) as knowledge-basis because it inaugurates the Diff ML area by comparing it to standard machine learning, both for supervised and unsupervised learning. Also, they bring the symmetric practice of Diff PCA by introducing the concept of "relevance" of a variable in contrast to the standard "variance" concept used in standard PCA.

2.1 Options Basics

When trading, there are two basic option types: call options (the right to buy the underlying asset at a future date at the contracted price) and put options (in the same way to sell). There are two types of participants: buyers and sellers. For example, in a call option transaction the buyer has the possibility to exercise its position until the option expires, while the seller is obliged to execute its position when the buyer does it. By "exercising", it means buying or selling the asset with the value fixed at the moment when contracting the option.

Execution can be done according to the property of the option, which are of several types: the so-called "European" ones allow execution only at expiration, while the "American" ones can be executed at any time. There are still "Bermudan", that may be exercised only on predetermined dates before expiration; the "Asian" when the payoff is determined by the average price of the underlying asset; the "barriers" that only are exercised if the price of the underlying asset passes a certain level, i.e., a barrier, during a certain period of time; and the "digitals" which offer a fixed payment if certain conditions happen during the term of the contract, otherwise zero profit.

Given a time interval $t = \{0, 1, 2, \ldots, T\}$, a European option, the most simple a European option, the most simple option in terms of how it works, is purchased at $t = 0$ for the expiration value called strike K and can only be sold at expiration, at $t = T$. Until this moment, the price S_t of the underlying asset fluctuates and, at maturity, it changes to S_T. So the cashflow generated by the option on the expiration date is:

$$CF_T = S_T - K \tag{2.1}$$

A transaction is defined as a set of cashflows $\{CF1, CF2, ..., CFT\}$ and the payoff π, or transaction profit, is taken as the sum of cashflows over time. However, as the European transactions only have execution at the end of the contract, the European transaction is made up of a single CF_T cashflow and the payoff is exactly equal to the value of the cashflow.

For a call, the advantage occurs when the stock price is greater than the strike:

$$\pi^c = S_T - K^c \tag{2.2}$$

And for a put the opposite:

$$\pi^p = S_T - K^p \tag{2.3}$$

Regarding the payoff sign, there is terminology that says that $i > 0$, then the option is In the money (ITM) in the process of profiting. Otherwise, when $i < 0$, the option is Out of the money (OTM) still at a loss. Finally, when $i = 0$, then the option is said to be At the money (ATM), that is, in a financially neutral position.

From the price relationship observed above, there is a detail that differentiates the operation of options in relation to other derivatives: if the option is OTM at expiration, the buyer has the choice not to exercise the contract and pay a small acquisition cost σ, called "option premium", whose notation is p for puts and c for calls. Therefore, if the premium is lower than the negative payoff, the hedging is executed, that is, the reduction of losses.

The best payoffs for a call and a put are, then, respectively:

$$\pi^c = max\{S_T - K^c; -\sigma\} \tag{2.4}$$

$$\pi^p = max\{S_T - K^p; -\sigma\} \tag{2.5}$$

Figure 2.1 illustrates, in graphic terms, the relationship between the asset price on the X axis and the payoff obtained on the Y axis (making one on your own). Strike K and option premium σ are displayed.

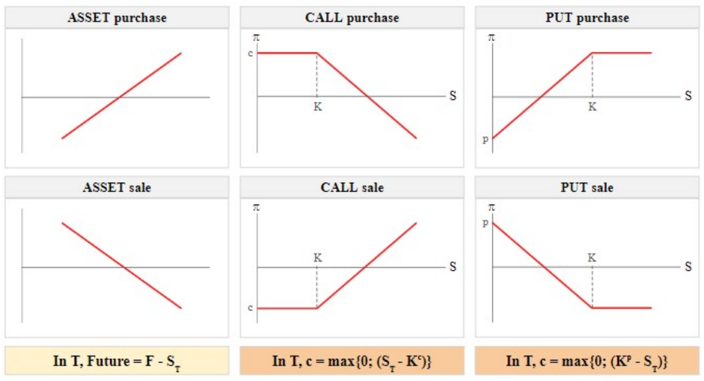


Figure 2.1: Payoffs in different option positions.
Source: Author's own work, based on Hull (2003).

Although a contract defines a target value K for the option, this target is defined only when purchasing the put or call. Up to this point, option prices are moving together with underlying assets prices. The buyer always has risks limited to the option premium, but the seller is much more exposed. Bloch (2019) remembers that "maturity risk plays against the seller and he can be exposed to an upward jump of the spot price just before maturity. As a result, the seller of the claim requires compensation for the risk he is bearing, and the buyer must pay a premium for the benefit he gets from the contract. The decisions for the appropriate pricing of such claims are made contingent on the price behavior of the underlying securities. Fortunately, the uncertainty affecting the underlying of the contract at maturity results from observed small daily movements, thus providing information that can be exploited. This information allows a model for the dynamics of the underlying asset to be defined, and a dynamic adjustment to be made against final risk. Consequently, financial markets model uncertainty by considering future trajectories of the risky asset seen as possible scenarios, which are generally assumed to be continuous functions defined on $\mathbb{R}^+$".

There are six factors that affect the current price of an option at each moment t:

1. The current price of the underlying asset S_t;
2. The volatility of the underlying asset σ_t;
3. The strike price K;
4. The validity period of the contract T;
5. The risk-free interest rate (the lowest on the market, commonly practiced by the Central Bank);
6. The expected returns from the underlying income-generating assets.

Hull (2003) summarizes the effect of these factors on the option current price. Table 2.1 is a reproduction of that, and its interpretation is as follows: an increase in the variable value affects positively, negatively or without a defined sign on European-type options. The only difference from the American ones is that, as the execution of the contract is allowed at any moment of time, the longer the validity period, the greater the opportunities for exercise and, therefore, at the present moment, the greater the value and hence the price of both calls as for puts.

Table 2.1: Effect of a variable on the option price holding other variables constant. Source: Hull (2003).

Variable	European call	European put
Underlying asset price	$+$	$-$
Underlying asset volatility	$+$	$+$
Strike	$-$	$+$
Maturity	?	?
Risk-free interest rate	$+$	$-$
Expected passive income	$-$	$+$

To enhance comprehension of the influence of certain variables, investors rely on indicators known as the "Greeks." They are a set of five measures to gauge the sensitivity of the option price P_t to various factors over time, aiding decision-making processes, as follows:

1. Delta (Δ): $\partial P_t / \partial S_t$
2. Gama (Γ): $\partial^2 P_t / \partial S_t^2$
3. Theta (Θ): $\partial P_t / \partial t$
4. Rho (P): $\partial P_t / \partial r_t$
5. Vega (V): $\partial P_t / \partial \sigma_t$

Respectively, the derivative with respect to (wrt) the current price of the underlying asset (delta), the second derivative also in relation to the current price of the same asset (gamma), in relation to time (theta), to the current risk-free interest rate (rho) and to the current volatility of the underlying asset (vega).

Given the intricate nature of the options market, characterized by its complexity and dependence on numerous external factors, investors should exercise prudence by integrating price forecasting into their activities. The more powerful the techniques, the more complete the amount of information available for investors to take the best actions. The next section presents a history of the main price forecasting methods used over the last few decades and how this advance reached current computing tools, including ML and Diff ML models.

2.2 Differential Machine Learning Basics

To understand existing relationships and produce a prediction with the best degree of accuracy, a standard neural network trains a model to fit into the data points. Neural networks using differential data in their loss work differently because they "check" how the function in prediction behaves in the neighborhood of the points. In other words, datapoints help make the model understand the dominant directions around them. Figure 2.2 shows a general picture of how the models in question work by comparing the theoretical curve of a call purchase on the left with a mockup of data observed and measured by a Diff ML model on the right.

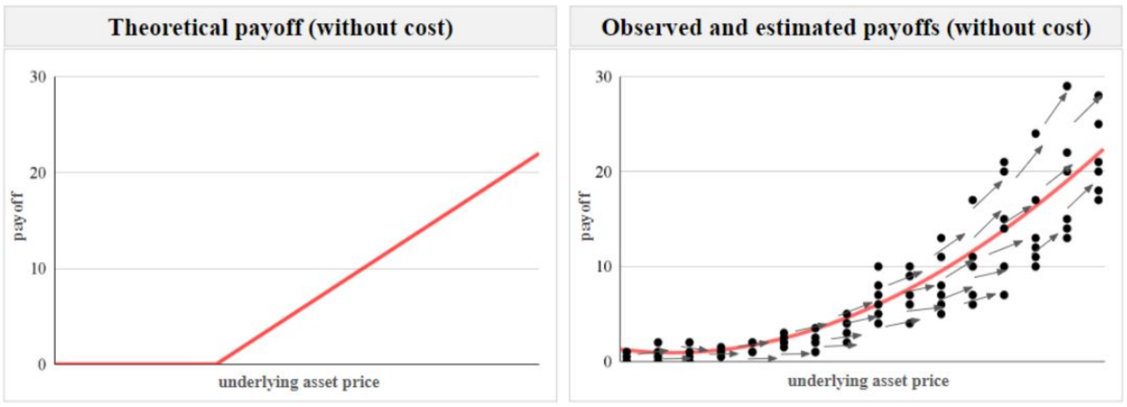


Figure 2.2: Illustration of the idea behind the differential model explanation. Source: Author's own work.

The core idea resides in the graph on the right: each option works based on a specific underlying asset, i.e., shares, government bonds, metals, commodities. Each one belongs to a micro and a macroeconomic environment with variables that influence it with more or less power. For each set of market parameters, there will be a specific curve. The mockup therefore relates the observed values of $S_t i$ (X axis) in relation to expected payoffs $S_t i - K$ (Y axis) at each moment of time and for a given set of market parameters.

Another intuition comes from Savine (2021) master class, when the author comments, with simple functions, that the derivative of a good approximation is not a good approximation of the derivative, but, conversely, the integral of a good (unbiased) approximation is a good approximation of the integral. So instead of data scientists learning from prices to get poor sensibilities, should not they do the opposite, i.e., starting from derivatives to get values at the level? This question is a good hypothesis to encourage the use of derivatives as inputs in an ML model.

As known, the model calculates derivatives. Such differentiations are, by definition, pathwise differentials $\partial\pi/\partial X_t$, where X is the set of market variables in addition to the price of the underlying asset. Huge & Savine (2020b) work on top of Black-Scholes-Merton Model (BSMM) in building their model and illustrate the value of the pathwise derivative for a European call:

$$\frac{\partial \pi}{\partial S_{T=1}} = \frac{\partial S_{T=1} - K}{\partial S_{T=1}} = \frac{\partial S_{T=2} - K}{\partial S_{T=2}} * \frac{\partial S_{T=2}}{\partial S_{T=1}} = 1 * \frac{\partial S_{T=2}}{\partial S_{T=1}} \tag{2.6}$$

S_i^t is almost perfectly continuous data (it is just not because in practice there are no divisions smaller than cents), so it is almost always possible to differentiate. Exceptions are made in some cases where "pathwise differentials are not well defined for discontinuous cashflows, like digitals or barriers. This is classically resolved by smoothing, i.e. the replacement of discontinuous cashflows with close continuous approximations. Digitals are typically represented as tight call spreads, and barriers are represented as soft barriers. Smoothing has been a standard practice on derivatives trading desks for several decades. Provided that all cashflows are differentiable by smoothing, the expectation and differentiation operators commute so that true risks are (conditional) expectations of pathwise differentials" (Huge & Savine, 2020b):

If

$$f(x) = E\left[\pi | X_{T=1}\right] \tag{2.7}$$

then

$$\frac{\partial f(x)}{\partial x} = E\left[\frac{\partial \pi}{\partial X_{T=1}} | X_{T=1} = x\right] \tag{2.8}$$

The main result of the implication in equations 2.7 and 2.8 is that difference and media equal each other. Hence, pathwise differentials are unbiased estimators of risk sensitivities. It allows the use of differentials because, despite having a relationship with the input variables (since they are derivatives in relation to the output), they do not add visual problems to the final results and, therefore, are specific to bringing additional information about the problem, thus improving the quality metrics of the model.

So, pathwise differentials are the gradients of labels Y with reference to wrt inputs X ($\partial Y / \partial X$). At this point, a clarification is necessary: in Economics, the relation between an input and their output is called "elasticity", that symbolizes the estimated risk sensibilities of X in relation to $\hat{Y}$ ($\partial \hat{Y} / \partial X$). So pathwise differentials are not sensibilities, but unbiased estimators of risk sensitivities, the Greeks seen in section 2.1.

It is worth remembering that "in pattern backpropagation, feed-forward networks are efficiently differentiated, which is generally applied to compute the derivatives of some cost function with relation to the weights and biases for optimization. For now, we are not interested in those differentials, but in the differentials of the predicted value $y = z_l$, with l layers from 1 to L, with relation to the inputs $x = z_0$" (Huge & Savine, 2020b), i.e., $\partial Y / \partial X$, the pathwise differentials.

In a model where:

$$\begin{cases} z_0 = x \\ z_l = g_{l-1}(z_{l-1}) * w_l + b_l \\ y = z_l \end{cases} \tag{2.9}$$

The differentiations obtained in the backpropagation step are:

$$\begin{cases} \frac{\partial y}{\partial z_l} = \frac{\partial y}{\partial y} = 1 \\ \frac{\partial y}{\partial z_{l-1}} = \frac{\partial y}{\partial z_l^T} \circ g'_{l-1}(z_{l-1}) \\ \frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_0} \end{cases} \tag{2.10}$$

The derivatives wrt Y participate in the training of the Diff NN trained by the authors and they use a specific architecture to represent how the model functions, the so-called "twin network". This architecture starts as a traditional feed-forward, but makes a new feed-forward starting from Y and the pathwise differential towards the X again. Figure 2.3 illustrates how a twin network functions.

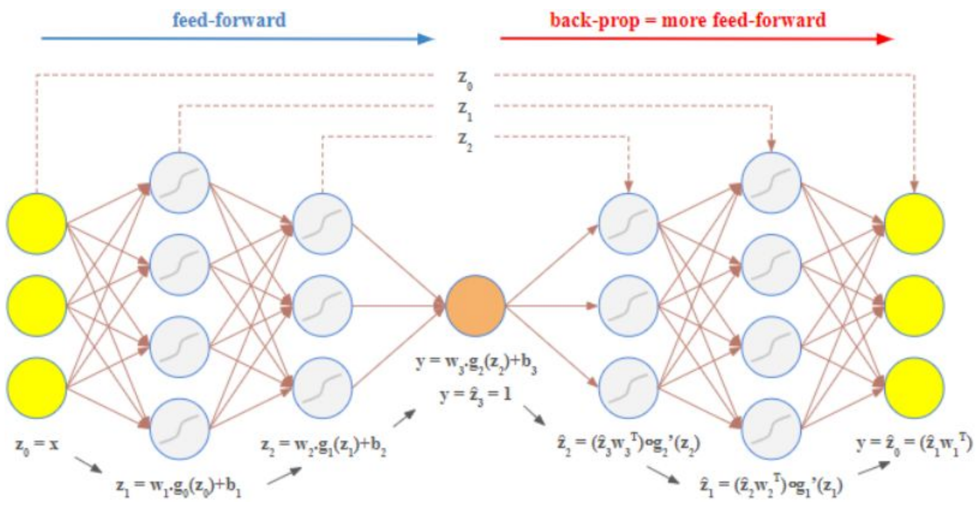

feed-forward
back-prop = more feed-forward
z_0
z_1
z_2
$z_0 = x$
$z_1 = w_1 \cdot g_0(z_0) + b_1$
$z_2 = w_2 \cdot g_1(z_1) + b_2$
$y = w_3 \cdot g_2(z_2) + b_3$
$y = \hat{z}_3 = 1$
$\hat{z}_2 = (\hat{z}_3 w_3^T) \circ g_2'(z_2)$
$\hat{z}_1 = (\hat{z}_2 w_2^T) \circ g_1'(z_1)$
$y = \hat{z}_0 = (\hat{z}_1 w_1^T)$

Figure 2.3: Twin network example.
Source: Author's own work, based on Huge & Savine (2020a).

When training differentiated labels, one wants to estimate the correct prices of *f(x)* from an estimated function $f(x; w_l; b_l)$, which learns the optimal weights of w_l and b_l from the training data $(x_i\ ;\ y_i\ ;\ \frac{\partial y_i}{\partial x_i})$. Discovery occurs through the computation of z_l values at each layer of the network, as observed in system of equations 2.10.

In classical training, the objective is to minimize the difference between Y and $\hat{Y}$, for example through Minimum Sum of Errors (MSE):

$$\min \left\{ \text{MSE} = \frac{1}{m} * (Y - \hat{Y})^T (Y - \hat{Y}) \right\} \tag{2.11}$$

In differential training, in turn, the objective is to minimize the difference between the pathwise differential ($\partial Y / \partial X$) and the estimated risk sensibilities ($\partial \hat{Y} / \partial X$):

$$\min \left\{ \text{MSE} = \frac{1}{m} * \text{tr} \left[\left(\frac{\partial y}{\partial z_0} - \frac{\partial y}{\partial x} \right)^T \left(\frac{\partial y}{\partial z_0} - \frac{\partial y}{\partial x} \right) \right] \right\} \tag{2.12}$$

i.e.

$$\min \left\{ \text{MSE} = \left(\frac{\partial y}{\partial x} - \frac{\partial \hat{y}}{\partial x} \right) \right\} \tag{2.13}$$

So, the loss used in a differential learning is:

$$\min \left\{ \text{MSE} = \alpha . \left(y - \hat{y} \right)^2 + \beta . \left(\frac{\partial y}{\partial x} - \frac{\partial \hat{y}}{\partial x} \right)^2 \right\} \tag{2.14}$$

where $\alpha + \beta = 1$

A logical condition that arises, therefore, is that in the best case scenario, when the price forecast is perfect, that is, when the best smooth payoffs are achieved, the risks are equal to zero. In practice, differential learning does not need to work with twin networks. More than that, it can be a simple feed-forward where, instead of just having X as inputs, it has differentials inputs also. So, the logic condition seen above, about the inversion relation between prediction and risk, is architecture-agnostic in Diff ML.

With differentiators available, a new vector of information is added and helps a model capture new relationships between features and labels. For example, consider three variables $(x_1 \, ; \, x_2 \, ; \, x_3)$ with their respective parameters $(p_1 \, ; \, p_2 \, ; \, p_3)$ and their respective derivatives wrt y $(d_1 \, ; \, d_2 \, ; \, d_3)$. Assuming the following scenario:

$(p_1 \, ; \, p_2 \, ; \, p_3) = (100 \, ; \, 100 \, ; \, 1)$

$(d_1 \, ; \, d_2 \, ; \, d_3) = (0 \, ; \, 0.5 \, ; \, 1000)$

In this situation, in a dimensionality reduction scenario, x_3 is easily eliminated, given that it is 100 times less explanatory at the level. However, in relative label terms, it is a relevant variable. Depending on the modeling, x_1 and x_2 could be eliminated, or all three could be considered relevant variables. Thus, while a dimensionality reduction via PCA, for example, seeks for variables that most explain the variation in y, in the differentiated version, Diff PCA looks for the ones most relevant to explain y. Hence the introduction of the concept of "relevance". "For this reason, Diff PCA may be safely applied to aggressively remove irrelevant factors and considerably reduce dimension. In the context of data generated by financial Monte-Carlo paths, Diff PCA exhibits the principal risk factors of the target transaction or trading book from data alone" (Huge & Savine, 2020c).

In this dimensionality reduction technique, it is necessary to standardize the variables. Since the features in the level undergo through linear transformation:

$$\tilde{y} = \frac{y_t - \mu_t}{\sigma_t} \tag{2.15}$$

$$\tilde{x_j} = \frac{y_{ij} - \mu_j}{\sigma_j} \tag{2.16}$$

Then for each differential input $\frac{\partial y}{\partial x_j}$:

$$\frac{\partial \tilde{y}}{\partial \tilde{x}_j} = \frac{\sigma_j}{\sigma_y} * \frac{\partial y}{\partial x_j} \tag{2.17}$$

From the standardized differentiated labels, it is possible to apply PCA normally, as the authors say after mathematical demonstrations. The authors show that even with data standardization, PCA can give worse results comparing the level. Furthermore, when the loadings are computed, the sum of their values for a given component may be greater than for other components, even if their variation is smaller. To explain the price curve, the sum of the loadings, rather than the norm, is what determines the variation of the short rate. And despite some negligible variation of a component, variation is simply not the correct criterion for dimension reduction because by removing some component based on it, important information is lost for a prediction. PCA only works as intended when the choice of representation is appropriate for a given problem. Because PCA is unsupervised, it does not know anything about the problem and it cannot infer the correct representation by itself. In other words, to safely apply PCA, one must correctly guess the correct result beforehand. (Huge & Savine, 2021b).

Diff PCA is simply PCA on differential labels, but unlike with this one, it is safe to filter irrelevance aggressively. A little change in a derivative can procure a big difference in the level and that is why Diff PCA is a more sensible method to clean irrelevant features. Thus, while PCA only works by looking at input variables, Diff PCA looks at the relation between inputs and outputs (Huge & Savine, 2020c). The interpretation of the cleaning is distinct too: assuming that a prior PCA step was performed, differentials are expressed in 'standard deviation of labels per standard deviation of inputs'. For example, if δy/δx=0,01, then 100 input standard deviations are needed to produce 1 output standard deviation and x is an irrelevant feature for y.

The interpretation of the cleaning is distinct too: assuming that a prior PCA step was performed, differentials are expressed in 'standard deviation of labels per standard deviation of inputs'. For example, if $\partial y / \partial x = 0.01$, then 100 input standard deviations are needed to produce 1 output standard deviation and x is a irrelevant feature for y.

More differences are made by Huge & Savine (2020c). Standard PCA performs an orthonormal transformation of inputs and eliminates constant and redundant ones, facilitating subsequent training of neural networks. Diff PCA further rotates data to an orthogonal relevance representation and may considerably reduce dimension, in a completely safe manner, by elimination of irrelevant directions. In the context of financial simulations, Diff PCA computes an orthogonal hierarchy of risk factors for a given transaction. For example, it identifies basket weights as the only relevant risk factor for a basket option, from simulated data alone: traditional risk reports are averages of these trajectory differentials, but the average ends up reducing the available information. A risk report for a delta-hedged European call option results in a zero value [at the level]. However, the price of the underlying share remains a relevant risk factor [in the derivative].

By eliminating this information at the media level, a trading book is affected in a non-linear way, but the information of its importance is embedded (not to say ”hidden”) in the trajectory differentials. Path differences therefore represent reward sensitivities across a variety of scenarios and they have a more comprehensive narrative than aggregate risk reports.

The example above shows the ability to positively impact the analysis of financial data, both with the introduction of derivatives in the input data and when cleaning, taking into account the derivatives themselves.

Related to the use of differentiated labels, Huge & Savine (2020b) comment that it converges much faster, allowing to train accurate approximations with much smaller datasets, as we see in the numerical examples, because (i) the effective size of the dataset is much large evidently as m training examples generating $n * m$ differentials (n being the dimension of the inputs x_j); (ii) the neural nets picks up the shape of the pricing function learning from slopes rather than points, resulting in much more stable and potent learning, even with few example; (iii) the neural approximation learns to produce correct Greeks by learning the correct shape and correct orders values in different economic scenarios; and (iv) differentials act as an effective, bias-free regularization since it converges to the correct approximation seen in Implication 1.

And regarding dimensionality reduction via relevance, Huge & Savine (2020c) conclude that differential PCA makes a major difference for training function approximations, reducing dimension, stabilizing nonconvex numerical optimization and reducing sensitivity to initial seed and hyperparameters like neural architecture or learning rate schedule. Hence, PCA is bad because it may truncate very relevant coordinates when relevance does not align with variation, so it should never be applied for pricing and risk: a safe dimension reduction must be supervised and that happens with Diff PCA. In short, it is possible to use the standard PCA in a standard ML or a Diff ML model, but the results will not be better compared to the application of Diff PCA, either in the ML model or in Diff ML one.

Saying that, Figure 2.4 compares data forecasting with the original information between a classical neural network that ran 64k lines (blue dots) and, in contrast, a Diff NN with only 8k observations (pink dots). Both models went through the same dimensionality reduction process with Diff PCA.

In the first view, pink dots can be considered by someone an overfitted model. It can be an overfitted standard neural network result, but for the differential reality, it is not.

The power of Diff ML is such that the sum of error was equal to 12,85M for a classical model with 64k observations and equal to 1,77M for a differential model with 8k observations. In linear terms, it’s 7,25x better with 8x fewer observations (Huge & Savine, 2020a).

It means that, for a Diff ML model to be considered overfitted, it would be necessary to increase the number of observations until the moment that the points would be infinitely far from the regression support line. In the limit, the regres-

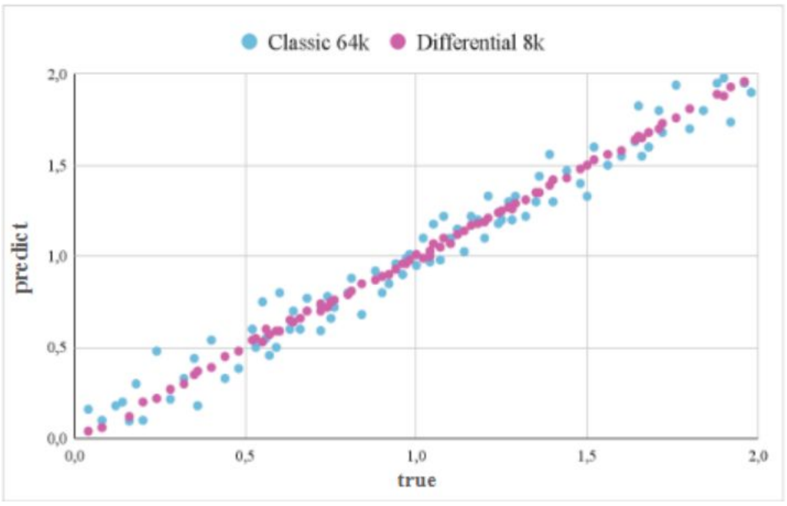


Figure 2.4: Comparison between classic and diff ML previsions.
Source: Huge & Savine (2020a).

sion line is the overfitted Diff ML model. But this situation is just hypothetical because with a small fraction of the classical model amount of data it is possible to achieve considerably better results.

A final comparison was made by the authors, who conclude that for Figure 2.4 example, the twin 8k error is very similar to the Monte-Carlo pricing error (1.70M with 8k observations). So in this very representative example, the twin network has the same degree of approximation as orders of magnitude slower nested Monte-Carlo simulation. (Huge & Savine, 2020a).

Based on the benefits mentioned by the introduction of differentiated labels related to the use of less data with increasing on speed of processing and improvement of metrics without adding bias, by the benefits of cleaning via relevance and by the similarity to the results observed with MC, the impact and power of Diff ML is evident not only for pricing derivatives prices, but for all other areas of human knowledge where standard ML has already or not been applied.

Chapter 3

State of the Art

Applying the most modern forecasting techniques to a type of product that has many specifications and particularities, such as option-type derivatives, requires extensive technical skills in computer science and a reading of the reality of the economy. The complexity of these areas and how they interact requires a review of their history.

Having seen the basic concepts of the two pillars of this report, we move on to the state of the art with a subsection for each. The first reviews the main option pricing models throughout history, showing the evolution from algebraic methods, through iterative methods and with the first computer applications, both in already consolidated models and based on simple data. The second presents techniques applied in the last 20 years to speed up the processing of forecasts, one of which is the use of derivatives in the labels and the loss calculation, the object of study in this research. The third looks directly at Diff ML models, focusing on the most recent applications, showing its potential for the specific situation of option pricing (from where it originated) and also in classic Data Science problems.

The abovementioned knowledge set compounds the basis of this chapter: starting with equations where economic theory had a strong influence and moving on to the present day when data overlaps theories, the economic and technological pillars converge and prepare the ground to understand what challenges still exist in the practice of option pricing and how a Diff ML technology already being put into practice can help with optimal solutions.

3.1 Option Forecasting Historic

Mathematical models generate a consistent sequence of outputs, revealing predictability based on input data. On the other hand, variables that change over time in an uncertain way are called stochastic. Whether deterministic or stochastic, it is possible to observe variables over time in a discrete or continuous manner. However, the prices of underlying assets, with the exception of crypto assets, whose markets operate 24 hours a day, 7 days a week, operate either continuously during the trading session or sequentially from one day to the next.

Over the decades, many methods were developed for calculating the price of options. Since the European ones do not allow liquidation before the expiration date, which is very useful for modeling because it adds degrees of freedom, risks and costs with unexpected and unwanted trajectories of the underlying assets can be better reduced. That said, there are three main approaches to option price modeling throughout history, which will be presented in the next section of this chapter: (i) analytical, with mathematical equations that have statistical support; (ii) interactive, when there is no descriptive model and, instead, sequential movements are generated according to a stipulated logic; and (iii) computational, based on numerous artificial intelligence techniques, from the simplest to the most advanced. This third one is not presented, because it serves as the introduction of subsection 3.2.

3.1.1 Analytical Approaches

The first technique for pricing options was introduced by Bachelier (1900), with the famous "Bachelier model", inspired by "brownian motion" associated with drift. Widely known and used in physics throughout the 19th century, brownian motion is characterized by a process whose trajectory is subject to continuous and random changes. This characterization of non-predictability provided Bachelier with an argument in favor of free-floating prices, so that the speculator's expected payoff would be equivalent to zero. Hence, the option price V_t depends only on an erratic random value whose values are independent of each other and in time, following a normal distribution $N(0;\epsilon_t^2)$:

$$V_t = V_{t-1} + \epsilon_t \tag{3.1}$$

Markov (1906) produced the first results for a stochastic process known as the "Markov Process", whose main property is that the value of the next step only depends on the current information, with no influence from previous steps. Also stochastic, this process, applied to Finance, is equivalent to saying that the current asset price already carries the necessary information from the recent past and comes a little closer to the reality of asset pricing that, in practice, is discrete. The so-called "Markov state" says that the price of the option is equal to the expected profit that can be obtained given a set of information Ft available only and only at the present time of contracting (F_0). This set of information refers to the price of the asset (S_0) and the state of the statistical model used for forecasting (U_0):

$$V_t = E[\pi|F_0] = E[\pi|S_0;U_0] \tag{3.2}$$

The focus on the present moment that characterizes Markov is a reference for pricing models, as market variables values are always changing. This characteristic is analogous to the incremental independence property of brownian motion. It follows that this essentially respects the Markov process.

Having said, considering that a random variable follows a Markov stochastic process $\Phi(0;1)$ with mean 0 and standard deviation 1 for a given period, if the process is divided into two sub-processes of equal time, independent and normally distributed, then each of them will follow a distribution process $\Phi(0;\sqrt{0.5})$. Such a transformation helps, in the end, to arrive at a continuous Markov process over time. In particular, the change in the variable during a short time interval of size δt is $\Phi(0;\sqrt{\delta t})$. The process described above is known as the "Wiener Process", a continuous-time stochastic process proving that the brownian trajectory has infinite length between any two points. Applied to the price of underlying assets for non-dividend paying assets, it is necessary to make a change to the process, because if an investor demands a return when the asset is worth S_1, it will continue to demand the same percentage when it is worth S_2, so that, maintaining the same rate of return over time, the price S of the underlying asset behaves exponentially, which is why this new process is called Geometric Brownian Motion (GBM) (Hull, 2003).

The discrete version of the GBM is presented below in two similar forms:

$$\frac{\delta S}{S} = \mu.\delta t + \sigma.\epsilon.\sqrt{\delta t} \tag{3.3}$$

$$\delta S = \mu.S.\delta t + \sigma.S.\epsilon.\sqrt{\delta t} \tag{3.4}$$

Variable δS is the change in the asset's price S over a very short period of time, the volatility and a random variable with a normal distribution (0;1). It means that $\frac{\delta S}{S}$ follows a probability distribution with mean t and standard pattern t, i.e., the price change follows a normal distribution $\Phi(\mu\delta t;\sigma\sqrt{\delta t})$. In the end, when $t \to 0$, you have a continuous stochastic process.

Another considerable contribution was made by Samuelson (1960), who, based on Bachelier's work, took up the issue of stock dynamics by stating that asset prices can be considered random processes and, consequently, a leak predictor. However, the most important advance lies in the fact that it is not the prices of underlying assets that follow a normal distribution, but their returns. This approach makes it possible to explain the occurrence of negative returns and to model the volatility presented in risk markets.

The GBM combined with Samuelson's innovation provided the basis for the most important model for the pricing option problem since Bachelier, the famous Black-Scholes-Merton Model (BSMM). It assumes that stock prices follow a brownian motion.

Black and Scholes (1973), with input from Merton (1973), introduced their model for pricing European options based on a hedging strategy and taking into account factors other than the price of the asset S and volatility , such as a W_t Wiener process:

$$V_t = V_{t-1}.exp\left(\frac{\sigma^2}{2}.(T_t - T_{t-1}) + \sigma.(W_t - W_{t-1})\right) \tag{3.5}$$

The BSMM equation is a Partial Differential Equation (PDE) that governs the evolution of European option prices based on a set of assumptions, namely:

1. The option is of the European type;
2. There are no transaction costs or fees;
3. No income is earned during the lifetime of the option;
4. The practice of "short selling" is permitted, i.e., selling at a price in order to buy back at a less expected price;
5. Asset prices follow the GBM assumption in that they are constant;
6. The interest rate r is constant;
7. The maturity time T is constant; and
8. Trading is continuous (not discrete).

The success of the model, however, was such that it managed to be processed in other ways over the decades with some of these assumptions relaxed, for example, can be described as a function of T. Another feature is that in the model equation there is no variable that affects investors' risk preference like interest rate at which investors neither arbitrage nor demand an additional price for security, i.e., the model is risk-neutral (Hull, 2003).

Black, Scholes and Merton were also able to unite two objectives that are commonly approached separately, that of option pricing and the hedging definition strategy, as in continuous negotiation [in the shortest time interval] and with the rebalancing of portfolio positions via new claims (purchases and sales of assumption 4), it is possible to eliminate the risks of the option with the replication of payments. The result is prices of the option offered by the PDE at each moment of time (Bloch, 2019).

A risk-free portfolio can be established due to the common influence of stock price movements on both the stock and derivative prices. In the short term, the derivative price is perfectly correlated with the underlying stock price. By creating a well-balanced portfolio of the stock and the derivative, any gains or losses from the stock position counteract those from the derivative position, ensuring the overall portfolio value is certain at the end of the brief period. "In BSMM, the position in the stock and the derivative is riskless for only a very short period of time. To remain riskless, it must be adjusted, or rebalanced, frequently. For example, the relationship between c and S in our example might change from $\Delta c = 0,4.S$ today to $\Delta c = 0,5.S$ in 2 weeks. This would mean that, in order to maintain the riskless position, an extra 0.1 share would have to be purchased for each call option sold. It is nevertheless true that the return from the riskless portfolio in any very short period of time must be the risk-free interest rate. This is the key element in BSMM analysis" (Hull, 2003).

Figure 3.1 illustrates, for the example of a call, the importance of a point-by-point analysis (the smallest t possible) to maintain the balance of the c-S relationship.

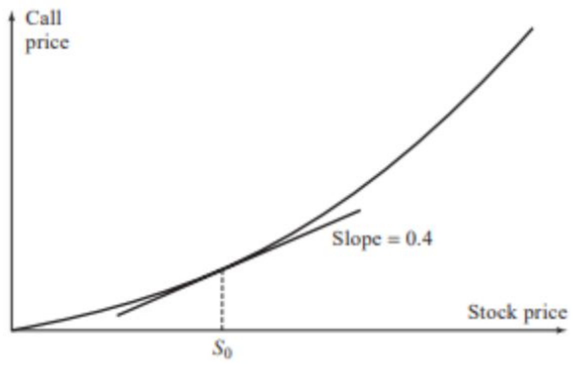


Figure 3.1: Relation between stock price and call price. Source: Huge (2003).

Once the pricing bases were created with a good model for continuous pricing and another for discrete pricing, the exploration of price volatility followed throughout the 1980s and 1990s. As presented at the end of section 2.1, volatility is responsible for defining the option price and has its own greek. Hence, the importance of modeling it. Despite being powerful, BSMM assumes price variation as something known and constant, which is unrealistic, while BOPM does not rely on this information. In practice, however, investors take volatility into account during their operations. "They assume the probability distribution of an equity price has a heavier left tail and a less heavy right tail than the lognormal distribution of Black-Scholes model. The so-called 'volatility smile' defines the relationship between the implied volatility of an option and its strike price. For equity options, the volatility smile tends to be downward sloping. This means that OTM puts and ITM calls tend to have low implied volatilities, whereas OTM calls and ITM puts tend to have low implied volatilities" (Hell, 2018).

"Smile volatility" is a resultant structure of future volatility that market participants expect the underlying asset to exhibit over the option life. There are some theoretical explanations for the emergence of the volatility smile. For example, in call options with strike prices significantly higher than the current asset price, implied volatility is higher due to the possibility of sharp price movements. The presence of the "smile" can also be explained by the asymmetry in the distribution of returns on the underlying asset. Price movements in the opposite direction often result in higher implied volatility for OTM options. Furthermore, when the option price is exactly worth the strike value, there is no risk involved, which causes the central region of the curve to be lower.

Figure 3.2 displays the "smile" model, which can be observed in different perspectives. Graphic (a) curve is the most common behavior as in the hedging strategy market agents are willing to pay more for protection against falls or rises that go against the desired trend. In the skew curves (b) and (c), respectively, market participants are willing to pay more for downside protection than for upward movements, and vice-versa. In graphic (d), the frown curve or "inverted smile" is an atypical situation in which investors are more concerned with price movements towards the money than with extreme movements outside the money, in or out, whether it may be temporary misalignment of markets and expectations.

Advances in the study of volatility have brought many modeling innovations.

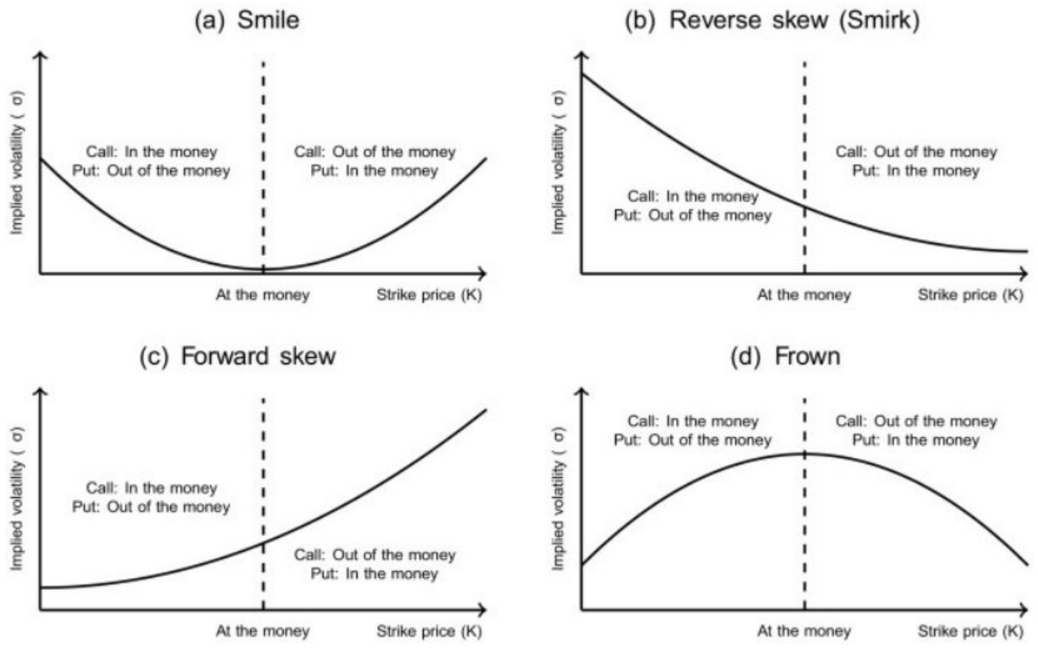


Figure 3.2: Main volatility curves in relation to the strike value.
Source: Soini & Lorentzen (2019).

Engle (1982) introduced the Autoregressive Conditional Heteroskedasticity Model (ARCH) model for time series into the literature, playing a fundamental role in understanding conditional heteroskedasticity. In a subsequent development, Bollerslev (1986) improved the model by incorporating additional terms into the modeling of conditional variance. These terms included autoregressive components in the conditional variance equation, resulting in the generalization of the ARCH model to the well-known Generalized Autoregressive Conditional Heteroskedasticity Model (GARCH). Such advances marked an evolution of volatility modeling and contributed to the advancement of these applications in financial series.

Hull & White (1987) provide a first application of stochastic volatility for option pricing, which is affected by three elements: a tendency to reverse the size of volatility, a stochastic term multiplied by the current volatility and a stochastic brownian term. The Hull-White model provides a more accurate and flexible framework for option pricing than BSMM, especially for options with longer maturities. They also consider that the biases that arise when assuming constant volatility in determining option prices are significant, especially for more distant maturities, both for options on futures, currencies and commodities.

Heston (1993), building on BSMM, progressed by providing a stochastic volatility model that offers a closed-ended solution for the price of a European call option when the spot asset is correlated with volatility, adapting BSMM to incorporate stochastic interest rates. Thus, the model could be applied to bond and currency options. The Heston Model (HM) was designed to accommodate the presence of all volatility scenarios existing in the options market, consolidating itself as an integral part of the series of main options models.

For this reason, the model is presented as a system of equations, where v is modeled as a function of a parameter θ, which is the long-term value of volatility and the parameter κ which is the speed of reversal from an extreme value to the long-term value. The volatility v is called vega because the parameter σ represents the volatility of the vol itself. S is the underlying asset price and W_1 and W_2 Wiener processes, one for each state:

$$\begin{cases} dS = r.S.dt + \sqrt{v}.S.dW_1 & (3.6) \\ dv = \kappa.(\theta - v).dt + \sigma.\sqrt{v}.dW_2 & (3.7) \end{cases}$$

Volatility is an important topic, as it can predict extreme events that follow by helping to understand the moment in the economic cycle in which assets are in, be it crisis, hype or stability. For example, in economic growth moments, less volatility in asset prices is expected, as is the opposite. From this observation, there is another group of models that deal with extreme events in their prediction, the so-called “jump models”.

The first model that incorporates sudden movements in asset prices ("jumps") was introduced by Samuelson (1969) when considering the possibility of unexpected events or unscheduled information causing significant movements in asset prices. Focused on the options market, Merton (1976) modeled the price jumps of underlying assets considering them as Poisson processes, which made it possible to capture extreme movements and unexpected events in asset prices in a discrete sequence. Merton also expanded Samuelson’s concept when dealing with asset prices, combining the brake components that are jumps with the more continuous ones based on the diffusion models used since Bachelier. The so-called “jump-diffusion models” provide a modeling spectrum between pure randomness on the one hand and discrete controlled movements on the other, such that the choice of components depends on the nature of the data and the ability to model extreme events in asset prices.

A major milestone in jump-diffusion was achieved with Whaley (1993), because while previous models were more focused on pricing options in relation to the price of the underlying asset and its volatility, the author proposed the creation of options directly related to implied volatility, i.e., the idea that it could be treated as a tradable underlying asset in its own right. This approach provides a more direct and specific way of trading, something that until then was modeled differently and which contributed significantly to risks understanding and management. The author says that “market volatility derivatives should prove to be valuable risk management tools for option market makers, portfolio insurers, and covered call writers”.

The equation for the price of asset S in a jump-diffusion model can be expressed as a function of the risk-free rate r, the average intensity λ of the jumps (average number of jumps per unit of time), the average magnitude μ of the jumps, the volatility σ of the asset, a Wiener process W for the asset, a function J that represents the value of the jump in the price of the asset and a Poisson process N that models the occurrence of the jumps:

$$dS = (r - \lambda.\mu).S.dt + \sigma.S.dW + J.dN \tag{3.8}$$

Advances in modeling followed. A model of great relevance at the turn of the century is the Stochastic Alpha, Beta, Rho Model (SABR), designed by Hagan *et al.* (2002). The authors solved a problem with the smile volatility theory when comparing its predictions with observed market data: when the price of the underlying decreases, local vol models predict that the smile shifts to higher prices; when the price increases, these models predict that the smile shifts to lower prices. Due to this contradiction between model and market, a stochastic differential system of equations is proposed to bring a more nuanced approach to understanding and managing volatility risk in financial markets.

The SABR model introduces four main parameters (three of them in the acronymous): (i) alpha as the volatility of volatility (similarly as previous theories), beta as the elasticity of volatility in relation to the price of the underlying asset; (iii) rho as the correlation between asset price and volatility; and nu the initial volatility level. F is the underlying asset and Wi are Wiener processes for each state:

$$\begin{cases} dF = \alpha.F^{\beta}.dW_1 & (3.9) \\ d\sigma = \alpha.\sigma.dW_2 & (3.10) \\ dW_1.dW_2 = \rho.dt & (3.11) \\ \sigma_0 = v.F^{(1-\beta)/2} & (3.12) \end{cases}$$

SABR influenced many trading desks in the financial market and expanded options trading in the 2000s, especially for European options, which it was initially built around. Variations of this model followed and were extended to other products. Even so, as Huge & Savine (2020) point out, like any mathematical formula, it is arbitrary to work with a specific product, requires computational effort to process and is based on mathematics rather than examples from reality.

In addition to the broad lines of research in option pricing models, diffuse or jump, discrete or continuous, and deterministic or stochastic, for different variables, such as average, volatility or interest rate, other important aspects include models: (i) general equilibrium, commonly mathematicians and quite classics in Economics, where the aim is to integrate different economic variables to understand the impact on asset prices; (ii) Bayesians, incorporating probability and probability intervals to make price estimates; (iii) filtering models, when there is asymmetric information and it is desired to select part of the information for the function design work; and (iv) more recently, machine learning. All of them bring interpretation gains, respectively, macroeconomics, uncertainty, incomplete information and the computational capacity to evaluate as many elements as desired.

3.1.2 Iterative Approaches

Option pricing theory has a long and illustrious history, but it also underwent a revolutionary change in 1973. At that time, Fischer Black and Myron Scholes presented the first completely satisfactory equilibrium option pricing model. (...) Unfortunately, the mathematical tools employed in the Black-Scholes and Merton articles are quite advanced and have tended to obscure the underlying economics (Cox *et al.*, 1979). For this reason, Cox, Ross and Rubinstein developed in the late 1970s a simpler model based on discrete time through iterative calculations rather than an analytical approach as previously presented by BSMM and show that this is a special case of the "binomial trees" or Binomial Options Pricing Model (BOPM), meaning that this approach is helpful when there is no exact mathematical formula to serve as a model.

Cox *et al.* (1979) are interested in a simple and interpretable model, without so much mathematical complexity and which financial perspectives are observable. The main idea is that, at each moment in time, the option price has a probability q to go up u% and probability $1 - q$ to go down d%. Figure 3.3 schematizes a tree with just 1 node, which evaluates prices C_u and C_d (respectively rise and fall) for the example of a call.

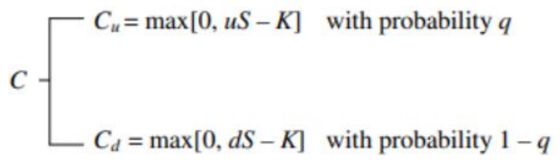

$C_u = \max[0, uS - K]$ with probability q
C
$C_d = \max[0, dS - K]$ with probability $1 - q$

Figure 3.3: Binomial tree composed of a single step. Source: Cox *et al.* (1979).

In order to discover what is the position of financial neutrality, where neither profits or losses are made, the authors create, for the call example, a portfolio composed of options and shares corresponding to them with the aim of hedging. The wallet contains Δ shares of stock and the dollar amount B in riskless bonds. This will cost $\Delta S + B$. At the end of the period, the value of this portfolio will be, as seen in Figure 3.4:

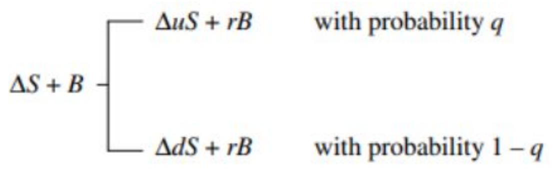

$\Delta uS + rB$ with probability q
$\Delta S + B$
$\Delta dS + rB$ with probability $1 - q$

Figure 3.4: Hedging strategy in a 1-step binomial tree. Source: Cox *et al.* (1979).

A property that arises from the discovery of the values of Δ and B is such that there is no need to know the probability q of movement:

$$\Delta = \frac{C_u - C_d}{S * (u - d)} \tag{3.13}$$

$$B = \frac{u * C_d - d * C_u}{r * (u - d)} \tag{3.14}$$

When returning to the calculation of the payoff value C, where $C = \Delta S + B$, a new parameter p is described, where $p = \frac{u-d}{r-d}$, whose interpretation is the value that q would have if investors were risk neutral. And that is exactly what happens, because when rewriting C as a function of C_u and C_d, it has to:

$$C = \left[\frac{p * C_u + (1-p) * C_d}{r}\right] \tag{3.15}$$

In other words, BOPM guarantees neutral risk assessment, because the payoff is equal to its expected future value discounted from the risk-free rate r, which is known data and is considered constant over time. Hence, the model depends essentially on the values u and d, which is simple to solve, creating a tree that follows N steps, each with a minimum variation, for example 0,1%. The number of steps generates 2^N expected payoffs. Going back through Equation 3.8 through all the steps, we have not only the value of C, but also the information whether the option is OTM, ATM or ITM.

In the 1970s, another method started to be more frequently processed, the well known Finite Difference Method (FDM), a heuristic for predicting derivative prices that, like BOPM, is used when there is no analytical solution available. FDM has no specific bibliographic reference, because it is a spread technique developed over space and time. It consists of all discrete PDEs that model the option price behavior, dividing the problem domain into a grid and estimating the partial derivatives in the cells of this grid. It leads to a set of algebraic equations that can be solved numerically to obtain derivative prices.

Hull (2003) summarizes FDM by saying that "it solves the underlying differential equation by converting it to a difference equation. They are similar to tree approaches in that the computation works from the end to the beginning of the derivative lifetime. It has an explicit method that functionally is the same as a trinomial tree. And there is an implicit method that is more complicated but has the advantage that the user does not have to take any special precautions to ensure convergence".

In simple problems with a very low dimension, solving a differential problem is not such a complex process in mathematical terms and is not very costly in computational scope. In general, they are composed of the following steps: (i) write down the PDE, i.e., the problem description; (ii) decompose the domain to reduce the problem in a finite domain, like in a 2D or 3D-grid; (iii) approximate derivatives inserting finite difference approximations for n-order derivatives into the PDE, for example, using polynomial Taylor Expansion; (iv) rewrite de PDE that has been approximate with finite differences and discretized on the grid as a linear equation; (v) solve the linear system; (vi) in a post process step, ask how good is that solution; if not, repeating the process with a new step (iii) n-order number.

The disadvantage presented by both BOMP and FDM is that, as they grow in size with each new observation over time, they begin to be very slow to process on machines. This high time consumption is already observed with just three variables. Furthermore, when the payoff depends on the past and the present values of any of the variables, especially the price of the underlying asset, there is a complexity that arises from the need to represent the payoff function in a discrete multidimensional grid and, even through models If possible, the weight of computational cost is the main factor. Due to these scalability problems, there is a method that is the most used today both in scientific and corporate means, given its better ability to deal with multivariate environments, the famous "Monte Carlo simulations".

The article by Ulam & von Neumann (1949) is a milestone for the method already known and used since the 19th century, applied mainly to physical processes. MC consists of a statistical technique that performs successive iterations, such that, in each, random values and some deterministic parameters are used to produce a sequence (path) and a final numerical result. Applied in the context of the options market, it is a tool capable of simulating a wide range of possible economic scenarios, dealing well with uncertainty. MC also incorporates risk-neutral pricing, that is, the condition of generating prices and payoffs under a risk-neutral measure, which facilitates the management of positions when there is a deviation in values, an indication that the risk is increasing or decreasing. And it can be applied to options with complex payments with dependencies on specific market conditions at different times, overcoming limitations associated with analytical approaches.

Consider an option that depends solely on an asset S that provides a payoff at time T. Assuming a constant and risk-free interest rate, the value of the option can be obtained as follows: several random paths for S are simulated and calculated - the final payoff of each of them; The average of the payoffs is, then, calculated and the risk-free interest rate is deducted from this value. The set of paths provides a way to evaluate not only the average (or expected) value of the variable of interest, such as the option's payoff, but the probability distribution of that variable, which is good to offer new information about the variability and risk associated with the outcome.

Figure 3.5 illustrates two MC simulations and their respective resulting probability distributions. In the absence of a stronger deterministic variable, the trend remains, on average, at the origin level. Otherwise, the set of paths is directed upwards or downwards, but the symmetry constant (symmetric shift).

The path can be modeled in several ways, for example as in Hull (2003), suppose that a process followed by the underlying asset valued in a risk-free world is:

$$dS = \hat{\mu}.S.dt + \sigma.S.dz \tag{3.16}$$

where dz is a Wiener process, $\hat{\mu}$ is the expected return in a risk-neutral world and σ the volatility, so to simulate a path followed by S, the life of the option is divided into N intervals of size t in order to approximate Equation 3.9 to the following:

$$S(t+\delta t) - S(t) = \hat{\mu}.S(t)\delta t + \sigma.S(t).\epsilon.\sqrt{\delta t} \tag{3.17}$$

where *S(t)* denotes the value of *S* at time *t* and is a random variable with a normal distribution of mean 0 and standard deviation 1.

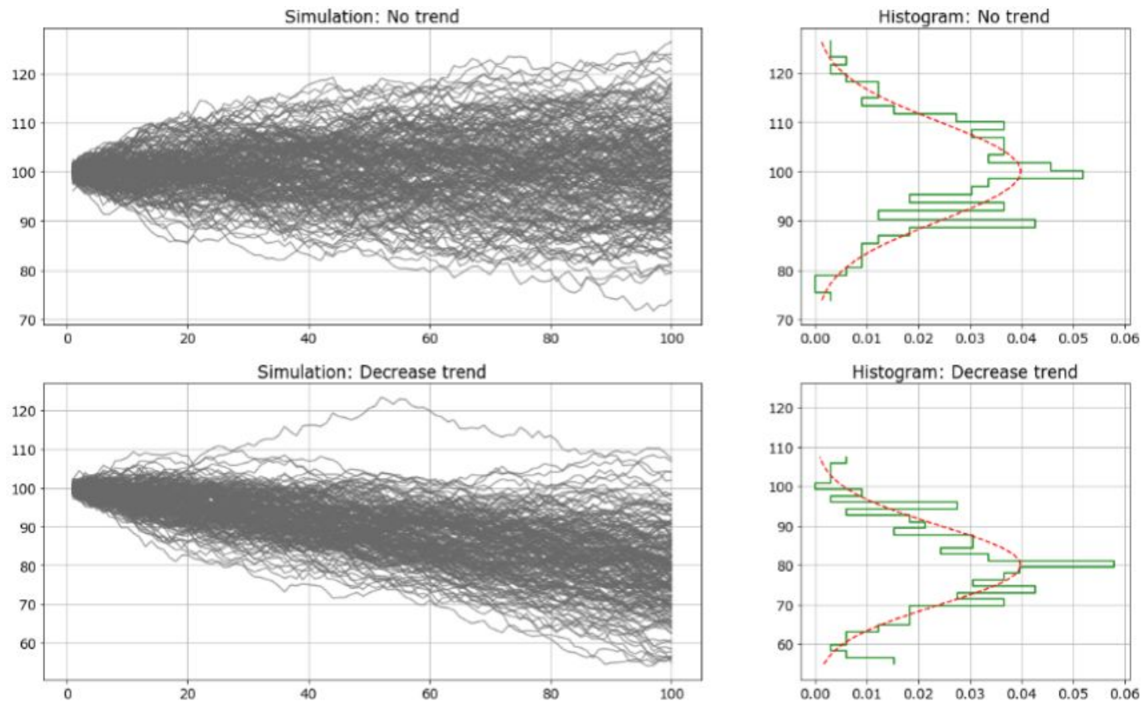


Figure 3.5: MC with the absence and presence of a deterministic variable. Source: Author's own work.

Simulations can be computed with the same parameters or not. For ML purposes, it is interesting that, with each parameter change, there is a set of paths. In this manner, with each adjustment made to the model during the simulation, there are corresponding results for evaluation - an iterative process of trial and error until an optimal price curve for the options is achieved.

One of the advantages of MC is that it deals with payoffs dependent on the path followed by the underlying asset S, unlike all previously seen methods. Payoffs can also occur at various times during the derivative lifecycle, instead of just at the end, making MC an excellent price projection tool not only for European assets, but especially for American companies. On the other hand, the main disadvantage is the expenditure of a large amount of resources, whether computational or temporal: each underlying asset is in a market that has macro and microeconomic variables with more or less effect compared to others. Simulating so many parameter variations for so many economic variables thousands of times becomes excessively costly, even though it provides loads of valuable information about the paths and final prices. Because of this, computational approaches began to be tested as soon as they were developed, whether with AI, ML or DL models.

3.2 Option Fast Pricing Modeling

The first major paper mentioning a machine learning model for option pricing is that of Malliaris & Salchenberger (1993a), who trained different neural networks and compared the results with BSMM. With different architectures and in a more exploratory analysis, the conclusion was that the neural network could achieve relatively better results than arithmetic models. Months later, the authors published a second paper, where they concluded that neural networks would be a promising alternative to BSMM particularly for options with non-linear returns or when the underlying asset presents stochastic volatility (Malliaris & Salchenberger, 1993b). Since then, DL developed a lot and brought many different ways to deal with neural networks using standard inputs.

Since then, ML has developed a lot and has brought different algorithms to deal with growing product complexity and the relation with distinct economic variables. Once the practice of pricing moves from a positive approach (starting from an equation and checking that it explains the world) to a normative approach (starting from the available data and then writing the equation) and given that ML's ability to deal with BD is well known, concern has arisen about computational inefficiencies and no longer about perfect analytical or iterative modeling. While it was possible to advance in better mathematical models with gains in pricing prediction, similar good MC predictions continued to be generated. But in the context of BD, since one simulation generates one payoff row and since many initial prices can be selected at any moment of trading evaluation, it becomes costly to create a statistically relevant database in real-time, taking hours to do it and losing trading gain opportunities.

An important advance in this methodological problem, made by Longstaff & Schwartz (2001), presents a simple yet powerful new approach for approximating the value of options by simulation. The key to this approach is using ordinary least squares to estimate the conditional expected payoff to the option holder from continuation instead of the ground truth price. In some cases, other techniques like weighted least squares, generalized least squares or generalized method of moments are more efficient in estimating the conditional expectation function. For example, if the process for the state variables has state-dependent volatility, the residuals from the regression may be heteroscedastic. Huge & Savine (2020) comment that "regardless of how it is estimated, this approach is applicable in path-dependent and multifactor situations where traditional FDM cannot be applied. Moreover, sample datasets are produced for the computation cost of one MC pricing". The authors started tests for American options, but "it worked well for Asian and Bermudian options as well. 'Sample datasets a la Longstaff-Schwartz' can also be made for European ones, simulating training sets in realistic time and allowing to learn pricing approximations in realistic time. So one of the new market practices was to learn the price curve by looking only at the expected payoff of the transaction".

Another development, made by Griewank (1992), extended the concept of Backpropagation (backprop) in a more generic way called Adjoint Differentiation (AD) and presented substantial improvements in computational efficiency compared to traditional reverse algorithms. Both AD and backprop involve the computation of gradients, but they differ in their application domains and the underlying principles applied to compute these gradients: while backprop is specific to training neural nets, AD is a more general-purpose technique applicable to a wider range of computational tasks and at the same time. Backpropagation encompasses reverse mode differentiation, while AD has both forward and reverse mode. Another difference is that backprop calculates the gradient of a scalar-valued function from $\mathbb{R}^n$ in $\mathbb{R}$ while AD calculates the Jacobian vector product of a vector-valued function from $\mathbb{R}^n$ in $\mathbb{R}^m$. These simple comparisons show how the first is a use case of the second.

The concept of Automatic Adjoint Differentiation (AAD) surges to specific financial issues, where real-time evaluations need to be made. Giles & Glasserman (2005) demonstrate the AAD application in financial engineering, particularly for computing sensitivities in stochastic control problems. Their paper highlights the relevance of AAD in derivative pricing and risk management, and shows that AAD is another specific AD case of use that applies operator overloading and template meta-programming to implement AD automatically, behind the scenes, over calculation code. From this, it follows that AAD is both a mathematical algorithm and a computer programming practice. For AD to be automatically applied in reverse order over the operations involved in a calculation, the calculation graph must be produced in memory, where all the operations must be recorded.

New techniques helped researchers and investors to deal better with complexity and to capture more relevant information with data besides a higher computation cost. "In the aftermath of the 2008 global financial crisis, massive regulations were imposed on investment banks, forcing them to conduct frequent, heavy regulatory calculations. A typical example of regulatory calculation was the Counterparty Value Adjustment (CVA), an estimation of the loss after a future default of a counterparty when the value of the sum of all transactions against that counterparty (called netting set) is positive, and, therefore, lost. The CVA is the value of a real option a bank gives away whenever it trades with a defaultable counterparty" (Savine, 2019).

Similarly to option pricing, the vast cashflow numbers and the high dimension of the simulation make CVA calculation take hours on a large net. So a similar problem of high demand for resources and the need to compute data quickly forced investment banks' quantitative research, say Savine (2019), to actively develop new techniques to deal with sensitivities to thousands of market variables that need to be repeated thousands of times with inputs bumped one by one, like happens in MC simulations. The technologies described in Giles & Glasserman (2005), Capriotti (2008), Savine (2014), Capriotti *et al.* (2015), Flyger *et al.* (2015), Huge & Savine (2017) and Andreasen & Savine (2018), respectively, AAD, parallel simulations, cashflows scripting, regression proxies, model hierarchies, how to bring them all together to better risk manage derivatives and xVA are all model agnostic: they are designed to work with all models.

For the construction and objectives proposed by Diff ML, AAD is the most important of the fast processing techniques described above: "Greeks have traditionally been calculated by making small adjustments to the values of the inputs in the pricing of a derivative and calculating the output value each time, a process known as 'bumping'. This can be time-consuming for a portfolio of thousands of trades because the valuation of each trade will involve many steps, each requiring the output from the previous step to proceed. AAD breaks this valuation process into several steps that can be carried out simultaneously instead of sequentially. This is made possible by exploiting a key mathematical property that applies to sensitivities called the chain rule of differentiation, which links the derivatives of parts of a function to the derivative of the whole. This allows the backward propagation of sensitivities of the output wrt the variables in the intermediate steps, until the sensitivities wrt the inputs are achieved" (Risk Glossary, 2022).

In summary, recently, instead of calculating the expected value of the payoff for each initial price resulting from MC or instead of running a simple ML model on payoff related to those initial states, the strategy is as follows: MC dataset (compound mainly by initial price feature and payoff label) is run in a simple ML model and AAD computes information about derivatives of the output wrt the input. These derivatives are added as a new label of the MC dataset, which is trained and tested in a ML model with a more complete loss function to discover the option pricing function. Diff ML modeling is a brand new fast technology capable of dealing with a high dimensional space. Huge & Savine's (2020) proposal is to invest time in learning computing that is not only fast and saves time with MC simulations, but also generates a methodological advance within the field of Computer Science: "even when training enormous amounts of data resulting from MC, there continued to be limitations when using standard ML, especially in supervised learning, namely: training on noisy payoffs is prone to overfitting, and unrealistic dataset sizes are necessary even in the presence of classic regularization and, in addition, risk sensitivities converge considerably slower than values and often remain too approximate even with training sets in the hundreds of thousands of examples".

For an option pricing context, a Diff ML model will learn the pricing function which approximates the expected value of the payoff, given two elements: (i) the initial market conditions (for example, just the current price in the present); and (ii) the type of option in question (for example, if European or a Bermudan, because each one defines a specific payoff calculation). In one hand, market condition X_t^M for a time t is given by an analytical model (like BSMM or a simple Random Walk) from an initial state X_0^M and a group of weights w_i. In the other hand, specific characteristics X_t^I of an option type are united with market conditions in a compound function to generate the so-called "pathwise payoff" h in Figure 3.6.

Two examples are provided by Savine (2021). The first example is a European call (derivative instrument) under BSMM (pricing model):

- European: $y = (x_T - K)^+$

- BSMM: $x_t = f(x;w) = x.exp[-\frac{\sigma^2}{2} + \sigma.\sqrt{T}.N^{-1}(w)]$

- Pathwise payoff: $x_t = f(x;w) = (x.exp[-\frac{\sigma^2}{2} + \sigma.\sqrt{T}.N^{-1}(w)] - K)^+$

The second example is a barrier call (instrument) under BSMM (analytical equation):

- Barrier: $y = g(alive[x_{T0}; x_{T1}; x_{T2}; ...; x_{Tp}]) = alive.1_{max[x_{T0};x_{T1};x_{T2};...;x_{Tp}]}.(x_T - K)^+$

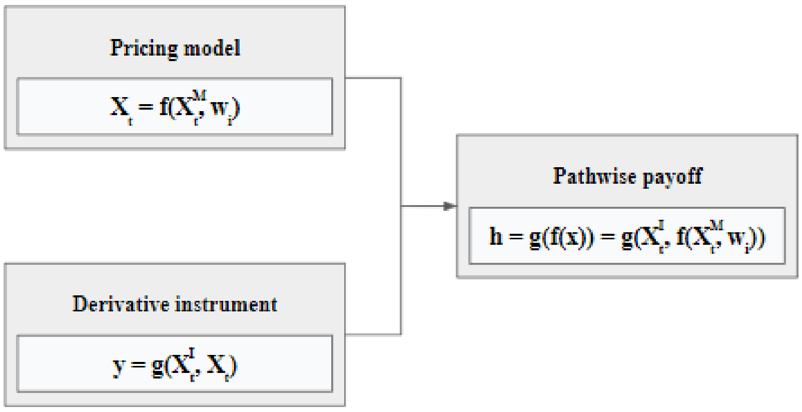


Figure 3.6: Pathwise payoffs formation.
Source: Author's own work, based on Savine (2021).

- BSMM: $x_t = f(x; w) = [x_0; x_{T1}; x_{T2}; ...; x_{Tt}]$, where $x_0 = x$ and

$x_{Tt+1} = x_{Tt}.exp[-\frac{\sigma^2}{2}.(T_{t-1} - T_t) + \sigma.\sqrt{(T_{t-1} - T_t}.N^{-1}(w_t)]$

- Pathwise payoff: $h(x; w) = g(alive[x_{T0}; x_{T1}; x_{T2}; ...; x_{Tp}]) =$

$= alive.1_{max[x_{T0};x_{T1};x_{T2};...;x_{Tp}]}.(x_T - K)^+$, where $x_0 = x$ and

$x_{Tt+1} = x_{Tt}.exp[-\frac{\sigma^2}{2}.(T_{t-1} - T_t) + \sigma.\sqrt{(T_{t-1} - T_t}.N^{-1}(w_t)]$

Many examples can outline $h(x; w)$, like a Asian put under SABR or a American call under Heston. Since to describe the option pricing function a Diff ML model needs, in the limit, just the current price of an underlying asset and a model to describe the future possible paths, it is possible to work with an agnostic dataset valid for any theoretical model and instrument.

The dataset is, initially, compounded by many values of some initial states X_0^M (features) and N pathwises h for each initial state generated by a MC (label). Derivatives are extracted from this dataset with the help of AAD in a simple ML model to add differential values as a new label. Savine (2021) follows the line of reasoning by presenting an example that uses the Liquid State Model (LSM) and another using a NN to forecast the option price curve. Figure 3.7 illustrates the general scheme by which it is possible to carry out the pricing activity quickly and agnostically.

The choice of Diff ML as the algorithm for finding $\alpha(x; \theta)$ is a way of speeding up pricing because, as noted in section 2.2, the use of differentials mapping the direction of the function around the points rather than trying to read all the points and fit them accelerate the computational process. Instead of using Diff ML, it could be a standard ML model, like the LSM model presented by Longstaff & Schwartz (2001) or a DL model as suggested by Lapeyre & Lelong (2021), but the process would not be so fast.

In terms of time spent, there is a problem when not using a Diff ML model. Another problem is the convergence process of values, or the size of the error in other words. Savine (2001), in its masterclass, presents four different models and tools to deal with the increasing complexity of pricing problems. Starting with polynomial models, for the same dimension, they offer better results with the increase of observations (in his examples, from 1024 to 4096), but when increasing dimension, they need even more data, so they are appropriate for low dimension and with lots of data. However, better results appear with regularization even in a dimension until 10, but with the cost of introducing bias. To solve these problems, NNs were tested for a correlated Gaussian basket of underlying assets and both pricing prediction and risk sensitivities performed well in a 30-dimension problem, but with the continuous need for more observations (in his examples, from 16384 to 131072). Putting in economic terms and in my own words, at least, the dimension supply grows faster than the observation demand and this represents some "inflationary modeling process".

The master class continues with a third example of a model, a differentiation polynomial regression, that has a sharp improvement over regularized polynomial regression with a smaller number of observations and even without cross-validation, but it still loses performance in higher dimension problems. By applying Diff PCA, it maintains just the necessary relevant dimensions to be regressed, so for a larger dimension preprocessed problem, for example 500, few data need to have an almost perfect prediction (in his example, 1024). Finally, a fourth model and the best model tested was the one present in subsection 2.2, the Diff NN in the shape of the twin network, and its results are in Figure 2.4.

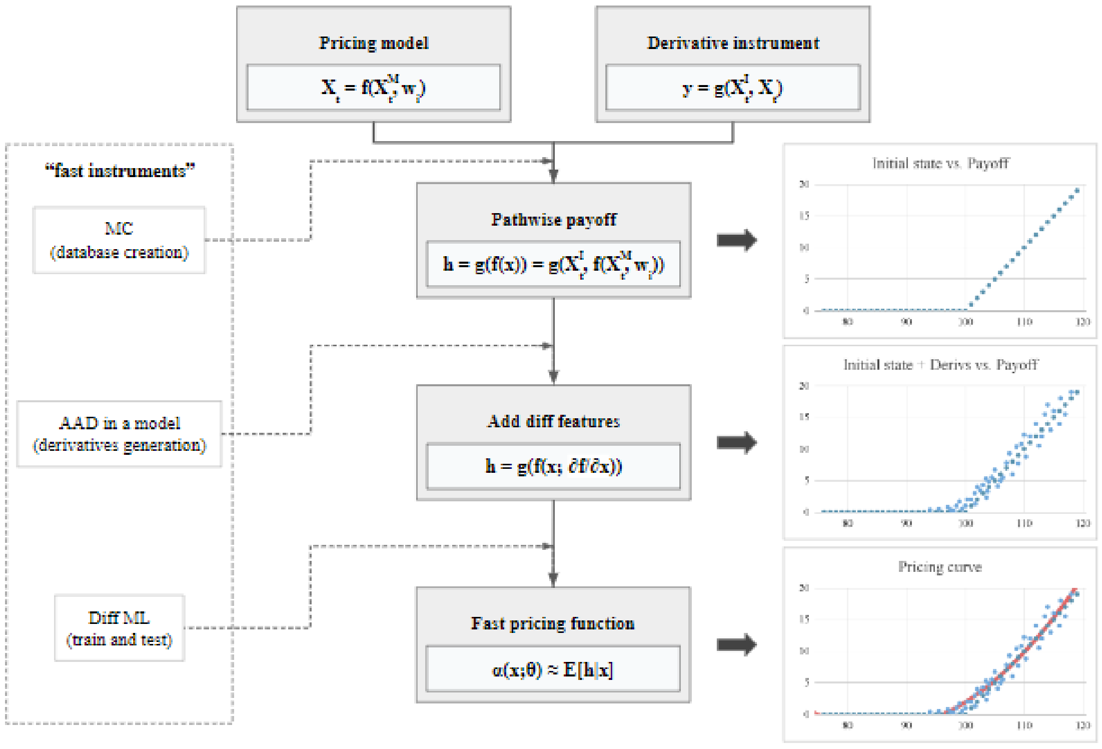


Figure 3.7: Fast pricing function formation.
Source: Author's own work, based on Savine (2021).

3.3 Differential Machine Learning applications

Although Diff ML is a recent practice within the ML universe, there are papers applying differential techniques inside and outside the Finance area. After all, as it is a methodological evolution, it can and should be tested in new areas of knowledge to validate the technique and test its robustness cross-application. Therefore, this subsection brings a first block of applications within the financial area, precisely all related to the option pricing problem, and a second not related to the world of Economics.

In addition to the contribution of Huge & Savine (2020) and other materials created by them, this thesis is the ninth study to deal with the subject and it reinforces the benefits of this fast pricing prediction technique. Below is a list of the eight papers already written, four applied to option pricing and four in other areas of Data Science.

3.3.1 Option Pricing Applications

Frandsen *et al*. (2021) "show how and why to use a financially meaningful differential regularization method when pricing options by Monte Carlo simulation, be that in polynomial regression or neural network context". To do this, they add the derivative of the objective function to the least squares regression and weight the portions corresponding to f and f′. Adding terms to the regularization function is common, but with the need to balance bias and variance. In the so-called "delta-regulation", the authors comment that the choice of weight to weight the elements f e f′ is not really crucial, since the errors observed wrt a standard neural network are better for different weight values.

Rasmussen & Buschardt (2022) reproduce the network created by Huge & Savine (2020a) that takes a variety of European calls in BSMM and compares it with the standard neural network, with a linear regression model and a regularized linear model. The authors demonstrated that polynomial models had similar results to neural networks when using polynomials of order seven. They also confirmed the superiority of Diff ML for both price prediction and deltas (a change in the price of an option in relation to a change in the price of the underlying asset). Furthermore, a differential model was tested for American puts, requiring a larger quantity of observations compared to European calls.

Goldin (2023) performs many exercises to check that Diff NNs provide much better accuracy and convergence rate than feed forward ones. For example, it was shown that sensitivities of Asian options in the case of volatility curves are estimated with good accuracy. Both options written on a basket of correlated stocks and those dependent on the LIBOR interest rate showed good performance in all its greeks. Also, it was empirically tested that the higher the order of volatility, the greater the advantage of Diff ML for any class of assets.

Similarly to this report, Simón i Ribas (2023) recapitulates concepts of the options market, ML and Diff ML and based on the article by Huge & Savine (2020), the author reproduces the price calculations using BSMM, but advances with tests on the stochastic volatility HM. She reinforces the superiority of differentiated networks over traditional ones, both in price and in Greek and despite the author "was not able to introduce more than one derivative at a time, that arises the possibility of creating a "tagging" function which would allow the model to know wrt which parameter the pathwise differential is being calculated".

3.3.2 Studies in Different Areas

Alahassa & Murua (2021) exclusively address the topic of Diff ML as a methodology capable of generalizing the multivariate interpolation problem. To this end, they comment on three large groups of differentiation methods: (i) the automatic, to automatically construct a procedure and numerically evaluate the derivative of a function, given the analytical expression of the function itself; (ii) symbolic, when the solution is the function derived from the original function; and (iii) numerical, when the derivative of a function is represented by calculating the limits around a given point. From different methods observations, the authors explore Taylor Series, which are one manner of explaining a function through an infinite series of terms. In the context of Diff ML, this is crucial as many machine learning algorithms rely on calculating derivatives wrt parameters for optimization and model tuning. Hence the authors comment that "this is a great revolution in Statistical Learning Theory, as we can sufficiently (i.e $h \geq 2$) interpolate any machine learning regression problem, with the reduced differentiation form when we know the perfect parameters that fit the training data".

In non-methodological problems, the first work is from Abegaz (2022), that studies an unsupervised machine learning-based control for autonomous cars. The approach incorporates inputs such as direction, velocity, and driver torque into electrical power-assisted steering. Furthermore, with the addition of differential relations between variables as inputs, it was seen to improve the overall efficiency of vehicles by 50% by generating steering related outputs (angular velocity, angular difference, output torque).

Abadeh *et al.* (2022) proposes an efficient collaborative filtering method for recommendation systems by using supervised Diff ML based on twin networks and integrating the cost of derivatives and errors in values, a process which increases remarkably accuracy. The so-called Differential Collaborative Filtering was implemented on three different training recommendation system networks and proved increasing training accuracy utilizing twin networks.

Abegaz & Mirzaie (2023) study "signed network", a way of evaluating the degree of trust in relationships between people on social networks based not only on negative interactions, but also positive ones between pairs of users. They comment that "traditional approaches for computing trust values in social networks are often based on the formula and take the advantages of being simple and quick, but they really do not adapt to different types of social networks". So "the introduction of delta value in differential labels leads to increased training accuracy according to the functions of the forward and backward training".

3.4 Summary

Investments in stock exchanges are a centuries-old practice, just like derivatives trading. Options are more recent compared, for example, to bonds and stocks, but its theoretical research history is very notable, concerned with modeling how the prices behave in order to optimize portfolios and increase the payoff of transactions. This report categorized them into three main classes: analytical, iterative and computational.

Analytical models are categorized into diffusion, jump and jump-diffusion models. Starting with Bachelier's, culminating in the Black-Scholes-Merton model and after by adding stochastic processes to capture the unpredictable nature of economic variables. Some parameters can be considered constant or not, and the temporal dimension can be continuous or discrete. Over the decades, more economic variables could be added and modeled in increasingly complex ways, capturing how each of them impacts the final price of the options.

Iterative models are exemplified mainly by binomial trees, finite difference methods and Monte Carlo simulations. It models options in a discrete way, but it can be approximated continuously with a large quantity of steps. However, given the increase in the universe of expected values at each given step, they cannot deal with multiple variables, which can be a difficulty when, in the real world observed in Economics and Finance, each new information requires adjustments in the decisions of investment investors. On the other hand, they are too fast and often produce sufficient auxiliary statistical results.

Then, the computation power gained prominence in the 1990s, providing versatile tools for multivariate environments modeling. The integration of artificial intelligence, particularly neural networks, has been explored for option pricing, with better performance compared to traditional models. In addition, recent advancements in Diff ML aim to address challenges associated with noisy payoffs and small datasets, leveraging differential information for more efficient model training.

Diff ML is part of a new era when efficiency is a rule, i.e., to do more with less. The paper authored by Huge & Savine (2020a) marks the beginning of a literature with such potential that, ultimately, all conventional ML models previously employed in the literature could be re-evaluated against a differential version, not only in Finance but also in Economics and various other domains of human knowledge. There is still little work done, as it takes time for a technology [in its most basic sense] to be publicized, tested and approved to be considered an established paradigm, not only with potential, but with demand for. Still, there are some applications not only for options pricing, but also in classic Data Science problems, such as recommendation systems, autonomous cars and social network evaluations.

Chapter 4

Methodology

Diff ML uses the differentials of the training labels wrt the input variables, resulting in an error reduction on relatively small datasets, thus reducing the time it takes to train the models. The result is an alternative for many data problems, especially for those with high dimensions that suffer from the problems of the curse of dimensionality, like MC, and those where datasets are small, where it would be necessary to create representative synthetics to improve the quality of the model. Furthermore, "contrary to classic linear models, neural networks don't regress on a fixed set of basis functions. They learn from data a relevant regression basis in their hidden layers, embedding a powerful dimension reduction capability in their structure" (Huge & Savine, 2021c).

The facts presented above raises some questions: How to construct a code capable of computing the derivatives, selecting the most relevant and using them in the loss function to make better predictions? Is there any good library that can introduce more flexibility during the code-building process? Once the code is done and once is well known Diff ML has better results than a standard ML model, which elements of the differential neural network help to achieve the desirable results in a supervised regression problem like pricing prediction?

This chapter answers these questions by presenting the technical work, divided into four subsections: (i) the code translation task from TensorFlow to PyTorch, justifying why this library was chosen and detailing the main classes and methods that make up the main code, as well as the final result of this part of the work; (ii) the training exploration task of making small changes to the code and only to the architectural components of the Diff NN to improve training; (iii) the evaluation of each architecture to confirm the best components and to check how the prediction behave wrt each change; and (iv) the validation of the best architectures found previously.

4.1 Code Translation

4.1.1 Using the PyTorch Library

Huge & Savine (2021a) code shows how to build a Diff ML model by creating a feed-forward network and doing a backprop on it to compute the derivatives of the output wrt the inputs. They compare the performance of this new model to the standard one, where the loss does not receive differential information, and plot results for data generated from a MC simulation that is supported in one example by a Bachelier model and in a second example by a BSMM. Huge & Savine (2021d) code builds a Diff PCA class to select the variables that present the biggest derivatives wrt the label, i.e., the variables that present biggest relevances to explain the payoff. In both cases and other demonstrations, the authors use the TensorFlow Machine Learning library. This library was created in 2015 and brought many features and simplification through methods that previously needed to be done manually by a developer.

PyTorch has been another fast-growing machine learning library, because it uses dynamic computation graphs (also known as define-by-run), i.e., graphs are built while operations are executed, making debugging and development tasks more intuitive and flexible compared to TensorFlow (Paszke *et al.*, 2019). It also brought more functionalities and methods to simplify coding work, for example, the computation of the derivatives from one tensor wrt another. In Huge & Savine (2021a), there are a couple of methods to execute the backprop, while PyTorch does it in just two lines as will be presented in the next subsubsection.

Another advantage of PyTorch in comparison to TensorFlow is that it has a cleaner syntax similar to conventional Python in data analysis, helping in the interpretation of third-party code, integrates seamlessly with other popular Python libraries, such as NumPy and Matplotlib, making it easier to use in a broader Data Science context, and it has a growing research and programming community with extensive reference documentation.

4.1.2 Code Main Artifacts

The code for this project is composed of a sequence of steps, from building the dataset to deciding on the model with the best performance for the regression problem of rapid pricing of options. This sequence consists of four mini-tasks: (i) data generation, which depends on the user's choice of, on the one hand, a theoretical model used to perform MC simulation and compute ST and, on the other, the type of option to compute the payoff resulting from each pathwise; (ii) computation of the derivatives of each payoff wrt the initial price of the underlying asset during the model training; (iii) Diff ML model training for a set of component parameters, parameters, hyperparameters and the usage or not of derivatives computed previously to calculate the loss; and (iv) demonstrations that the architecture used in Huge & Savine (2020) models have the same performance, confirming the superiority of differential learning over the standard.

The data generation mini-task follows the logic explained in section 3.2, so that it will be generated from scratch and automatically from: (i) an analytical equation such as a Random Walk, Bachelier's model or BSMM; and (ii) the type of option, be it a put, a call, American, European or other.

To recap, the models used for the analytical equations are:

- Bachelier: $S_t = S_{t-1} + \epsilon_t$
- BSMM: $S_t = S_{t-1}.exp\left(\frac{\sigma^2}{2}.(T_t - T_{t-1}) + \sigma.(W_t - W_{t-1})\right)$

And the payoff (π) equations, for a stock price at maturity S_T and its strike K:

- European call: $\pi^c = S_T - K^c$
- European put: $\pi^p = S_T - K^p$

With the analytical equation, a MC simulation is performed for a set of possible underlying asset current prices. N initial values are chosen randomly and in large numbers, each one put in a MC simulation with M pathwises, then generating n.m final prices in the maturity. With the option type, $N.M$ final prices are converted into $N.N$ payoffs. Huge & Savine (2021a) present an example for a Bachelier model in an N-dimension problem, where N is the number of stocks correlated to the underlying asset and an example for BSMM in a 1-dimension scenario, both for a European call. This data generation happens inside a parent class and its children, each one for each analytical equation. During this task, a BSMM N-dimension scenario, the European put instrument and an M-dimension problem for M the number of option baskets were also developed.

The derivative computation mini-task consists of applying some parameters and methods in the input and the output during the training of the ML model. Before training, the input tensor needs to be configured to generate derivatives when they are called, so it needs to have the parameter *requires_grad_* set as True. Immediately after training, the output tensor is aggregated around the sum and some computational graphs are created to calculate the derivative between the explicit values of input and aggregated output. To execute this process, method *autograd.grad()* is used as following:

Listing 4.1: Simple Python code

```
input = X.requires_grad_(True)
diff_input = dYdX
output = self.forward(input)
diff_output = autograd.grad(output.sum(),
                            input,
                            retain_graph=True,
                            create_graph=True,
                            allow_unused=True)[0]
```

The *.sum()* operation aggregates the output tensor into a single scalar, allowing PyTorch to compute the gradient wrt input. This is necessary because gradients are defined for scalar outputs, ensuring consistent and correct backprop. Using *.sum()* instead of *.mean()* in gradient computation is often preferred because the first preserves the scale of the gradients, which is important for correctly updating the model's parameters during optimization. By applying True in all three parameters *retain_graph*, *create_graph* and *allow_unused*, it is possible to, respectively, perform new backward passes on the same computational graph, create new graphs during the backward pass if it is necessary and execute the passes even if inputs are not being used to generate outputs.

In the original TensorFlow code, Huge & Savine (2002) created a couple of methods to do the gradient calculation, but in PyTorch it takes place over just one single line, since the *requires_grad_* parameter is passed when the input object is created.

Model training mini-task is executed by the authors with the help of epochs and batching data, so the loss is calculated during the computation of the gradients for each batch in each epoch. They use Xavier/Glorot weights initialization, Softplus activation function and Adam optimizer with a non-constant learning rate. The scheduler technique is used in the learning rate by initializing it in a low value, increasing fast before half the epoch number and cooling down until the end of the training. All these model instances are unchanged in the translation main task to achieve the same results observed in the inaugural paper. The training happens inside a second class, composed of internal methods, each one dedicated to a different part of the training, i.e., weights initialization, parameters definition, hyperparameters calculation, and training and testing themselves.

The last mini-task is the demonstration that in PyTorch, as well as on TensorFlow, differential learning had lower values of RMSE than standard learning. The proof happens in a final method responsible also for visualizing both models.

After all mini-tasks, Figure 1.1 shows a summary of the code. For a given Diff NN architecture and a given MC simulation, there are four possible use cases for each dimension (each correlated variable and basket options number):

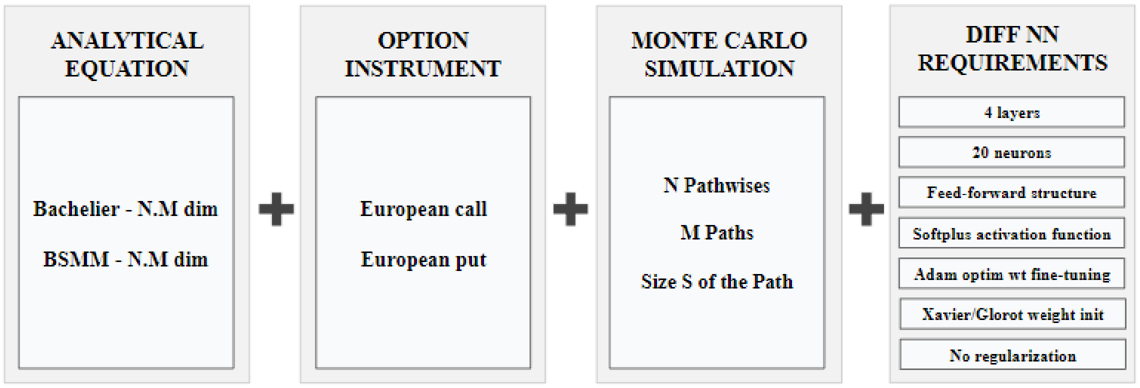


Figure 4.1: Elements of the code after translation main task.
Source: Author's own work.

The preprocessing step of applying Diff PCA to select the most relevant variables for training, which would be the fourth in five steps, was not applied by a restriction of time. An initial translation was done with no coding error, but the results were not expected. Therefore, the decision not to carry out this part was made to ensure the following exploratory work on the different architectures. This mini-task would happen inside a class that computes an eigenvalue decomposition, selecting those with the biggest values and deciding how many components are necessary to explain most of the variation of the label wrt the inputs in terms of derivatives. Two auxiliary methods would complement the class, one by doing the transformation to project inputs and differential labels from Euclidean space to the reduced space defined only by the first L singular vectors, and the other to reverse the previous transformation. Huge & Savine (2021d) presents its class for an *N.M*-dimension problem, where *N* is the number of correlated stocks and *M* is the number of option baskets.

4.2 Training Exploration

Considering there are *N* different analytical equations, *M* different option instruments and *P* possible MC, there is a multidimensional problem of programing *M.N.P* isolated models. Given a MC structure, once an implementation can be made from one analytical equation and one option instrument, data generation for two or more of them will be a simple code job later. With the correct compartmentalization of the main elements, the agnostic nature of the implementation is guaranteed.

This simplification means that the focus is on the possible models for each set of Diff NN requirements, i.e., for a given architecture. For this reason, the goal of the second main task is to improve the translated Diff ML model by exploring different architectures. Once the refinement of the model does not depend on the analytical equation, the option instrument and the MC simulation, Figure 4.2 presents the initial plan for different possible architectures.

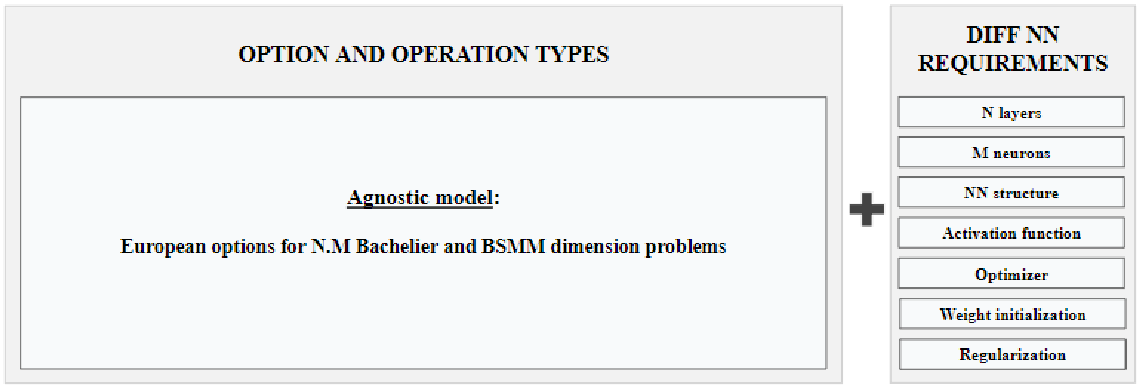


Figure 4.2: Focus on Diff NN requirements in the second main task.
Source: Author's own work.

A neural network has several elements. It is possible to create several versions of a supervised model, each with its hyperparameters, number of hidden layers, depth of each layer, weights initialization, activation functions, optimizers and regularization techniques. In other words, there are many specifications to test.

During the practical work, two of the Figure 4.2 elements were not changed: NN structure and regularization. NN structure could be a recurrent neural network depending on the data time series horizon, but because data can be treated as static, the choice was to keep it a feedforward network following the idea of the twin network presented in 2.2. And most common regularization techniques as L1 (Lasso), L2 (Ridge) or Elastic Net could be applied too, however, as seen in subsection 2.2, adding a differential component in the loss already acts as a regularization solution. Any dropout regularization addition generated bad results all the time, both for differential and standard learning, so it was easily discarded.

On the other hand, four new parameters were introduced as well: the number of epochs, the number of mini-batches, the batch size and the learning rate schema. So Figure 4.3 presents the final elements that have been changed:

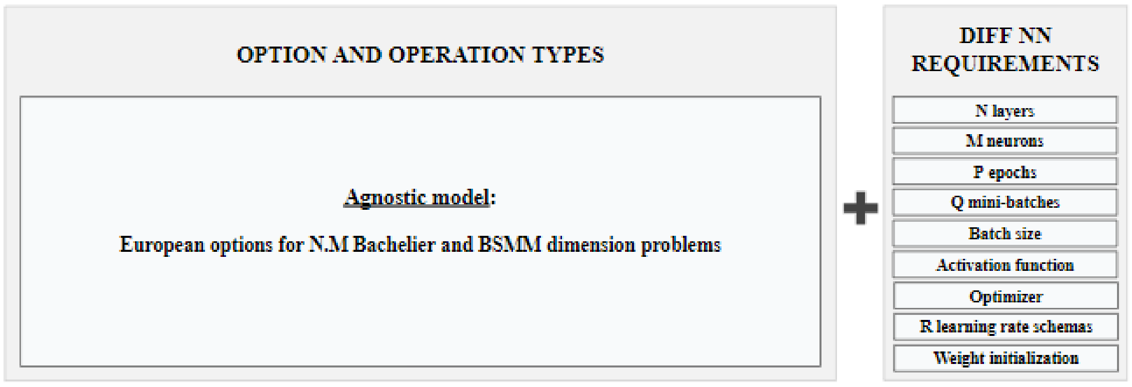


Figure 4.3: Tested Diff NN requirements in the second main task.
Source: Author's own work.

Related to the number P of epochs, during the translation task, 200 epochs were introduced to help understand the behavior of differential learning and then placed in comparison to the initial value of 100.

About the batching data, the authors originally used a different number of observations for each mini-batch, namely, 256 in 8 batches for 1024 size and 512 in 16 batches for 8192 size. This difference in training can lead to divergences in the updating of parameters and a different risk of overfitting with the introduction of more or less noise, since partitioning the data also acts as a form of regularization. So a first comparison was made using 16 batches and a second using 256 observations for both sizes, an attempt to create a good basis for comparison.

Between possible activation functions, they might not be Sigmoid and its variations, because option pricing has a regression nature and not a classification nature. They must be ReLU and its variations: Leaky ReLU with different slopes, ELU with different parameters, Softplus and Softmax.

Finally, among optimizers, tests were executed with first-order, adaptive learning rate, momentum-based and combination methods, namely, SGD, RMSProp, Adam, Adamax, Adagrad and Adadelta.

Since working with different optimizers, the level of the learning rate is a possibility to change. Huge & Savine (2020) don't use a not constant learning rate, but a schema that starts with a "learning rate warmup" followed by a "learning rate decay" along epochs. So between different constant rates and different learning values for different epoch percentages, there are R possible schemas. In PyTorch, *torch.optim.lr_scheduler* module is the library used to provide classes and functions to adjust the learning rate during training.

Weight initialization could be maintained the same because, contrary to classic regression, training a neural network is a nonconvex problem, hence, its result is sensitive to the starting point. Xavier/Glorot initialization is based on the implicit assumption that the units in the network, including inputs, are centered and orthonormal (Huge & Savine, 2020c). The article continues: continues: "it therefore performs best when the inputs are at the very least normalized by mean and standard deviation, and ideally orthogonal. This is specific to neural networks. Training classic regression models, analytically or numerically, is a convex problem, so there is no need to normalize inputs or seed weights in a particular manner". Despite that, the initialization was changed just to check its performance compared to others.

4.3 Evaluation Metrics

The nature of the option pricing problem is inherently a regression task, where the goal is to predict the price of an option based on various underlying factors such as the current stock price, volatility and time to expiration. So RMSE metric is the basis for all the evaluation process, which will be divided into two stages. The first stage corresponds to obtaining the metrics of interest for each of the architectures individually. The second, to compare between architectures based on other criteria to be explained below.

In the first stage, for each architecture, an evaluation will be made not only of Diff NN, but of the standard one as well to reinforce in this research the predictive superiority of the first over the second. To achieve this, some metrics will be used throughout the application: three of them are related to the quality of the prediction and one related to the time some of the architectures take to process results: (i) RMSE, a gross metric; (ii) advantage differential learning has over standard (*adv*), a relative metric that compares RMSE in differential learning and RMSE in standard learning; (iii) aggregated architecture performance taking into account RSME and adv (*arch_adv*), a proposed metric; and (iv) process time.

RMSE computes the size of the error of the predicted values wrt the target value. By way of demonstration, the errors of a Diff NN are smaller than those of a standard NN, which results in predicted values much closer to their real value even in a high-dimensional environment (Huge & Savine, 2021b).

RSME is calculated as:

$$RMSE = \sqrt{\sum_{i=1}^{n} i \frac{(y_i - \hat{y}_i)^2}{n}} \tag{4.1}$$

where y is the value of the observed payoff and $\hat{y}$ the payoff prediction.

The models are compared not just in absolute terms with RSME, but in a comparative term as well. For that, a formula is used that measures the advantage that differential learning has for every j number of observations in comparison with standard learning. If negative then the differential presents a better prediction.

$$adv = \frac{RMSE_j^{diff}}{RMSE_j^{stdd}} - 1 \tag{4.2}$$

It is good to have a relative measure because sometimes absolute values are bigger or lower, so it is possible to know the size of the value differential learning aggregates in predictions.

These two metrics are good when making unique comparisons between performances, but for a group of learnings, especially in an activity where it's necessary to compare the effect of different architectures and the divergence predictions that some of them can produce, some aggregation is welcome.

To compare different architectures, instead of just using an aggregation of measures, like the mean, we propose, for a group of evaluations, a metric to evaluate the real advantage of a group of k learnings in that architecture (differentials and their standard comparisons). For that, we take into consideration the mean of the gross RMSE values of the k learnings and the mean advantage that differential has over the standard in the same k learnings:

$$arch\_adv = \left[\sum_{i=1}^{k} i \frac{RMSE_k}{k} . \left(1 + \sum_{i=1}^{k} i \frac{adv_k}{k} \right) \right] \tag{4.3}$$

The so-called architecture advantage metric presented above takes into consideration not just the gross values, but also the percentual comparison values. The lower *adv*, the lower *arch_adv* and if the mean RMSE is low, then a good architecture is found because it generates predictions as similar as possible to the original values and is much better than the standard ML models.

arch_adv metric does not replace RSME and *adv*, because it is possible to have the same value for a scenario where the RMSE is low and the advantage is not so big and for a scenario where RSME is high, but the advantage is high.

To finish the first evaluation stage, time will be measured for specific architecture changes, like the number of layers, neurons and epochs, changes whose impact is clear in terms of time processing. Sometimes, the results get better after changing some of these components, but if the loss in time is greater than the gains in prediction, maybe the change is not positive.

In the second stage, the interest is in comparing different architectures. For this, two criteria will be used to complement the *arch_adv* values (C_1 - criteria 1): (i) the number of times differential learning is, on average, better than standard (C_2 - criteria 2); and (ii) number of times differential learning is better than standard for each stock number for each analytical equation for each instrument (C_3 - criteria 3). This last criteria can be summarized by the C_2, but it is presented as an extra metric.

Finally, R^2 could be a useful measure for a regression problem. Still, since the performance of the differential neural network is superior to that of the standard network and since R^2 values have been tested for different architectures without much variation, this metric will therefore not be considered in the training and validation of the network.

4.4 Validation

After completing the two stages of evaluating RMSE metrics for all architectures and identifying the best-performing models, the next step is to validate their results. In this validation phase, rather than generating new data, the same dataset is utilized with different seed values to ensure robustness. Huge & Savine (2020) employed the seed value "1234" to test all architectures, which served as a standard benchmark for comparison.

To rigorously evaluate the stability and generalization of the best architectures, the numerical sequence was systematically extended by continuing the sequence forward to "0" and backward to “1”: specifically, the seed values "5678," "9009," "8765," and "4321" were used. This approach of altering the seed values while keeping the underlying data constant ensures that the validation process thoroughly tests models' performance under varied random conditions without introducing additional complexity or noise.

The decision to use the same data for different seeds offers several advantages. It guarantees consistency in the validation process while minimizing computational resource consumption, as only a single variable (the seed value) is reset. This method also provides a controlled environment for assessing how sensitive the model is to different initialization states, which is crucial for confirming the reliability of the results.

Finally, the metrics used in this evaluation part are the same presented in subsection 4.3: three metrics for the first stage (RMSE, *adv* and *arch_adv*) and three criteria for the second stage (*arch_adv*, C_2 and C_3).

Chapter 5

Experimental Study

This chapter focuses on describing the architectural changes made in the main exploration task. Once the translation has been completed, it is time to list the parameters that can be changed and carry out a *ceteris paribus* analysis of the RSME values obtained (by *ceteris paribus*, we mean do one change at a time). This means that no change was made to two or more parameters at the same time, since any such initiative needs to be completed in order for the comparative analysis to be complete.

From the description made about the multidimensional problem in subsection 4.2, the number of possible architectures is *N.M.P.Q.R.5.6.7*, once 5 activation functions, 6 optimizers and 7 weight initializations have no changes in its hyperparameters. The problem therefore has an even higher dimensionality, the resolution of which will be discussed in the final section of this thesis.

In the first part of this chapter, improvements made during the translation phase will be presented, such as some changes to hyperparameters already at this point and also the comparison between differential learning and standard learning for baskets of options with a number greater than 20, advances which already improve the understanding of the problem compared to the Huge & Savine (2020) original paper.

The second part contains a more detailed list of the parameters changed in relation to section 4.2, as well as the values chosen and their justification. The third part assembles all the architectures and shows its results in terms of RMSE and time as discussed in 4.3. The final section discusses the results obtained, as well as the strengths and opportunities found. Graphs are used in subsections 5.1 and 5.3 to help visualize the results and their dimensions. More graphs are found in Appendix B.

5.1 Improvements During Translation

5.1.1 Improvements in the Code

During the translation phase, three code adjustments were made before the exploratory phase: (i) using 200 epochs instead of 100; (ii) changing the beta parameter of the Softplus activation function; and (iii) not taking into account the learning rate "warmup" scheme proposed by the authors.

By using 200 epochs, the aim was to see the evolution of the hyperparameter values while no major problems were found in the translation. At the end of the translation, they were considered part of the final architecture before exploration.

Softplus activation function has a “beta” parameter that is responsible for making the function a smoothed version of ReLU around zero and by default, beta is equal to 1. This feature is one of the biggest advantages of Softplus compared to ReLU, because it is differentiated in all values of its domain.

Listing 5.1: Simple Python code

```
activation = nn.Softplus(beta=f)
```

If beta is close to zero, Softplus is similar to ReLU and vice-versa. Changing the beta parameter helped to improve RMSE values. The observed rule was that when increasing the number of stocks in a basket, it was interesting to reduce beta value. ReLU is piecewise linear, activating the neuron by passing through positive values unchanged while setting negative values to zero, promoting sparsity by creating sparse activations, which can be beneficial for high-dimensional data. Softplus tends to be less sparse, providing smoother gradients which can be advantageous for low-dimension tasks.

Another consideration is that in higher dimensions, predictions become more variable and the introduction of simple non-linearities helps in the stabilization of predictions. On the other hand, in low-dimensional environments, predictions are very stable, and introducing smooth non-linearities helps in capturing new relations. This balance showed a good practice for predicting payoffs around zero. f function used to model will be presented in subsection 5.3 and because it was improved yet during the translation task, they were also considered as part of the final architecture before exploration.

The learning rate warm-up schema was not considered a translation error. In a debugging task, it was realized that the learning rate to be used until that moment was constant and equal to 0.001, the Adam optimizer default value. Two lines of code were then added:

Listing 5.2: Simple Python code

```
for param_group in optimizer.param_groups:
    param_group['lr'] = learning_rate$
```

which allow the application of a previously defined learning rate scheme:

Listing 5.3: Simple Python code

```
lr_schedule = ((0.0, 1e-8), (0.2, 0.1), (0.6, 0.01), (0.9, 1e-6),
               (1.0, 1e-8))
lr_schedule_epochs, lr_schedule_rates = zip(*lr_schedule)
```

These three changes to the final architecture, with 200 epochs, a Sotfplus beta parameter modeled as a linear function and a learning rate constant at 0.001, resulting in a model that is not identical to the one proposed by the authors.

The final result is a model with a good performance in terms of RMSE when comparing differential learning and standard learning for the one European call model based on the Bachelier model in a basket with 1 stock (Figure 5.1) and when comparing the losses between both learnings from 1 to 20 stocks (Figure 5.2).

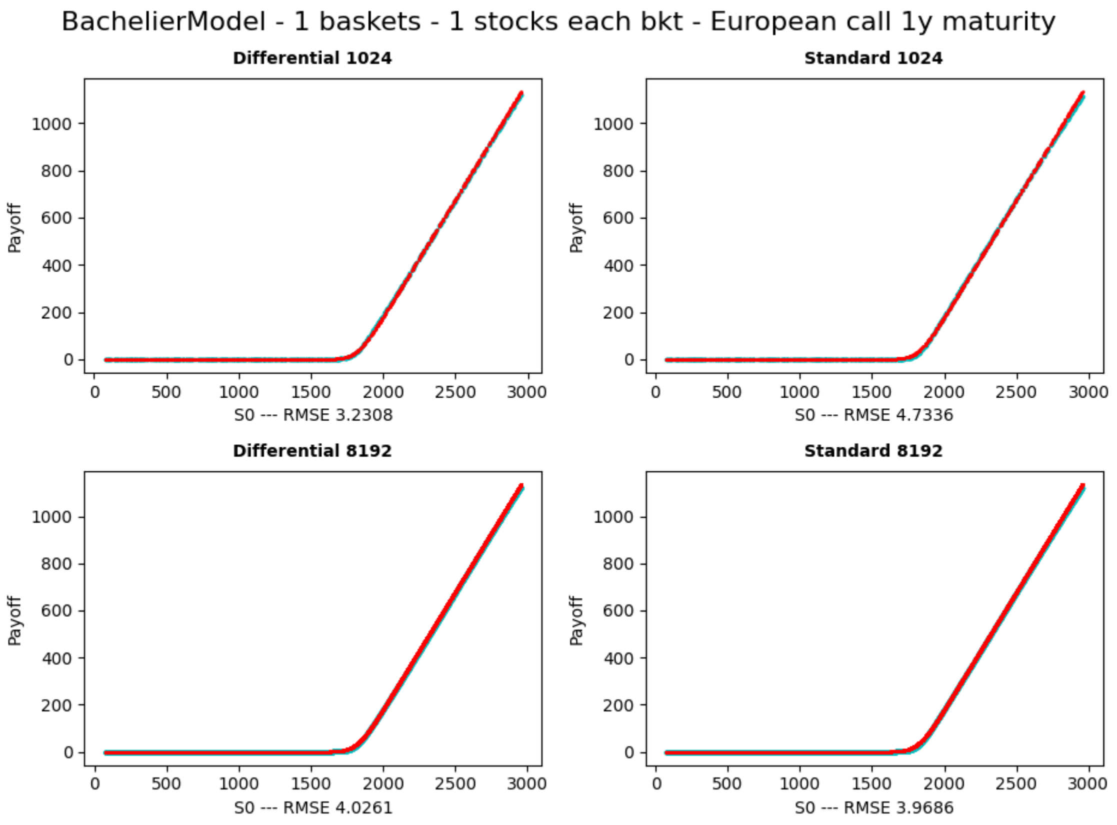


Figure 5.1: Predictions for 1 stock. Source: Author's own work.

Looking only at the predictions in Figure 5.1, one might think that differential learning is not as superior in prediction as standard learning, as Huge & Savine (2020) have shown. Even if we ran the same architecture as the authors, for the same seed, PyTorch and TensorFlow, the sequence of random numbers generated by each framework would be different because they rely on different Random Number Generator (RNG) implementations. For this reason, Figure 5.2 shows the *adv* values for a macro comparison along the dimensions.

Both visualizations were generated for BSMM, European put, with more than 2 baskets of options, for maturity 2 and a special one extrapolating the multidimensional universe with 20 stocks, in 20 baskets with maturity 20. In all, 244 visualizations like those in Figure 5.1 and another 244 like those in Figure 5.2 were generated to see if the translation was carried out properly and with the differential training substantially better to the standard.

Because of the large number of plots, just a few were put in Appendix B.

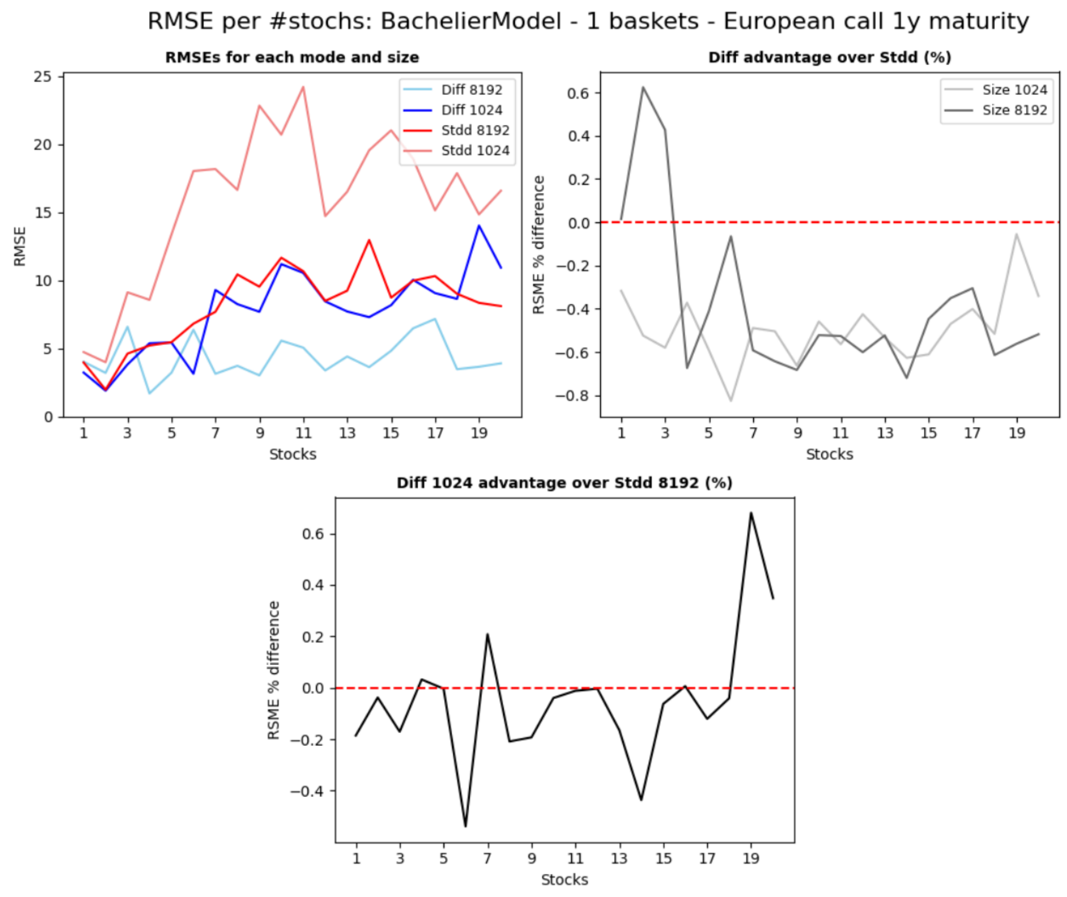


Figure 5.2: *adv* values for a European call from a Bachelier model (1 to 20 stocks). Source: Author's own work.

5.1.2 Improvements in the Domain

Once the translation task is completed, a no expected output appeared and it was added to the list of useful information that advances the knowledge of the thesis: *adv* plots similar to the one above were made from 1 to 50 stocks, and after 30 to 40 stocks, it looks like the differential model lost efficiency in predicting payoffs in comparison with the standard learning.

This phenomenon is observed for European calls and European puts, both for the Bachelier model and for BSMM. Sometimes it has a similar performance in terms of RMSE and all the times, the gross RMSE value of both learnings explodes. The phenomenon is also seen for 2 baskets and for maturity equals to 2. Figure 5.3 illustrates the European call under the Bachelier model. The other three plots are added in Appendix B.

The reason for this difference has not been verified mathematically, but it is worth

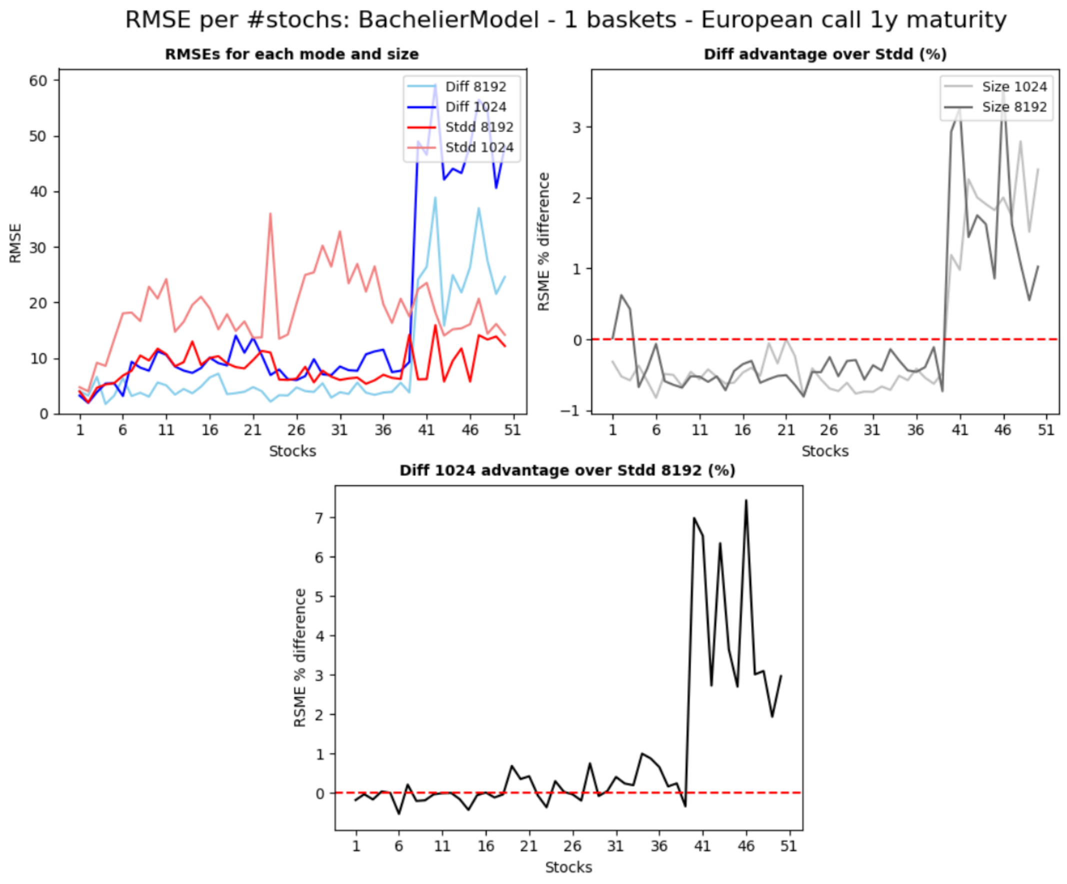


Figure 5.3: *adv* values for a European call under Bachelier model (1 to 50 stocks). Source: Author's own work.

noting that it is not common to trade options made up of baskets of dozens of shares. Therefore, the fact that is verified in this subsubsection remains a theoretical advance that can be taken in future works.

5.2 Components to Explore

5.2.1 Main Requirements

This subsection details the components and their parameters that will change. Following the list of Diff NN requirements presented in Figure 4.3:

- Number of layers: {3, 4, 5};
- Number of neurons: {15, 20, 25};
- Hidden layers: {3x20, 5x20, 4x15, 4x25, 5x25}, where 3x15 presented very bad results in the first runs and soon was discarded and where 5x25 was an additional test after 5x20 and 4x20 presented good results;

- Epochs: {100, 200}, where 100 is the number used by the authors, but 200 was considered our default as commented in subsubsection 5.1.1;
- Mini-batch data sets: { (256x4, 512x16), (16x64, 16x512), (8x256, 32x256) }, where the first pair of the set is the default;
- Activation function: {ReLU, LeakyReLU, LeakyReLU(slope=0.05), Softplus, ELU, ELU(alpha=2), Sotplus(beta=f), Softmax}, where the default slope for LeakyReLU is 0.01, the default alpha for ELU is 1, the default activation function used by the authors is Softplus with its default soft beta equal to 1 and f is a linear function:

$$f = \begin{cases} 3.05 - 0.5 * n\_stocks & \text{If } n\_stocks <= 11, \\ 3.1 - 0.1 * n\_stocks & \text{if } n\_stocks <= 21, \\ 1.0 & \text{if } n\_stocks >= 22 \end{cases} \tag{5.1}$$

In other words, an inverse relation between the number of the stocks and beta parameter value helps decrease the value of RMSE. A set of tests was made to prove this heuristic and they are presented in Appendix B. Important to say that a learnable beta parameter was tested with no success and that new tests can be made to improve the exact parameter value of f function. Figure 5.4 shows the relation.

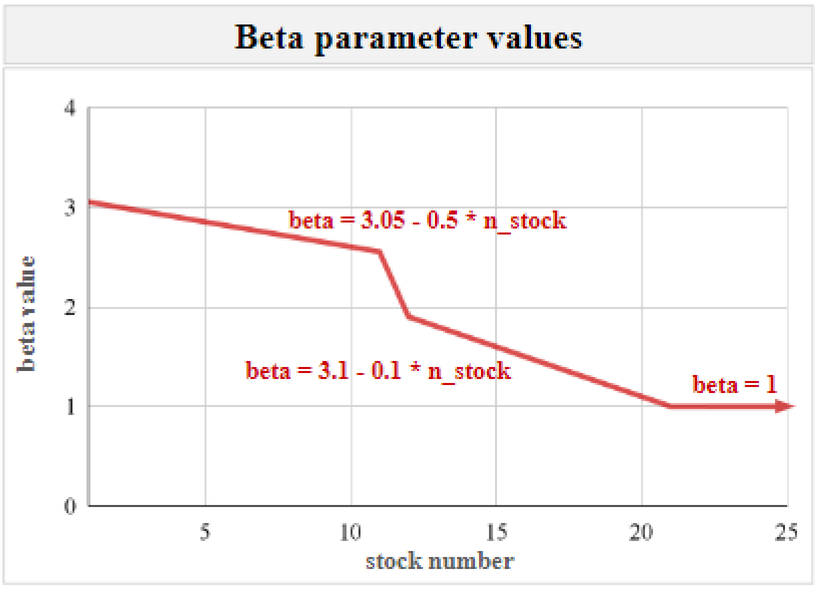


Figure 5.4: Beta parameter function. Source: Author's own work.

Other requirements to change are:

- Optimizer: {Adam, Adam(lr_schemas_below), Adamax, Adagrad, Adadelta}, where SGD and RMSProp presented bad results in the first runs and soon were discarded, proofing that adaptive optimizers are the best for a differential learning;

- Learning rate schema for Adam(): { (1.00e-08, 0.1, 0.01, 1.00e-06, 1.00e-08), (1.00e-06, 0.01, 0.0001, 1.00e-06, 1.00e-08), (0.001, 0.01, 0.001, 1.00e-05, 1.00e-08), (0.005, 0.02, 0.0075, 0.00025, 1.00e-08), constant=0.001 };

Figure 5.5 presents all schemas and illustrates the initial heat in the rate followed by a descent cold down.

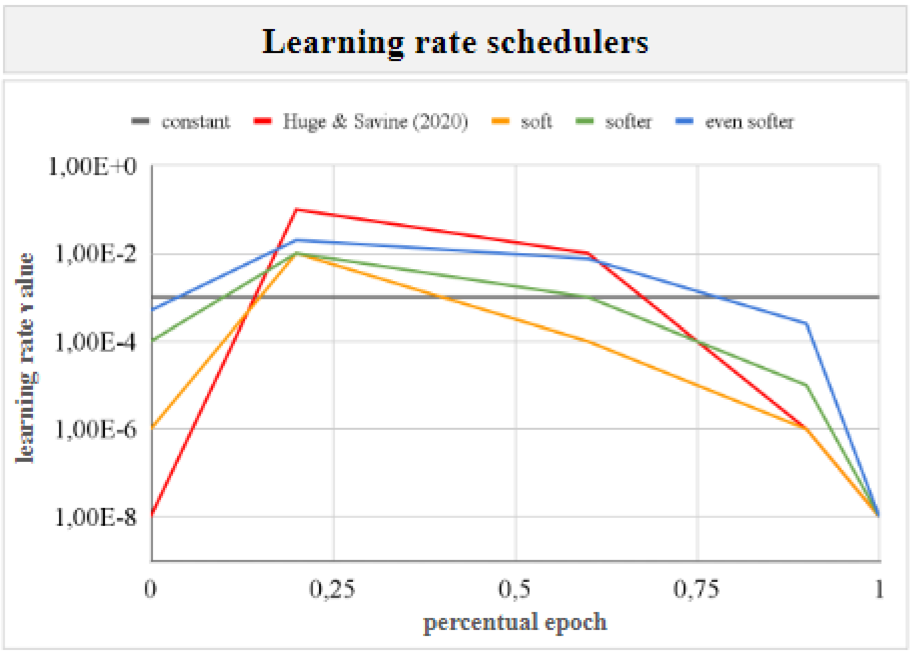


Figure 5.5: Learning rate scheduler schemas. Source: Author's own work.

Finally, last requirements to change are:

- Epoch percentage for each value of learning rate: stayed constant equal to the author's proposal: {0, 0.2, 0.6, 0.6, 1} for 100 epochs, and where it was adjusted to 200 epochs by maintaining 1.00e-08 learning rate value after 100th epoch. So, for 200 epochs the scheduler is {0, 0.1, 0.3, 0.45, 0.5}; and

- Weights initialization: {constant=0.05, uniform=(-0.1, 0.1), normal=(0, 0.5), Xavier uniform, Xavier normal, Kaiming uniform, Kaiming normal}, were the parameter values for the first three were decided on a discretionary basis based on the values observed on default weight initialization kaiming normal.

Considering 5 hidden layer structures (having one previously discarded), 2 epoch numbers, 3 mini-batch sets, 8 activation functions and 9 optimizer schemas (having 2 previously discarded), there are almost 7000 possible architectures. Considering the discarded parameters, almost 10000 architectures, a *ceteris paribus* analysis was carried out to find obvious suggestions for improving the architecture presented by Huge & Savine (2020).

5.2.2 Additional Requirements

Three additional requirements were introduced in the code just for future new approaches and why they were introduced:

- Activation function in the first layer: if the activation function is not ReLU, then it is not applied in the first layer, because in the generated data, when the stock price is lower than the strike, payoffs are already zero, so there is no need to apply this functions and resources are saved;
- Early stopping: option and by default "False", with a patience of 50 epochs, useful to save resources; and
- *lam* hyperparameter: when the differential loss is computed, two parameters balance the size of the standard and the differential loss, *alpha* and *beta*.

With respect to *lam*, it has impact in the differential loss:

Listing 5.4: Simple Python code

```
loss = alpha * loss(output, label) +
       beta * loss(diff_output * lambda_j, diff_label * lambda_j)
```

where $beta = 1 - alpha$ and *lambda_j* is a measure to put differential outputs and differential labels in the same level of magnitude of standard ones.

But *alpha* depends on *lam* that, by default, is equal to 1 by the authors. This value was not changed:

Listing 5.5: Simple Python code

```
lam = 1
alpha = 1 / (1 + lam * n_stocks)
```

Because of the constant value of $lam = 1$, when the dimension increases, so does *alpha* and the weight of the differential learning in the total learning. This is a good strategy to compensate for both the loss of precision and the increase in data variability with a good method to do predictions in Machine Learning.

Despite this, a commented line has been added for the situation in which it is desired to keep the share of differential training constant at 50% regardless of the number of stocks in the basket:

Listing 5.6: Simple Python code

```
lam = 1 / n_stocks
alpha = beta = 0.5
```

5.3 Evaluation Scenarios

This subsection is divided into four parts: (i) the first creates groups of components that will be changed and a list of the values that each component can obtain;

(ii) the second introduces how RMSE values are presented and does this for the initial architectures presented by Huge & Savine (2020) that we call "original architecture" and the so-called "default architecture" here found at improvements presented in subsubsection 5.1.1; (iii) the third advances with RMSE values for each of the all architectures listed and select those who better performance has; and (iv) the last validate the best architectures for different seeds.

5.3.1 Building Scenarios

The construction of the scenarios organizes the alterations made to the original architecture *ceteris paribus*. Based on the requirements presented in 5.2.1, tables are presented with the list of changes per group of components.

Table 5.1 refers to changes in the structure of the hidden layers.

Table 5.1: Hidden layers list to be tested. Source: Author's own work.

Code	Component structure
Default	4x20 = (20, 20, 20, 20)
A.1	3x20 = (20, 20, 20)
A.2	5x20 = (20, 20, 20, 20, 20)
A.3	4x15 = (15, 15, 15, 15)
A.4	4x25 = (25, 25, 25, 25)
A.5	5x25 = (25, 25, 25, 25, 25)

Table 5.2 refers to changes in the number of epochs:

Table 5.2: Epochs list to be tested. Source: Author's own work.

Code	Component structure
Default	200
B.1	100

Table 5.3 refers to changes in the structure of mini-batch sets:

Table 5.3: Mini-batch data sets to be tested. Source: Author's own work.

Code	Component structure
Default	(4x256, 16x512)
C.1	(16x64, 16x512)
C.2	(4x256, 32x256)

Table 5.4 refers to changes in activation functions:

Table 5.4: Activation functions list to be tested. Source: Author's own work.

Code	Component structure
Default	Softplus(beta=f)
D.1	Softplus()
D.2	ReLU()
D.3	LeakyReLU()
D.4	LeakyReLU(slope=0.05)
D.5	ELU()
D.6	ELU(alpha=2)
D.7	GELU()
D.8	Softmax()

Table 5.5 refers to changes in the optimizers and their learning rates:

Table 5.5: Optimizers list to be tested. Source: Author's own work.

Code	Component structure
Default	Adam()
E.1	Adagrad()
E.2	Adamax()
E.3	Adadelta()

Table 5.6 refers to changes in the learning rate for the Adam optimizer:

Table 5.6: Learning rate list to be tested. Source: Author's own work.

Code	Component structure
Default	Adam(lr=0.001)
F.1	Adam(lr=(1.00e-08, 0.1, 0.01, 1.00e-06, 1.00e-08)) $\rightarrow$ "authors' schema"
F.2	Adam(lr=(1.00e-06, 0.01, 0.0001, 1.00e-06, 1.00e-08)) $\rightarrow$ "soft")
F.3	Adam(lr=(0.001, 0.01, 0.001, 1.00e-05, 1.00e-08)) $\rightarrow$ "softer"
F.4	Adam(lr=(0.005, 0.02, 0.0075, 0.00025, 1.00e-08)) $\rightarrow$ "even softer"

Table 5.7 refers to changes in the way the model weights are initialized:

Table 5.7: Weight initialization list to be tested. Source: Author's own work.

Code	Component structure
Default	Kaiming normal
G.1	Constant = 0.05
G.2	Uniform = (-0.1, 0.1)
G.3	Normal = $\mathcal{N}(0, 0.5)$
G.4	Xavier uniform
G.5	Xavier normal
G.6	Kaiming uniform

5.3.2 Results of Initial Architectures

This subsubsection presents four metrics to show the effect of each change on the default architecture. Three metrics are related to the prediction quality and one is related to the processing time of the code, all presented in subsection 4.3.

RMSE is calculated for training with 1024 and 8192 observations, for the differential and standard model. The four trainings take place both for Bachelier and BSMM as analytical equations, and for European put and call as financial instruments. Therefore, 16 trainings are carried out for each architecture. RMSE values used to compare the architectures are the average of the training carried out on between 1 and 20 stocks per basket, as it is simpler to compare them with an aggregation of the quantity of stock to the individual quantity of stock analysis. Visualizations are generated to observe the behavior at each number of stocks not not to lose this individual dimension and to understand what happens to the RMSE results as the dimension is increased.

To offer two examples of this analysis, let's recall the architecture proposed by the authors and the one found by this research at the end of the translation (the default architecture). First, we list the characteristics of Huge & Savine (2020):

- Hidden layers: (20, 20, 20, 20)
- Epochs: 100
- Mini-batches data sets: (4x256 , 16x512)
- Activation functio: Softplus()
- Optimizer: Adam()
- Learning rate: (1.00e-08, 0.1, 0.01, 1.00e-06, 1.00e-08)
- Weight initialization: Kaiming normal

Figure 5.6 shows the evolution of the RMSE from 1 to 20 stocks for the Bachelier model and the call instrument.

Remembering the concept of "preference" in Economics, where one result is better than the other ("$\succ$"), Table 5.8 presents results for Huge & Savine (2020) architecture. The last lines simplify the reading of the three advantage lines, by writing 1 when differential learning has better performance over standard, -1 the opposite and 0 when is similar with a margin of error of plus or minus 5 percentage points.

Table 5.8: Huge & Savine (2020) architecture results. Source: Author's own work.

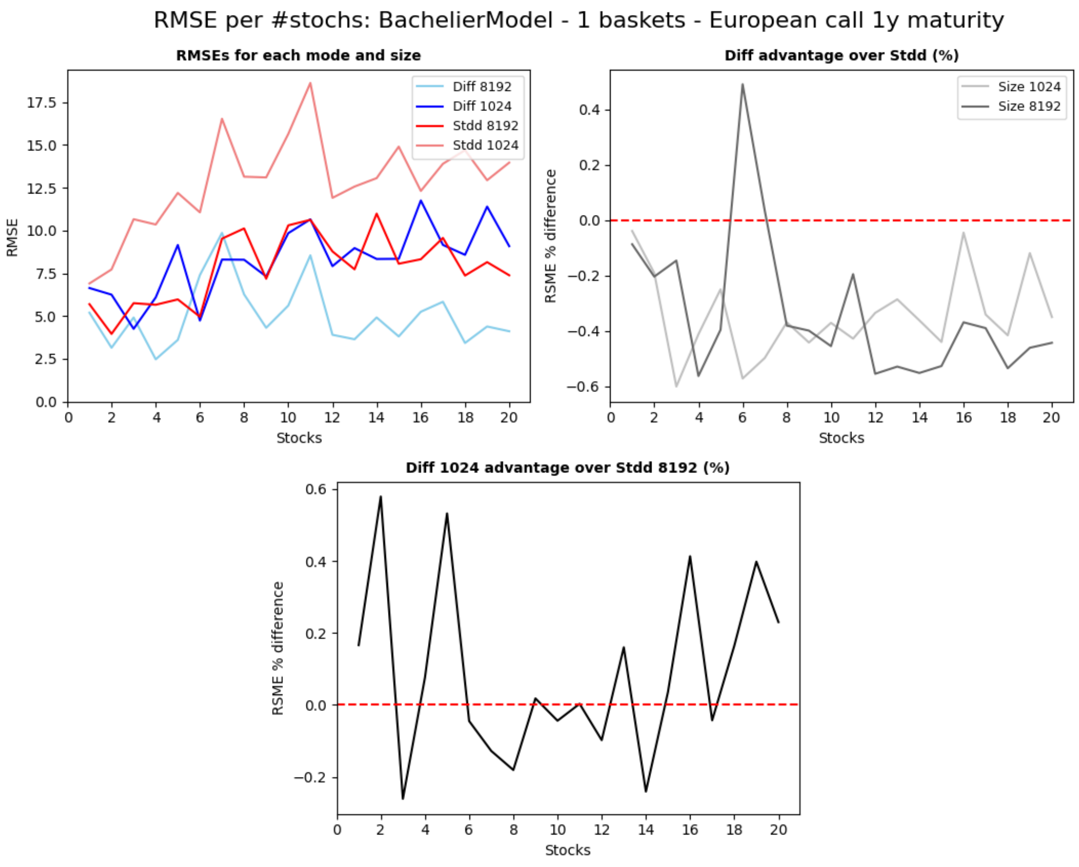


Figure 5.6: *adv* values for European call from a Bachelier model (1-20) for Huge & Savine (2020) architecture. Source: Author's own work.

Huge & Savine (2020) architecture				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	8.25	16.97	11.83	12.14
Stdd 1024	12.80	34.53	32.20	27.54
Diff 8192	5.03	7.52	10.74	10.15
Stdd 8192	7.80	16.13	16.90	17.78
1024 adv	-35.55%	-50.85%	-63.37%	-55.92%
8192 adv	-35.51%	-53.38%	-36.45%	-42.91%
D1024-S8192 adv	5.77%	5.21%	-30.00%	-31.72%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	-1	-1	1	1

In aggregation terms for the third indicator, call and put only vary in the payoff calculation by an inversion of the S and K portions and are, therefore, practically identical in performance terms. Table 5.9 aggregates all the elements of Bachelier and BSMM. For the moment, we don't want to compare analytical equations, only architectures, but it is interesting to note that Bachelier and BSMM are similar in different ways, the former for having a better average RMSE and the latter for having a greater average advantage.

Table 5.9: Summary of Huge & Savine (2020) architecture results. Source: Author's own work.

Huge & Savine (2020) architecture		
Indicator	Bachelier	BSMM
Avg RMSE	13.63	17.42
Avg adv	-27.38%	-43.40%
Arch adv	9.90	9.86

Evaluating the default architecture, remembering the characteristics:

- Hidden layers: (20, 20, 20, 20)
- Epochs: 200
- Mini-batches data sets: (4x256 , 16x512)
- Activation function: Softplus(beta=f)
- Optimizer: Adam()
- Learning rate: 0.001
- Weight initialization: Kaiming normal

The figure that repeats the logic of the previous one is Figure 5.2, presented in subsubsection 5.1.1.

Table 5.10 shows the results for the default architecture.

Table 5.10: Default architecture results. Source: Author's own work.

Default architecture				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	7.70	9.04	13.13	12.37
Stdd 1024	15.76	26.29	76.90	39.94
Diff 8192	4.32	5.57	10.73	11.37
Stdd 8192	8.15	12.90	38.47	21.89
1024 adv	-51.44%	-65.61%	-82.94%	-69.03%
8192 adv	-46.99%	-56.82%	-72.11%	-48.06%
D1024-S8192 adv	-5.52%	-29.92%	-65.90%	-43.49%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	1	1	1	1

And in Table 5.11, a comparative summary of the analytical equations. In this case, Bachelier proved to be better because it had a good advantage with a low RMSE and, despite a greater advantage, BSMM has an average RMSE whose value is more than double that of Bachelier. Even so, compared to the authors' architecture, this architecture has a superior predictive performance and is therefore considered the starting point for the following *ceteris paribus* comparison.

Table 5.11: Summary of default architecture results. Source: Author's own work.

Huge & Savine (2020) architecture		
Indicator	Bachelier	BSMM
Avg RMSE	11.62	27.94
Avg adv	-43.35%	-63.13%
Arch adv	6.57	9.47

5.3.3 Scenarios Results

In this subsubsection, the same exercise done before is made for codes from A.1 to G.6, presenting results and summary results for each one. The changes are made over the default architecture. In the discussion part of the chapter, all results are summarized and compared. A new information group is added: "Better adv than default architecture?". The question checks if *adv* results are better ("YES"), similar with a margin of error of plus or minus 5 percentage points ("SAME") or worse ("NO"). Tables 5.12 present one example of a change A.1, by removing one layer. All other 28 *ceteris paribus* changes are presented in Appendix B.

Table 5.12: 3x20 results. Source: Author's own work.

A1: 3x20 = (20,20,20)				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	13.97	9.61	12.87	18.33
Stdd 1024	15.60	28.07	37.29	58.95
Diff 8192	9.84	5.18	10.45	15.87
Stdd 8192	8.13	12.51	20.56	24.92
1024 adv	-10.45%	-65.76%	-65.49%	-68.91%
8192 adv	21.03%	-58.59%	-49.17%	-36.32%
D1024-S8192 adv	71.83%	-23.18%	-37.40%	-26.44%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	-1	1	1	1
D 1024 $\succ$ S 8192	-1	1	1	1
Better adv than default architecture?	NO	SAME	NO	SAME
	NO	SAME	NO	NO
	NO	NO	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	12.86	24.91
Avg adv	-10.85%	-65.12%
Avg arch adv	11.47	13.13

After making individual *ceteris paribus* changes to the neural network components, some changes presented good results. The statement that the result is good was based on the following criteria (C_i):

- $\underline{C_1}$: *arch_adv* value;

- $\underline{C_2}$: sum of times Diff is preferable to the Stdd on Tables 5.12 to 5.40; and

- $\underline{C_3}$: the sum of times differential is preferable to standard for all stock numbers, i.e., in this measure, RSME plots are made to compare advantage results and all negative advantages, meaning differential is better than standard, are added. It is the sum of 1024 *adv*'s, 8192 *adv*'s and 1024-diff x 8192-stdd *adv*'s for all analytical equations and instruments. To provide a visualization of C3, we return to Figure 5.6 and in the *adv* plots, we count how many points are above 0. This measure might look arbitrary, but is a way to check how robust results are along new dimensions since it is expected differential learning presents many times better performance (see appendix B).

Table 5.13: Criteria results for all changes. Source: Author's own work.

Code	C1 value	C2 value	C3 value	C1	C2	C3	t
Original	7.90	11/12	211/240	●	●		●
Default	9.21	12/12	220/240	●	●	●	●
A.1	13.39	9/12	213/240				●
A.2	9.49	12/12	225/240	●	●	●	
A.3	9.05	12/12	220/240	●	●	●	●
A.4	10.83	11/12	227/240	●		●	
A.5	9.65	12/12	228/240	●	●	●	
B.1	10.19	11/12	206/240	●			●
C.1	10.83	12/12	218/240	●	●		●
C.2	10.38	11/12	214/240	●			●
D.1	9.15	9/12	195/240	●			●
D.2	63.30	11/12	196/240				●
D.3	63.17	11/12	202/240				●
D.4	64.81	11/12	201/240				●
D.5	16.22	12/12	218/240		●		●
D.6	10.14	12/12	227/240	●	●	●	●
D.7	31.53	9/12	213/240				●
D.8	165.24	8/12	187/240				●
E.1	9.44	12/12	222/240	●	●	●	●
E.2	9.71	12/12	210/240	●	●		●
E.3	20.46	11/12	195/240				●
F.1	14.49	12/12	222/240		●	●	●
F.2	9.37	11/12	222/240	●		●	●
F.3	8.51	12/12	223/240	●	●	●	●
F.4	11.44	11/12	223/240	●		●	●
G.1	26.26	9/12	163/240				●
G.2	10.61	10/12	191/240	●			●
G.3	8.81	10/12	195/240	●			●
G.4	19.25	9/12	206/240				●
G.5	8.57	11/12	205/240	●			●
G.6	18.63	7/12	212/240				●

For three metrics, it is considered both Bachelier and BSMM, because the interest is in an agnostic architecture. For C_1, *arch_adv* for all four scenarios (Bachelier and BSMM for Europeans call and put) and for C_2 and C_3, sum of the values.

By checking all measures, Table 5.41 presents values for each criterion and if it presents good values for each architecture. For C_1, less than 12 is a good heuristic result. For C_2, it is considered to have a 12/12 value similar to default architecture. For C_3, it is considered more than the default 220/240 value. Time observations made in 5.3.3 are added as well.

After checking Table 5.41, the best architecture changes are presented in Table 5.42 and will be analyzed in the next subsubsection.

Table 5.14: List of best architectures. Source: Author's own work.

Code	Best architecture changes
A.3	4x15 = (15,15,15,15)
D.6	ELU(alpha=2)
E.1	Adagrad()
F.3	Adam(lr = (0.001, 0.01, 0.001, 1.00e-05, 1.00e-08)) $\rightarrow$ "softer"

5.3.4 Testing Robustness of the Best Scenarios

As commented in subsection 4.4, four more seeds were selected to test the robustness of the best architectures: 5768, 9009, 8765 and 4321.

Authors' and default architectures were tested in new seeds to check their robustness as this will help you choose the best architecture. Tables 44 and 45 show the results of the three metrics for both the Bachelier model and BSMM.

Table 5.15: Original architecture for different seeds. Source: Author's own work.

Huge & Savine (2020) architecture			
Seed	Indicator	Bachelier	BSMM
1234	Avg RMSE	13.63	17.42
(Table 5.9)	Avg arch	-27.39%	-43.40%
	Arch adv	9.90	9.86
5678	Avg RMSE	14.15	17.19
	Avg arch	-31.22%	-45.87%
	Arch adv	9.73	9.30
9009	Avg RMSE	17.42	21.80
	Avg arch	-8.15%	-38.48%
	Arch adv	16.00	13.41
8765	Avg RMSE	19.68	22.15
	Avg arch	-13.72%	-21.65%
	Arch adv	16.98	17.35
4321	Avg RMSE	17.62	22.85
	Avg arch	25.43%	17.97%
	Arch adv	22.10	26.96

Table 5.16: Default architecture for different seeds. Source: Author's own work.

Default architecture			
Seed	Indicator	Bachelier	BSMM
1234 (Table 5.11)	Avg RMSE	11.62	27.94
	Avg arch	-43.35%	-63.13%
	Arch adv	6.57	9.47
5678	Avg RMSE	12.03	27.47
	Avg arch	-24.68%	-51.89%
	Arch adv	9.06	13.22
9009	Avg RMSE	21.15	33.94
	Avg arch	105.26%	-38.66%
	Arch adv	43.41	20.82
8765	Avg RMSE	13.47	30.02
	Avg arch	-24.22%	-49.59%
	Arch adv	10.21	15.13
4321	Avg RMSE	16.90	31.30
	Avg arch	74.04%	-24.15%
	Arch adv	29.42	23.75

For the best architectures found in the previous subsubsection, Tables 5.45 to 5.48 show the results of the three metrics for both the Bachelier model and BSMM. 1234 seed values are put again to help compare all five seed simulations.

Table 5.17: A.3 results for different seeds. Source: Author's own work.

A.3 architecture change			
Seed	Indicator	Bachelier	BSMM
1234	Avg RMSE	11.01	23.19
	Avg arch	-38.09%	-56.01%
	Arch adv	6.82	10.20
5678	Avg RMSE	10.17	16.48
	Avg arch	-33.59%	-44.30%
	Arch adv	6.75	9.18
9009	Avg RMSE	10.68	21.13
	Avg arch	-12.07%	-37.15%
	Arch adv	9.39	13.28
8765	Avg RMSE	12.03	21.07
	Avg arch	-27.29%	-41.02%
	Arch adv	8.75	12.43
4321	Avg RMSE	10.05	17.55
	Avg arch	-33.55%	-39.44%
	Arch adv	6.68	10.63

Table 5.18: D.6 results for different seeds. Source: Author's own work.

D.6 architecture change			
Seed	Indicator	Bachelier	BSMM
1234	Avg RMSE	12.67	25.74
	Avg arch	-42.71%	-51.74%
	Arch adv	7.26	12.42
5678	Avg RMSE	18.86	22.08
	Avg arch	119.43%	-43.91%
	Arch adv	41.38	12.38
9009	Avg RMSE	23.99	26.66
	Avg arch	186.70%	-47.92%
	Arch adv	68.78	13.88
8765	Avg RMSE	16.74	25.98
	Avg arch	68.91%	-50.11%
	Arch adv	28.28	12.96
4321	Avg RMSE	20.78	24.47
	Avg arch	138.82%	-44.17%
	Arch adv	49.63	13.66

Table 5.19: E.1 results for different seeds. Source: Author's own work.

E.1 architecture change			
Seed	Indicator	Bachelier	BSMM
1234	Avg RMSE	14.22	21.42
	Avg arch	-42.88%	-51.15%
	Arch adv	8.12	10.46
5678	Avg RMSE	189.18	115.02
	Avg arch	24.33%	22.60%
	Arch adv	235.20	141.01
9009	Avg RMSE	228.17	149.21
	Avg arch	11.46%	20.50%
	Arch adv	254.31	179.79
8765	Avg RMSE	256.47	156.21
	Avg arch	35.96%	39.61%
	Arch adv	348.70	218.08
4321	Avg RMSE	213.06	133.69
	Avg arch	22.21%	29.32%
	Arch adv	260.39	172.89

Table 5.20: F.3 results for different seeds. Source: Author's own work.

F.3 architecture change			
Seed	Indicator	Bachelier	BSMM
1234	Avg RMSE	10.91	27.70
	Avg arch	-46.07%	-65.79%
	Arch adv	5.88	9.48
5678	Avg RMSE	10.96	17.15
	Avg arch	-41.24%	-46.37%
	Arch adv	6.44	9.20
9009	Avg RMSE	10.89	20.81
	Avg arch	-11.22%	-40.21%
	Arch adv	9.67	12.44
8765	Avg RMSE	15.70	22.27
	Avg arch	-4.04%	-46.23%
	Arch adv	15.07	11.97
4321	Avg RMSE	11.97	18.26
	Avg arch	-26.37%	-41.04%
	Arch adv	8.81	10.77

A new analysis using the three criteria follow in Table 5.49. Because of variation in the initialization, more variable data is expected, so values for each criterium are relaxed to accept some variation: for C_1, values under 14.5 (around 20% relaxation); for C_2, at least the Huge & Savine (2020) 11/12 value; and for C_3, at least the Huge & Savine 211/240 value.

Table 5.21: Criteria results for best changes. Source: Author's own work.

Code	Seed	C1 value	C2 value	C3 value	C1	C2	C3
Original	1234	7.90	11/12	211/240	•	•	•
	5678	9.63	11/12	208/240	•	•	
	9009	15.04	10/12	203/240			
	8765	17.21	9/12	176/240			
	4321	24.63	6/12	164/240			
Default	1234	9.21	12/12	220/240	•	•	•
	5678	12.19	11/12	218/240	•	•	•
	9009	36.71	9/12	200/240			
	8765	13.72	11/12	199/240	•	•	
	4321	30.12	8/12	173/240	•		
A.3	1234	9.05	12/12	220/240	•	•	•
	5678	8.14	12/12	210/240	•	•	
	9009	11.99	10/12	205/240	•		
	8765	10.90	11/12	196/240	•	•	
	4321	8.76	12/12	200/240	•	•	
D.6	1234	16.22	12/12	218/240	•	•	•
	5678	28.20	6/12	163/240			
	9009	42.90	8/12	177/240			
	8765	23.37	6/12	160/240			
	4321	33.33	7/12	148/240			
E.1	1234	9.44	12/12	222/240	•	•	•
	5678	187.79	2/12	31/240			
	9009	218.84	2/12	21/240			
	8765	284.30	3/12	31/240			
	4321	218.05	5/12	12/240			
F.3	1234	8.51	12/12	223/240	•	•	•
	5678	7.90	11/12	205/240	•	•	
	9009	11.77	10/12	206/240	•		
	8765	14.21	11/12	197/240	•	•	
	4321	10.02	10/12	204/240	•		

From this final evaluation of robustness, D.6 and E.1 architectures had, randomly, excellent results on seed 1234, since differential learning proved poor at prediction under other seeds. On the other hand, default, A.3 and F.3 architectures are relatively robust for different seeds.

It means that compared to the default architecture found at the end of the translation (which already had changed compared to the original), it is possible to reduce the number of neurons from 20 to 15 in all four layers as a way of saving a little more time and maintaining or changing the learning rate schedule to smooth out the learning intensity without large jumps.

5.4 Discussion

This subsection discusses, based on the limitations and contributions of this work: (i) the lessons learned during the process of developing and testing the code; (ii) a critical analysis of the results; and (iii) an assessment of how to move forward in the literature.

At the beginning of the practical part, there were technical difficulties in gaining access to the data and for this reason, data was created from scratch, with which it was possible to reproduce the inaugural study by Huge & Savine (2020). This choice of action does not diminish the validity of the results because, like the authors, simple models were used for the analytical equations. More complex equations better reflect the reality of the operating tables and in them, the application of Diff PCA as an optimization strategy is more evident, however, there was the initial work of translating from one library to another. Once the translation of simpler models, the initial focus was guaranteed, it would be possible to work with more complex equations in the future.

Therefore, a second challenge was getting to know PyTorch, which has so many features and flexibility that lines of code could often be summarized in methods that only exist in this library. As mentioned in 4.1.1, there are many advantages to using PyTorch over TensorFlow, so this was a non-negotiable requirement. In parallel with the translation, there was a lot of learning and testing around the documentation to get the outputs right. As a result, the main task of translation took up most of the work, leaving less time to carry out the architectural tests.

About the results, both the RMSE values and the visualization of the theoretical expected payoff curve against the cloud of expected payoffs, together, are in line with the initial expectations of the research. Without the visualization, the RMSE values are even slightly higher than those observed by Huge & Savine (2020), but the plots materialize the correct prediction.

Another interesting observation regarding the results can be made by looking at Tables 13 to 41 and Tables 44 to 49: in general, the RMSE values are better for the Bachelier model and the adv values are better for BSMM. This is a finding based on data and requires a mathematical explanation which was not carried out in this study.

As for the comparative analysis using different amounts of data (1024 and 8192 observations) and the most interesting comparison between 1024 observations for Diff ML and 8192 for standard learning, the predictive superiority of the differential was proven. In this sense, demonstrating this property was carried out successfully, because although unexpected behavior was discovered from 30 to 40 stocks, it is unusual for baskets to have large numbers of stocks. When focusing on predictive analysis of up to 20 stocks, as Huge & Savine (2020) do, Table 42 shows that the worst architecture had 163 [out of 240] predictions in which Diff ML had the best RMSE results. The average of the changes made was 209 out of 240, which corresponds to 87.08% for the same seed used by the authors. And although PyTorch's "1234" seed doesn't have the same internal initialization engine as TensorFlow's seed, for the six architectures tested with the other four seeds in 5.3.4, three of them proved to be more robust: default, A.3 and F.3, with the A.3 architecture still helping to improve fast prediction since it has 5 fewer neurons in each of the 4 hidden layers.

An additional evaluation that could have been carried out is to return to the list of changes and look for the architectures that were not the best, but were good, because it is likely that in seed 1234 there were satisfactory results that could be maintained or even improved under other seeds. In other words, we could have used less stringent values for C_1, C_2 and C_3 when selecting the best architectures. For example, using the same criteria used in the second evaluation, architectures C.1, C.2, E.2, F.1, F.2 and F.4 could have been tested. This shows that changes in the organization of the batches of data and the size of the learning rates are the ones that cause the most positive changes and that any change in the initialization of the weights will always worsen the predictions. It was also observed that the adaptive optimizers of the Adam family contribute the most to good predictions. In contrast, optimizers such as those based on gradients and the adaptive RMSProp worsen the results. On the other hand, changes to activation functions and the number of epochs do not bring any relevant gains or losses, since the former have different behaviors and the latter change the time in the same proportion as they change the quality of the forecast. And related to the learning rate schedule, the percentage of the epoch where the learning values are altered has not changed and, in this sense, there is room for further exploration of architectures. Table 5.50 shows a summary of the effect of changes to the main requirements in terms of improving the forecast based on changes made to the "default" architecture, i.e., reducing the RMSE.

Table 5.22: Summary of the effects of the changes on RMSE.
Source: Author's own work.

Requirement	Effect on RMSE
Layers	Easy to improve
Neurons	Easy to improve
Epochs	Neutral
Mini-batch datasets	Easy to improve
Activation functions	Neutral
Optimizers (Adam family)	Easy to improve
Other optimizers	Easy to worse
Learning rate scheduler	Easy to improve
Weight initialization	Easy to worse

During the evaluation of the architectures, a metric that had not yet been applied in any research previously produced within the Diff ML area, *adv_arch*, is proposed, since there is still little research carried out and none of them focused on comparing both learnings. This is a great contribution to the literature and can be explored by other areas of knowledge.

Move towards an architecture that can be faster and that certainly brings improvements, advances the Diff ML literature and offers a starting point for future research using PyTorch. In this sense, the excess time spent on translation creates a new benchmark for this area of research. Any of the three architectures presented as relatively robust (default, A.3 and F.3) are good ML tools for Fast Pricing Option and can be tested in other areas of Data Science to assess their qualities.

With better architectures found about the inaugural architecture of Huge & Savine (2020), it is possible to improve the Bachelier RMSE from a minimum of 13.63 to 10.05 in A.3 and improve the BSMM RMSE from 17.19 to 16.48 also in A. 3. Likewise, it is possible to improve Bachelier's *adv* from a maximum of -31.22% to -46.07% in F.3 and improve BSMM's *adv* from -45.87% to -65.79% also in F.3. Finally, the lowest Bachelier's *arch_adv* was 9.73, while F.3 offers 5.88 and the lowest BSMM's *arch_adv* was 9.30, while A.3 and F.3 offer, respectively, 9.18 and 9.20.

Based on the difference in behavior between Bachelier and BSMM, the robustness of the second, widely known and used in theory and practice in the options market, is such that differential learning proves to be much superior to traditional learning. Therefore, Diff ML can and should be applied as a Fast Option Pricing tool for widely known models.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

When recalling the main objective of the thesis, the initial proposal was to perform a supervised option pricing model for a regression problem by replacing the traditional Monte Carlo simulation with a Diff ML algorithm, divided into four main tasks: (i) translation of all elements of Huge & Savine (2020) code that executes the generation, preprocessing, training and testing of data; (ii) exploration of elements that allow the use of the Diff ML model for different options, theoretical models and training types; (iii) evaluation of each architecture; and (iv) validation of the best ones.

As part of the organization, during the translation, no changes to parameters and hyperparameters would be made, however, as part of the attempt to obtain the expected results, some small changes were made, so that at the end of the first task, not only was the translation carried out such as obtaining an architecture with better functioning about that of the authors. Although this deviation generates different results for each change *ceteris paribus*, since they are not being carried out based on the authors' architecture, this did not affect the exploration of distinct components or their evaluation.

Therefore, it can be stated that the objective was not just achieved but overachieved, since the basics were to simply translate and before the changes in architecture, some improvements were made in the original architecture: change in the functioning of the beta parameter of the Softplus() activation function as the basket size increases, increase in the number of epochs without relative losses in processing time, learning rate that can be constant or scheduler, and a number of neurons that can be 15 or 20 for a fixed number of 4 hidden layers.

Another overachieved objective of the work carried out with BNP Paribas CIB was the integration of different parts of the Huge & Savine (2020) code in terms of running a code for more than one analytical equation, more than one instrument, more than one basket of shares and different types of maturity. The original code had all these elements, but in separate Jupyter Lab files.

For the three proposed metrics, RMSE, *adv* and *adv_arch*, it was possible to find in the default architectures, A.3 and F.3, with the characteristics mentioned above, values whose maximums were better than those of Huge & Savine (2020), contributing, therefore, for a not only methodological advancement in evaluation between different architectures with the introduction of the *adv_arch* metric, but also in the promotion of a new benchmark for PyTorch within the Fast Pricing Option activities.

6.2 Future Work

Since Diff ML is a brand new area with a potential for growing in research and development inside university and companies, some future works can be done in the following of this thesis.

A first main idea (very specific and well related to the work that was did) is to give continuity to the developing code and the first line of attack is to keep adding more options for objects that the user can choose, for example, in order to program new analytical equations and new option instruments for final users. For analytical equations, add simple moves like a GMB or complex models such as HM or even an internal bank model capturing different economic variables. For option instruments, there are dozens of different financial instruments that can be applied for both puts and calls. Figure 6.1 shows a more complete current code.

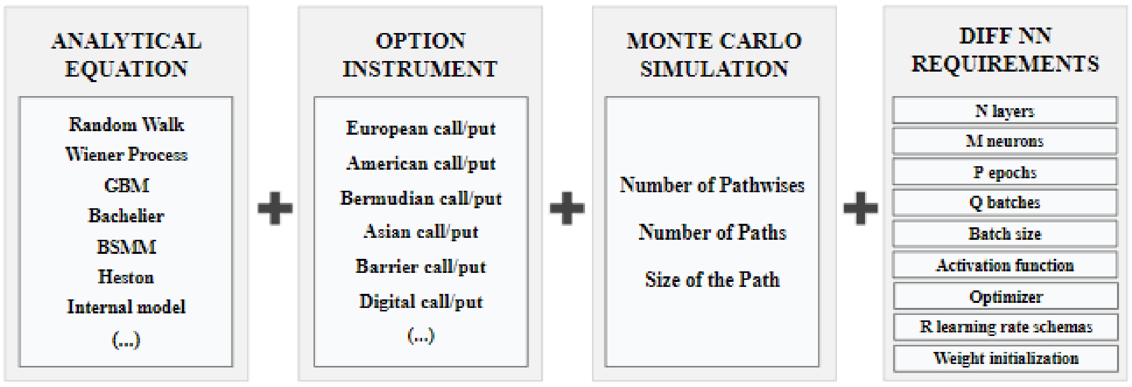


Figure 6.1: Completed scenario by adding new analytical equations and instrument options. Source: Author's own work.

Also in this first line of improving existing code, it is possible to enhance parameter testing so that the model is automated to improve itself. To do this, you could apply a "grid search" mechanism to the components, their parameters and hyperparameters, or work with "learnable parameters" by creating small classes to encapsulate and organize the information on the desired parameters. For both approaches, it will be necessary to validate the prediction gain-loss of processing speed relationship.

A third line of attack is to complete Huge & Savine (2020) code to proceed with Diff PCA, because, in addition to speeding up the training even more, following the main idea of Fast Option Pricing, it will also be possible to offer this option to the user.

A last line of attack inside the first main idea is to model variables differently. For example, in this work, volatility was treated as constant across time, but it can be variated or stochastic. It would be interesting to add a trend to prices as well, be it continuous or stochastic. This line of research, in particular, is an example of what can be done within analytical equations. The interesting thing about this part is that, as there are more non-constant variables, the usefulness of applying Diff PCA increases.

Since it is possible to predict the profit curve of options contracts, another potential application of Diff ML in Finance is to price other derivatives, such as futures and swaps, and even other variable income assets like cryptocurrencies and bonds or fixed income like bonds. The applications are immense since capital markets are full of forecasting problems and dependent on the expectation of so many economic variables.

Once it is well known the superiority of the first over the second, a second main line of research could compare between differential architectures themselves and between standard architectures themselves, in order to know how sensible each one is for the same component change. The potential knowledge is to understand which components generate the greatest changes in the learning of each network, for example, understanding whether differential networks are more sensitive to dropout than standard networks, or whether these are more sensitive to hyperparameters of a particular family or activation function than those ones. Perhaps some development and mathematical argumentation will be necessary. Once it's proved in Literature that some changes are better than others, it will be possible to reduce the space for grid search or learnable parameters mentioned above.

Another methodological research can compare the efficiency of encoders and decoders by reducing dimensionality compared to Diff PCA. Even more, a theoretical and practical research about applying these technologies inside a differential learning. It is interesting to know whether it is possible to use the same mathematical and computational structure with differential data or whether it would be necessary to create a new dimensionality reduction logic.

In terms of complementing other areas of Data Science, other work could focus on collecting information via the Internet to help improve option price forecasting. In this way, another category of information is added to the set of financial variables: market sentiment, which can even be passed through dimensionality reduction filters to validate its importance for option pricing.

Bibliography

[1] Abadeh, M. N., Derakhshandeh, Z., & Mirzaie, M. (2022). An Efficient Collaborative Filtering for Recommendation Systems Using Differential Machine Learning. In 2022 9th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS) (pp. 1-7). Bam, Iran, Islamic Republic of. 10.1109/CFIS54774.2022.9756424.

[2] Abadeh, M. N., & Mirzaie, M. (2023). A differential machine learning approach for trust prediction in signed social networks. Journal of Supercomputing, **79**(12), 9443-9466. 10.1007/s11227-023-05044-2.

[3] Abegaz, B. W. (2022). ADMSV - A Differential Machine Learning based Steering Controller for Smart Vehicles. In 2022 IEEE World AI IoT Congress (AIIoT) (pp. 629-634). Seattle, WA, USA. 10.1109/AIIoT54504.2022.9817270.

[4] Alahassa, N. K.-A., & Murua, A. (2021). The Shallow Gibbs Network, Double Backpropagation and Differential Machine Learning. ScienceOpen Preprints. 10.14293/S2199-1006.1.SOR-.PPS25DJ.v1.

[5] Andreasen, J., & Savine, A. (2018). Modern Computational Finance: Scripting for Derivatives and xVA. Wiley Finance.

[6] Bachelier, L. (1900). Théorie de la spéculation. Annales scientifiques de l'École Normale Supérieure, **3**(17), 21-86.

[7] Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. Journal of Political Economy, **81**(3), 637-659.

[8] Bloch, D. A. (2019). Option Pricing With Machine Learning. Working Paper. Retrieved from: `https://ssrn.com/abstract=3486224` Last accessed 10 Nov 2023.

[9] Bollerslev, T. (1986). Generalized Autoregressive Conditional Heteroskedasticity. Journal of Econometrics, **31**(3), 307-327.

[10] Capriotti, L. (2008). No more bumping: The promises and challenges of adjoint algorithmic differentiation. Invited talk at the Seventh Euro Algorithmic Differentiation Workshop, Oxford University.

[11] Capriotti, L., Jiang, Y., & Macrina, A. (2015). Real-time risk management: An AADPDE approach. 10.2139/ssrn.2630328.

[12] Cox, J. C., Ross, S. A., & Rubinstein, M. (1979). Option Pricing: A Simplified Approach. Journal of Financial Economics, **7**(3), 229-264.

[13] Desjardins, J. (2022). All of the World's Money and Markets in One Visualization. Retrieved from: `https://www.visualcapitalist.com/all-of-the-worlds-money-and-markets-in-one-visualization-2022/`. Last accessed 01 Feb 2024.

[14] Engle, R. F. (1982). Autoregressive conditional heteroskedasticity with estimates of the variance of United Kingdom inflation. Econometrica, **50**(4), 987-1007.

[15] Flyger, H.-J., Huge, B., & Savine, A. (2015). Practical implementation of AAD for derivatives risk management, XVA and RWA. Global Derivatives.

[16] Frandsen, M. G., Pedersen, T. C., & Poulsen, R. (2021). Delta force: Option pricing with differential machine learning. Digital Finance, 4, 1–15. 10.1007/s42521-021-00041-7.

[17] Giles, M., & Glasserman, P. (2005). Smoking Adjoints: Fast evaluation of Greeks in Monte Carlo calculations. Risk, 19.

[18] Goldin, P. (2023). Non-Linear pricing with differential machine learning. Retrieved from: `https://arxiv.org/abs/2311.04178`. Last accessed 01 Dec 2023. [cond-mat.dis-nn].

[19] Hagan, P., Kumar, D., Lesniewski, A., & Woodward, D. (2002). Managing Smile Risk. Wilmott Magazine, 1, 84-108.

[20] Heston, S. (1993). A closed-form solution for options with stochastic volatility with applications to bond and currency options. Review of Financial Studies, **6**(2), 327-343.

[21] Huge, B. N., & Savine, A. (2017). LSM reloaded: Differentiate XVA on your iPad mini. SSRN.

[22] Huge, B. N., & Savine, A. (2020a). A. Differential machine learning. arXiv preprint arXiv:2005.02347. 10.48550/ARXIV.2005.02347.

[23] Huge, B. N., & Savine, A. (2020b). appendices. App1-LSM.pdf. Retrieved from: `https://github.com/differential-machine-learning/appendices`. Last accessed 17 Nov 2023.

[24] Huge, B. N., & Savine, A. (2020c). appendices. App2-Preprocessing.pdf. Retrieved from: `https://github.com/differential-machine-learning/appendices`. Last accesed 17 Nov 2023.

[25] Huge, B. N., & Savine, A. (2020d). appendices. App3-Regression.pdf. Retrieved from: `https://github.com/differential-machine-learning/appendices`. Last accesed 19 Nov 2023.

[26] Huge, B. N., & Savine, A. (2020e). appendices. App4-UnsupervisedTraining.pdf. Retrieved from: `https://github.com/differential-machine-learning/appendices`. Last accesed 19 Nov 2023.

[27] Huge, B. N., & Savine, A. (2021a). Differential Deep Learning [Jupyter Notebook]. GitHub. Retrieved from: `https://github.com/differential-machine-learning/notebooks/blob/master/DifferentialML.ipynb`. Last accesed 13 Jul 2024.

[28] Huge, B. N., & Savine, A. (2021b). Differential PCA and regression on the Bermudan five-factor dataset [Jupyter Notebook]. GitHub. Retrieved from: `https://github.com/differential-machine-learning/notebooks`. Last accesed 03 Jul 2024.

[29] Huge, B. N., & Savine, A. (2021c). Differential Deep Learning [Jupyter Notebook]. GitHub. Retrieved from: `https://github.com/differential-machine-learning/notebooks`. Last accesed 09 Jul 2024.

[30] Hull, J.C. (2003) Options, Futures and Other Derivatives. 5th Edition, Prentice-Hall.

[31] Hull, J., White, A. (1987). The Pricing of Options on Assets with Stochastic Volatilities. The Journal of Finance, **42**, 281-300.

[32] Lapeyre, B., & Lelong, J. (2021). Neural network regression for Bermudan option pricing. Monte Carlo Methods and Applications, **27**(3), 227-247. 10.1515/mcma-2021-2091. [hal-02183587v3].

[33] 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.

[34] Malliaris, M., & Salchenberger, L. (1993a). Beating the best: A neural network challenges the Black-Scholes formula. In Proceedings of 9th IEEE Conference on Artificial Intelligence for Applications; pp. 445–449.

[35] Malliaris, M., & Salchenberger, L. (1993b). A neural network model for estimating option prices. Journal of Applied Intelligence, **3**(2), 193-206.

[36] Markov, A. A. (1906). Extension of the law of large numbers to quantities depending on each other. Izvestiia Fiziko-matematicheskogo obschestva pri Kazanskom universitete, **15**, 135-156.

[37] Merton, R. C. (1976). Option Pricing When Underlying Stock Returns Are Discontinuous. Journal of Financial Economics, **3**(1-2), 125-144.

[38] Merton, R. C. (1973). Theory of Rational Option Pricing. Bell Journal of Economics and Management Science, **4**(1), 141-183.

[39] Rasmussen, M. T. A., Buschardt, S. V. (2022). Option Pricing Using Differential Machine Learning. Master's thesis. Copenhagen Business School.

[40] Samuelson, P. A. (1969). Lifetime Portfolio Selection by Dynamic Stochastic Programming. The Review of Economics and Statistics, **51**(3), 239-246.

[41] Samuelson, P. A. (1960). Proof that properly anticipated prices fluctuate randomly. Industrial Management Review, **6**, 41-50.

[42] Savine, A. (2014). Automatic differentiation for financial derivatives. Global Derivatives.

[43] Savine, A. (2019). Modern Computational Finance: AAD and Parallel Simulations. 1st Edition, Wiley Finance.

[44] Savine, A. (2021). Differential Machine Learning Masterclass. Quant Minds International, Barcelona. Retrieved from: `https://pt.slideshare.net/AntoineSavine/differential-machine-learning-masterclass>`. Last accesed 02 Mar 2024.

[45] Sedo Paredes, M., & Kadry, S. (2020). Pricing European Options with Deep Learning Models. In Proceedings of 2022 Fifth International Conference of Women in Data Science at Prince Sultan University (WiDS PSU), 101-111. 10.1109/WiDS-PSU54548.2022.00033.

[46] Simón i Ribas, A. (2023). An excursion into differential machine learning and applications to finance. Graduation thesis. Universitat Politècnica de Catalunya.

[47] Soini, V., Lorentzen, S. (2019). Option prices and implied volatility in the crude oil market. Energy Economics, 83, 515-539. 10.1016/j.eneco.2019.07.011.

[48] Ulam, S., & von Neumann, J. (1949). "Monte Carlo calculations in problems of mathematical physics." Bulletin of the American Mathematical Society, **53**(12), 1120-1129.

[49] Whaley, R. E. (1993). Derivatives on market volatility. The Journal of Derivatives, **1**(1), 71-84.

Appendices

Appendix A

Activities Planning

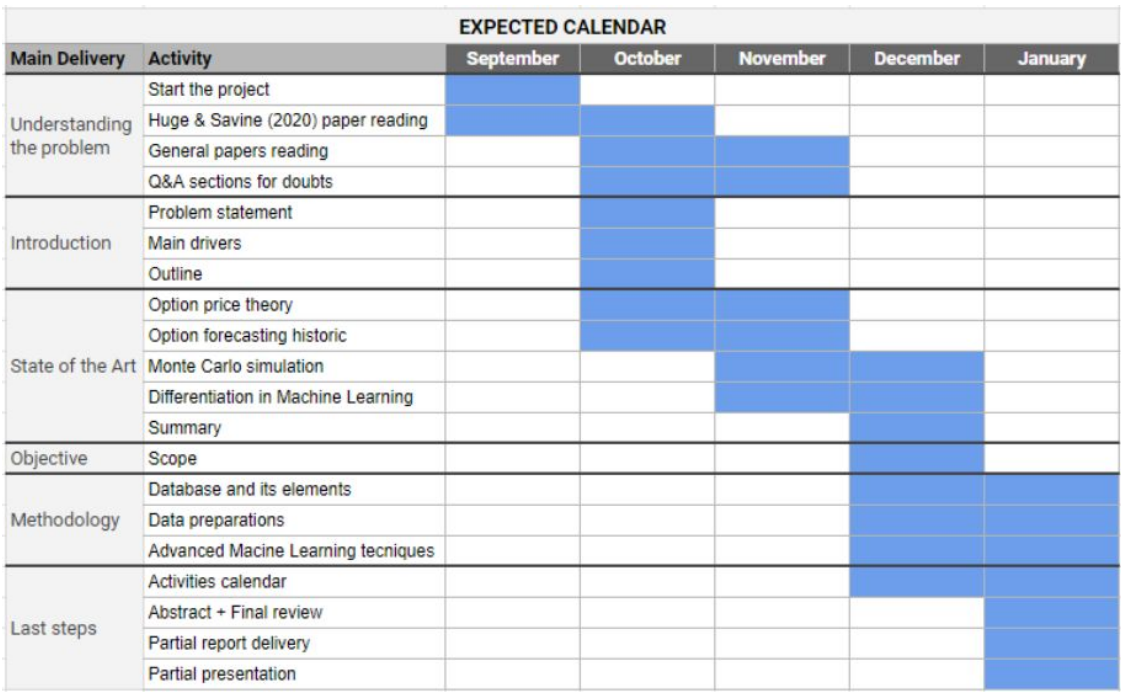
This appendix shows the planning of the thesis activities. The main deliverables and activities are ordered and defined according to the items in the Contents on page ix. In addition, there is an initial and final block, which refer, respectively, to the time dedicated to explore the main topics and finish details of the project.

A.0.1 First Semester Agenda

Figure A.1 shows a GANTT of the activities to be carried out during the first semester, which begins on September 16th and ends, according to the university's official calendar, on February 29th. However, as the presentation of the partial report takes place on January 22th, this is now considered the last month of the first semester agenda.

At the end of September, six main blocks were planned: (i) understand the problem by reading the main paper by Huge & Savine (2020), further reading as the topic progressed and question-and-answer meetings to clarify doubts and present theoretical developments; (ii) writing the Introduction as a results of the first meetings, consisting of identifying the problem, the main drivers and the project outline; (iii) writing the State of the Art from the last question meetings and initially consisting of four themes: option price theory to explain the basis of option markets, option forecasting historic to review most important papers and theories regarding option pricing, Monte Carlo simulation to explore in detail the tool hitherto used by large financial institutions for forecasting, and differentiation in Machine Learning to introduce the basis of the new computational methodology; (iv) writing the Scope of the report, once the theoretical evolution was clear, the practical application of MC and how Diff ML brought results similar to the latter, but with much more efficiency; (v) writing the Methodology, consisting of a description of the data, pre-processing and an in-depth look at Diff ML; and, finally, (vi) the final stages of writing, with the last textual adjustments, delivery and presentation of the partial report.

What is observed, however, is different and is the result of four changes. Figure A.2 shows the final GANTT drawn up at the beginning of January after the methodology had been finalized and the last block of activities had begun.



		EXPECTED CALENDAR				
Main Delivery	**Activity**	**September**	**October**	**November**	**December**	**January**
Understanding the problem	Start the project	🟦				
	Huge & Savine (2020) paper reading	🟦	🟦			
	General papers reading		🟦	🟦		
	Q&A sections for doubts		🟦	🟦		
Introduction	Problem statement		🟦			
	Main drivers		🟦			
	Outline		🟦			
State of the Art	Option price theory		🟦	🟦		
	Option forecasting historic		🟦	🟦		
	Monte Carlo simulation			🟦	🟦	
	Differentiation in Machine Learning			🟦	🟦	
	Summary				🟦	
Objective	Scope				🟦	
Methodology	Database and its elements				🟦	🟦
	Data preparations				🟦	🟦
	Advanced Macine Learning tecniques				🟦	🟦
Last steps	Activities calendar				🟦	🟦
	Abstract + Final review					🟦
	Partial report delivery					🟦
	Partial presentation					🟦

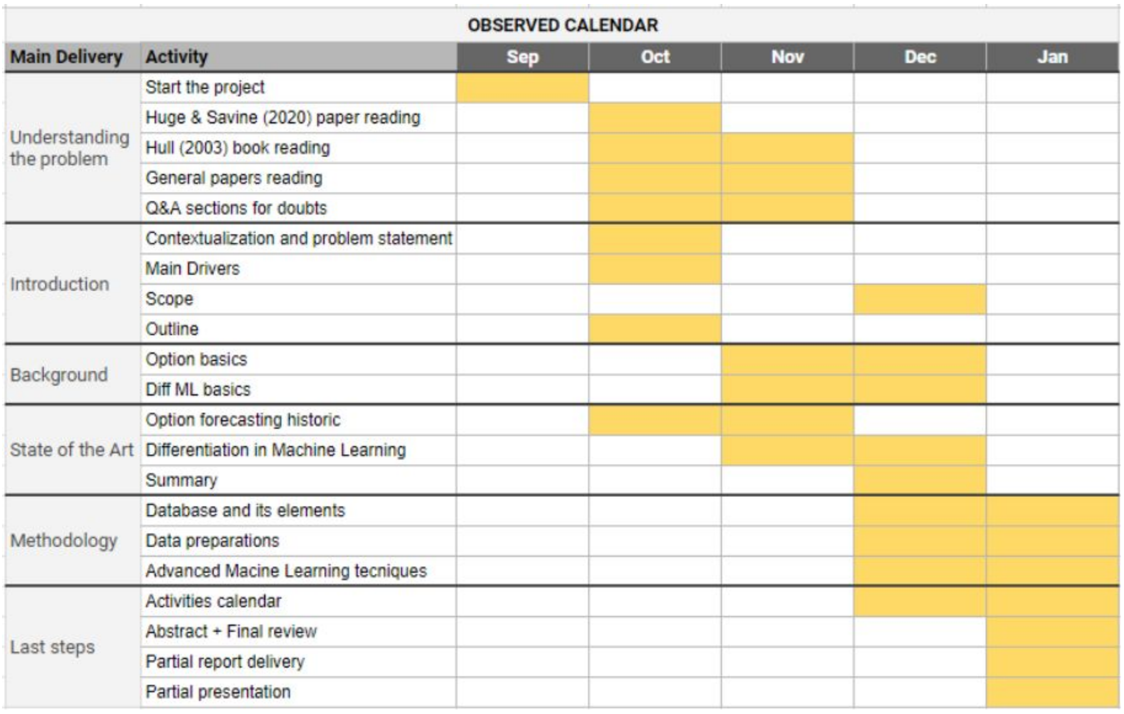
Figure A.1: Expected activities plan for the first semester.
Source: Author's own work.

Talking about the four changes, two relate to the structure and two to the document content. As for the structure, this took place in November and hence the shift observed in GANTT with regard to the following activities: (i) the writing of the Objective is added to the introduction block in order to make the intention of the research clear from the outset; and (ii) State of the Art is divided and it generates a "Background block" before it, where information relating to basic knowledge of both the options market and Diff ML is separated; in addition, MC content now belongs to the list of pricing models within State of the Art first activity. As for the changes to the content: (i) a mandatory reading is added to the first part, the book by Hull (2003), which helped both in the creation of the first Background activity and complemented the first State of the Art activity; and (ii) with the presence of the second Background activity where the whole logic of Diff ML is presented, it is possible to dedicate the second State of the Art activity to an unprecedented task of bringing together all the existing bibliography on the differential technique not only for Finance, but beyond the subject as well.

Two additional events helped in a strong alignment for the contend and the structure this report: two visits to the BNP Paribas CIB office in Lisbon, the first on November 10th, where last remaining doubts were all clarified, and the second on December 18th, when work on the Methodology was explained and started.

A.0.2 Second Semester Agenda

The second semester will primarily center around practicing with Diff ML tools and models. Figure A.3 shows the GANTT of the expected activities. Given that

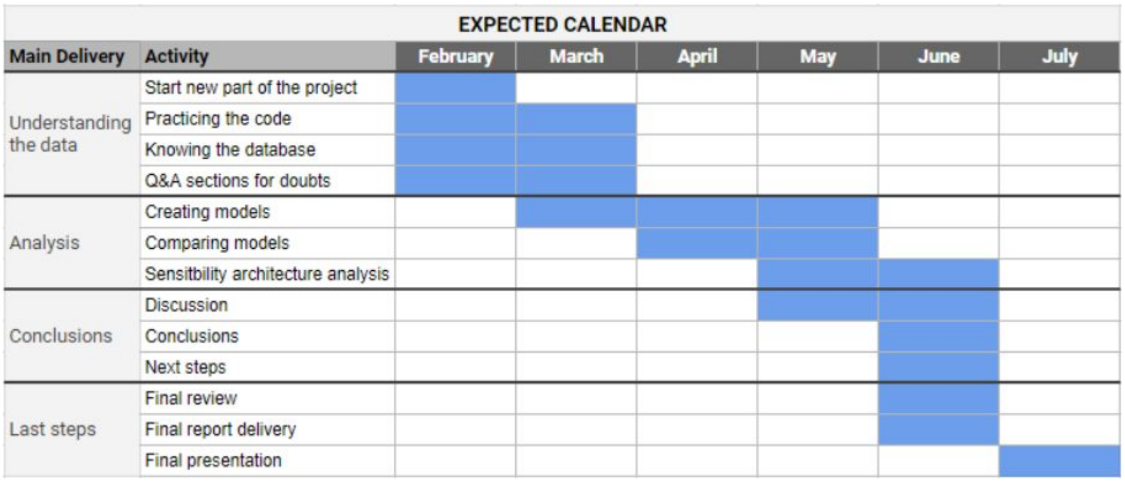


OBSERVED CALENDAR

Main Delivery	Activity	Sep	Oct	Nov	Dec	Jan
Understanding the problem	Start the project	🟨				
	Huge & Savine (2020) paper reading		🟨			
	Hull (2003) book reading		🟨	🟨		
	General papers reading		🟨	🟨		
	Q&A sections for doubts		🟨	🟨		
Introduction	Contextualization and problem statement		🟨			
	Main Drivers		🟨			
	Scope				🟨	
	Outline		🟨			
Background	Option basics			🟨	🟨	
	Diff ML basics			🟨	🟨	
State of the Art	Option forecasting historic		🟨	🟨		
	Differentiation in Machine Learning			🟨	🟨	
	Summary				🟨	
Methodology	Database and its elements				🟨	🟨
	Data preparations				🟨	🟨
	Advanced Macine Learning tecniques				🟨	🟨
Last steps	Activities calendar				🟨	🟨
	Abstract + Final review					🟨
	Partial report delivery					🟨
	Partial presentation					🟨

Figure A.2: Observed activities calendar in the first semester.
Source: Author's own work.

the partial defense takes place on January 22th and that, according to the official calendar, classes begin on February 5th despite defenses's calendar finishing on February 29th, February is considered the beginning of the second semester's work.



EXPECTED CALENDAR

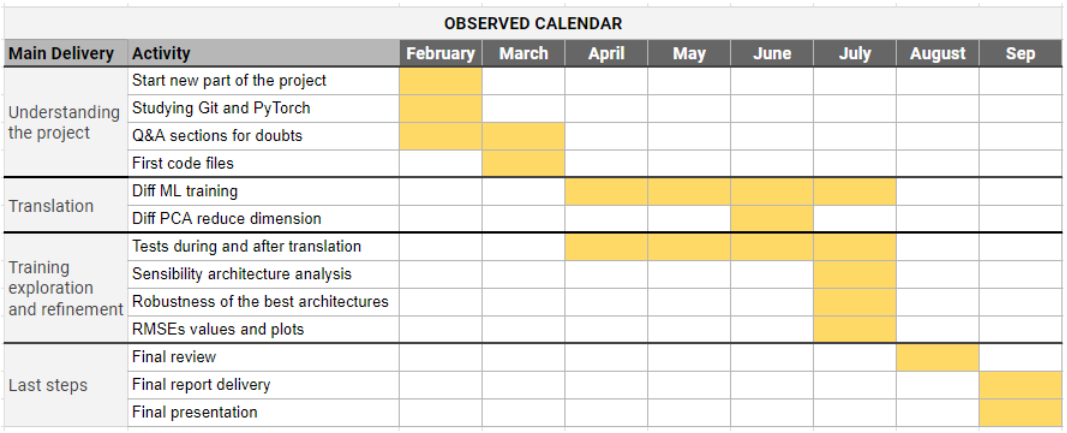
Main Delivery	Activity	February	March	April	May	June	July
Understanding the data	Start new part of the project	🟦					
	Practicing the code	🟦	🟦				
	Knowing the database	🟦	🟦				
	Q&A sections for doubts	🟦	🟦				
Analysis	Creating models		🟦	🟦	🟦		
	Comparing models			🟦	🟦		
	Sensitbility architecture analysis				🟦	🟦	
Conclusions	Discussion				🟦	🟦	
	Conclusions					🟦	
	Next steps					🟦	
Last steps	Final review					🟦	
	Final report delivery					🟦	
	Final presentation						🟦

Figure A.3: Expected activities plan for the second semester.
Source: Author's own work.

Activities were divided into four main deliveries: (i) know the data and understand how PyTorch works and, for that, new Q&A discussions will happen (similarly to the first calendar) to ensure a good understand future work processes and to practice the library; (ii) translation main task, composed of the main code for generating, training and plotting the graphs and the secondary code for Diff

PCA which cannot be completed; (iii) exploration main task, with the improvements made before and after the translation, the evaluation of the architectures with *ceteris paribus* changes and the test of the robustness of the best architectures found; and (iv) like in the first semester, final textual adjustments, delivery and presentation of the final report.

What is observed is a different type of work where there is no work directly with data, but a programming work. Figure A.4 shows the final GANTT for the second semester.



OBSERVED CALENDAR

Main Delivery	Activity	February	March	April	May	June	July	August	Sep
Understanding the project	Start new part of the project	🟨							
	Studying Git and PyTorch	🟨							
	Q&A sections for doubts	🟨	🟨						
	First code files		🟨						
Translation	Diff ML training			🟨	🟨	🟨	🟨		
	Diff PCA reduce dimension					🟨			
Training exploration and refinement	Tests during and after translation			🟨	🟨	🟨	🟨		
	Sensibility architecture analysis						🟨		
	Robustness of the best architectures						🟨		
	RMSEs values and plots						🟨		
Last steps	Final review							🟨	
	Final report delivery								🟨
	Final presentation								🟨

Figure A.4: Observed activities calendar in the second semester.
Source: Author's own work.

Talking about the changes, there was the expectation of working with data from the bank, but it was not possible due to technical problems, so data was generated within the main code.

In addition, the translation task took longer and the Diff PCA subtask was not delivered in order to prioritize the evaluation of architectures. The delay in completing the translation also meant that the thesis had to be submitted in September.

Appendix B

Analytical support

B.0.1 *Ceteris paribus* changes - RMSE plots

This part refers to subsubsection 5.1.2 when it was shown that after 30 to 40 stocks the performance of the differential learning decays. Below, Figure B.1, Figure B.2 and Figure B.3 show RMSEs values, from 1 to 50 stocks, respectively, for a European put under the Bachelier model, a European call under BSMM and a European put under BSMM.

In general, there is a spike in RMSE values for both learning, which means it is difficult to deal with high dimensions. For the European put under the Bachelier model, the 1024 differential learning presents an outlier behavior. For the other two scenarios, between 25 and 30, both learning are more or less the same in terms of *adv* and RMSE.

Again, Bachelier proved to be good in low values of RMSE, while BSMM presents the spikes but is better in terms of *adv*. The same phenomenon happens when it is considered 2 or more baskets of stocks or a maturity bigger than 1.

B.0.2 *Ceteris paribus* changes - Summary table results

Following on from the results in subsubchapter 5.3.3, Table B.1 to Table B.28 show the RMSE and *adv* results for each of the other 28 *ceteris paribus* changes within the seven major groups of components that will change.

Table B.1: 5x20 results. Source: Author's own work.

A2: 5x20 = (20,20,20,20,20)				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	7.66	8.68	13.39	12.05
Stdd 1024	16.18	25.43	90.23	42.84
Diff 8192	4.61	6.49	10.31	10.67
Stdd 8192	8.47	12.82	34.67	22.37
1024 adv	-52.66%	-65.87%	-85.16%	-71.87%
8192 adv	-45.57%	-49.38%	-70.26%	-52.30%
D1024-S8192 adv	-9.56%	-32.29%	-61,38%	-46.13%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	1	1	1	1
Better adv than default architecture?	SAME SAME YES	SAME NO YES	SAME SAME NO	SAME YES YES

Indicator	Bachelier	BSMM
Avg RMSE	11.29	29.57
Avg adv	-42.55%	-64.52%
Avg arch adv	6.49	10.49

Table B.2: 4x15 results. Source: Author's own work.

A3: 4x15 = (15,15,15,15)				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	7.72	9.55	12.94	12.06
Stdd 1024	14.95	25.97	58.68	36.66
Diff 8192	4.42	5.68	10.83	10.15
Stdd 8192	7.76	12.03	24.02	20.18
1024 adv	-48.36%	-63.23%	-77.95%	-67.10%
8192 adv	-43.04%	-52.78%	-54.91%	-49.70%
D1024-S8192 adv	-0.52%	-20.62%	-46.13%	-40.24%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	0	1	1	1
Better adv than default architecture?	NO NO NO	SAME NO NO	NO NO NO	SAME SAME NO

Indicator	Bachelier	BSMM
Avg RMSE	11.01	23.19
Avg adv	-38.09%	-56.01%
Avg arch adv	6.82	10.20

Table B.3: 4x25 results. Source: Author's own work.

A4: 4x25 = (25,25,25,25)				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	11.75	8.73	13.88	12.66
Stdd 1024	16.91	26.13	94.56	48.48
Diff 8192	4.60	5.66	11.07	10.60
Stdd 8192	8.80	13.85	34.44	24.94
1024 adv	-30.51%	-66.59%	-85.32%	-73.89%
8192 adv	-47.73%	-59.13%	-67.86%	-57.50%
D1024-S8192 adv	33.52%	-36.97%	-59.70%	-49.24%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	-1	1	1	1
Better adv than default architecture?	NO	SAME	YES	YES
	SAME	YES	NO	YES
	NO	YES	NO	YES

Indicator	Bachelier	BSMM
Avg RMSE	12.05	31.33
Avg adv	-34.57%	-65.58%
Avg arch adv	7.89	10.78

In scenario A5: 5x25, results are better, on average, 10.15% in *adv* for the Bachelier model and 9.72% for BSMM, and similar in terms of gross RMSE, but the time spent on processing was 94.96% higher for the first and 130.47% higher for the second, so this architecture that follows is not good at all.

Table B.4: 5x25 results. Source: Author's own work.

A5: 5x25 = (25,25,25,25)				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	8.07	8.66	12.55	13.60
Stdd 1024	17.63	26.46	52.92	108.06
Diff 8192	4.56	5.69	9.97	11.25
Stdd 8192	9.41	13.63	25.96	42.59
1024 adv	-54.23%	-67.27%	-76.28%	-87.41%
8192 adv	-51.54%	-58.25%	-61.59%	-73.59%
D1024-S8192 adv	-14.24%	-36.46%	-51.66%	-68.07%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	1	1	1	1
Better adv than default architecture?	YES	SAME	NO	YES
	YES	SAME	NO	YES
	YES	YES	NO	YES

Indicator	Bachelier	BSMM
Avg RMSE	11.76	34.61
Avg adv	-47.00%	-69.77%
Avg arch adv	6.23	10.46

Table B.5: 100 epochs results. Source: Author's own work.

B1: 100 epochs				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	6.85	16.26	12.52	11.34
Stdd 1024	16.37	42.84	28.68	34.25
Diff 8192	4.31	6.83	10.62	10.21
Stdd 8192	6.86	13.29	15.71	16.21
1024 adv	-58.16%	-62.04%	-56.35%	-66.89%
8192 adv	-37.17%	-48.61%	-32.40%	-37.01%
D1024-S8192 adv	-0.15%	22.35%	-20.31%	-30.04%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	1	-1	1	1
Better adv	YES	NO	NO	SAME
than default	NO	NO	NO	NO
architecture?	NO	NO	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	14.20	17.44
Avg adv	-35.56%	-40.50%
Avg arch adv	9.85	10.38

Table B.6: 16 batches results. Source: Author's own work.

C.1 - (16x64, 16x512)				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	8.33	11.01	13.52	12.54
Stdd 1024	16.98	25.03	89.85	46.13
Diff 8192	4.32	5.57	10.74	11.37
Stdd 8192	8.15	12.90	39.47	21.89
1024 adv	-50.94%	-56.01%	-84.95%	-72.82%
8192 adv	-46.99%	-56.82%	-72.79%	-48.06%
D1024-S8192 adv	2.21%	-14.65%	-65.75%	-42.71%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	0	1	1	1
Better adv	SAME	NO	SAME	YES
than default	SAME	YES	SAME	NO
architecture?	NO	NO	SAME	NO

Indicator	Bachelier	BSMM
Avg RMSE	11.54	30.69
Avg adv	-37.20%	-64.51%
Avg arch adv	7.24	10.89

Table B.7: 256 batch size results. Source: Author′s own work.

C2: (4x256, 32x256)				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	10.21	16.56	13.45	13.66
Stdd 1024	17.73	28.71	48.70	36.98
Diff 8192	4.75	5.90	9.27	9.18
Stdd 8192	9.21	16.45	19.22	19.71
1024 adv	-42.41%	-42.32%	-72.38%	-63.06%
8192 adv	-48.43%	-64.13%	-51.77%	-53.42%
D1024-S8192 adv	10.86%	0.67%	-30.02%	-30.70%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	-1	0	1	1
Better adv	NO	NO	NO	SAME
than default	YES	YES	NO	NO
architecture?	NO	NO	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	13.69	21.27
Avg adv	-30.96%	-50.23%
Avg arch adv	9.45	10.59

Table B.8: Softplus() results. Source: Author′s own work.

D1: Softplus()				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	6.77	11.07	11.57	10.59
Stdd 1024	11.07	17.98	30.67	21.95
Diff 8192	4.74	6.25	10.14	11.00
Stdd 8192	5.80	9.77	14.54	14.05
1024 adv	-38.84%	-38.43%	-62.28%	-51.75%
8192 adv	-18.28%	-36.03%	-30.26%	-21.71%
D1024-S8192 adv	16.72%	13.31%	-20.43%	-24.63%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	-1	-1	1	1
Better adv	NO	NO	NO	NO
than default	NO	NO	NO	NO
architecture?	NO	NO	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	12.37	45.37
Avg adv	-26.05%	-18.61%
Avg arch adv	7.63	10.09

Table B.9: ReLU() results. Source: Author′s own work.

D2: ReLU()				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	43.90	104.02	73.14	48.53
Stdd 1024	47.84	117.16	103.97	88.91
Diff 8192	40.26	102.32	70.48	38.68
Stdd 8192	39.43	106.51	83.87	54.22
1024 adv	-8.24%	-11.22%	-29.65%	-45.42%
8192 adv	2.10%	-3.93%	-15.97%	-28.66%
D1024-S8192 adv	11.34%	-2.34%	-12.79%	-10.49%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	0	0	1	1
D 1024 $\succ$ S 8192	-1	0	1	1
Better adv	NO	NO	NO	NO
than default	NO	NO	NO	NO
architecture?	NO	NO	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	75.18	70.23
Avg adv	-0.02	-0.24
Avg arch adv	73.64	53.49

Table B.10: LeakyReLU() results. Source: Author′s own work.

D3: LeakyReLU()				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	43.77	104.41	47.94	73.01
Stdd 1024	47.72	117.21	82.29	102.25
Diff 8192	40.24	102.76	38.51	70.64
Stdd 8192	38.91	106.31	55.10	83.24
1024 adv	-8.28%	-10.92%	-41.74%	-28.60%
8192 adv	3.42%	-3.34%	-30.11%	-15.14%
D1024-S8192 adv	12.49%	-1.79%	-12.99%	-12.29%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	0	0	1	1
D 1024 $\succ$ S 8192	-1	0	1	1
Better adv	NO	NO	NO	NO
than default	NO	NO	NO	NO
architecture?	NO	NO	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	75.17	69.12
Avg adv	-0.01	-0.23
Avg arch adv	74.11	52.89

Table B.11: LeakyReLU(slope=0.05) results. Source: Author's own work.

D4: LeakyReLU(slope=0.05)				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	43.82	104.27	48.38	72.75
Stdd 1024	47.46	115.53	87.89	98.50
Diff 8192	40.22	102.83	38.71	78.90
Stdd 8192	39.17	106.03	51.77	82.08
1024 adv	-7.67%	-9.75%	-44.95%	-26.14%
8192 adv	2.68%	-3.02%	-25.23%	-3.87%
D1024-S8192 adv	11.87%	-1.66%	-6.55%	-11.37%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	0	0	1	1
D 1024 $\succ$ S 8192	-1	0	1	1
Better adv	NO	NO	NO	NO
than default	NO	NO	NO	NO
architecture?	NO	NO	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	74.92	69.87
Avg adv	-0.01	-0.20
Avg arch adv	73.97	56.12

Table B.12: ELU() results. Source: Author's own work.

D5: ELU()				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	14.96	23.34	17.60	21.22
Stdd 1024	32.92	31.53	47.07	32.90
Diff 8192	12.34	23.59	14.51	19.44
Stdd 8192	26.01	26.60	28.68	23.85
1024 adv	-54.56%	-25.98%	-62.61%	-35.50%
8192 adv	-52.56%	-11.32%	-49.41%	-18.49%
D1024-S8192 adv	-42.48%	-12.26%	-38.63%	-11.03%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	1	1	1	1
Better adv	YES	NO	NO	NO
than default	YES	NO	NO	NO
architecture?	YES	NO	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	23.91	25.66
Avg adv	-0.33	-0.36
Avg arch adv	15.97	16.44

Table B.13: ELU(alpha=2) results. Source: Author′s own work.

D6: ELU(alpha=2)				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	9.56	7.32	16.97	11.35
Stdd 1024	21.00	17.21	71.53	30.12
Diff 8192	10.65	7.52	16.43	10.39
Stdd 8192	17.55	10.56	31.13	17.99
1024 adv	-54.48%	-57.47%	-76.28%	-62.32%
8192 adv	-39.32%	-28.79%	-47.22%	-42.25%
D1024-S8192 adv	-45.53%	-30.68%	-45.49%	-36.91%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	1	1	1	1
D 1024 $\succ$ S 8192	1	1	1	1
Better adv than default architecture?	YES	NO	NO	NO
	NO	NO	NO	NO
	YES	SAME	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	12.67	25.74
Avg adv	-0.43	-0.52
Avg arch adv	7.26	12.42

Table B.14: GELU() results. Source: Author′s own work.

D7: GELU()				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	29.68	47.20	29.87	32.77
Stdd 1024	33.03	56.84	67.59	51.19
Diff 8192	25.46	24.14	25.30	20.87
Stdd 8192	27.75	28.22	36.06	28.78
1024 adv	-10.14%	-16.96%	-55.81%	-35.98%
8192 adv	-8.25%	-14.46%	-29.84%	-27.48%
D1024-S8192 adv	6.95%	67.26%	-17.17%	13.86%
1024: Diff $\succ$ Stdd	1	1	1	1
8192: Diff $\succ$ Stdd	0	1	0	0
D 1024 $\succ$ S 8192	-1	-1	1	-1
Better adv than default architecture?	YES	YES	YES	YES
	SAME	YES	YES	YES
	NO	NO	YES	NO

Indicator	Bachelier	BSMM
Avg RMSE	34.04	36.55
Avg adv	0.04	-0.25
Avg arch adv	35.42	27.27

Table B.15: Softmax() results. Source: Author's own work.

D8: Softmax()				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	85.42	365.60	74.20	206.89
Stdd 1024	95.03	365.72	81.36	207.70
Diff 8192	58.35	254.48	52.37	139.20
Stdd 8192	61.24	341.53	57.78	180.96
1024 adv	-10.11%	-0.03%	-8.80%	-0.39%
8192 adv	-4.72%	-25.49%	-9.36%	-23.08%
D1024-S8192 adv	39.48%	7.05%	28.42%	14.33%
1024: Diff $\succ$ Stdd	0	1	0	1
8192: Diff $\succ$ Stdd	1	0	1	0
D 1024 $\succ$ S 8192	1	1	1	1
Better adv than default architecture?	NO SAME NO	SAME YES NO	NO YES NO	SAME YES NO

Indicator	Bachelier	BSMM
Avg RMSE	203.42	125.06
Avg adv	0.01	0.00
Avg arch adv	205.52	125.29

Table B.16: Adagrad() results. Source: Author's own work.

E1: Adagrad()				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	9.16	13.05	13.63	12.79
Stdd 1024	16.74	32.69	49.89	37.16
Diff 8192	5.30	7.16	9.46	9.30
Stdd 8192	9.92	19.75	19.05	20.05
1024 adv	-45.28%	-60.08%	-72.68%	-65.58%
8192 adv	-46.57%	-63.75%	-50.34%	-53.62%
D1024-S8192 adv	-7.66%	-33.92%	-28.45%	-36.21%
1024: Diff $\succ$ Stdd	0	0	0	0
8192: Diff $\succ$ Stdd	0	0	0	0
D 1024 $\succ$ S 8192	1	1	0	0
Better adv than default architecture?	NO NO YES	NO NO YES	NO NO NO	NO NO NO

Indicator	Bachelier	BSMM
Avg RMSE	14.22	21.42
Avg adv	-0.43	-0.51
Avg arch adv	8.12	10.46

Table B.17: Adamax() results. Source: Author's own work.

E2: Adamax()				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	8.48	11.16	13.44	12.55
Stdd 1024	16.23	25.01	68.72	32.38
Diff 8192	4.43	5.48	10.89	10.62
Stdd 8192	8.08	13.07	24.27	21.29
1024 adv	-47.75%	-55.38%	-80.44%	-61.24%
8192 adv	-45.17%	-58.07%	-55.13%	-50.12%
D1024-S8192 adv	4.95%	-14.61%	-44.62%	-41.05%
1024: Diff $\succ$ Stdd	0	0	0	0
8192: Diff $\succ$ Stdd	1	0	0	0
D 1024 $\succ$ S 8192	0	1	0	0
Better adv	NO	NO	NO	NO
than default	NO	NO	NO	NO
architecture?	NO	YES	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	11.49	24.27
Avg adv	-0.36	-0.55
Avg arch adv	7.35	10.82

Table B.18: Adadelta() results. Source: Author's own work.

E3: Adadelta()				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	17.89	32.77	16.01	20.44
Stdd 1024	23.87	49.83	103.82	64.54
Diff 8192	10.91	25.78	12.26	14.80
Stdd 8192	14.34	31.31	33.88	27.95
1024 adv	-25.05%	-34.24%	-84.58%	-68.33%
8192 adv	-23.92%	-17.66%	-63.81%	-47.05%
D1024-S8192 adv	24.76%	4.66%	-52.74%	-26.87%
1024: Diff $\succ$ Stdd	0	0	0	0
8192: Diff $\succ$ Stdd	0	1	0	0
D 1024 $\succ$ S 8192	-1	-1	0	-1
Better adv	NO	NO	NO	NO
than default	NO	YES	NO	NO
architecture?	NO	NO	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	25.84	36.71
Avg adv	-0.12	-0.57
Avg arch adv	22.76	15.70

Table B.19: Author's schema results. Source: Author's own work.

F1: Author's schema				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	39.82	6.72	13.09	16.28
Stdd 1024	39.12	25.01	53.91	82.06
Diff 8192	24.40	2.50	9.14	9.18
Stdd 8192	25.77	9.75	34.15	38.09
1024 adv	1.79%	-73.13%	-75.72%	-80.16%
8192 adv	-5.32%	-74.36%	-73.24%	-75.90%
D1024-S8192 adv	54.52%	-31.08%	-61.67%	-57.26%
1024: Diff $\succ$ Stdd	0	-1	-1	-1
8192: Diff $\succ$ Stdd	1	-1	-1	-1
D 1024 $\succ$ S 8192	-1	0	-1	-1
Better adv	NO	NO	NO	NO
than default	NO	NO	NO	NO
architecture?	YES	YES	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	21.64	31.99
Avg adv	-0.21	-0.71
Avg arch adv	17.04	9.39

Table B.20: Soft schema results. Source: Author's own work.

F2: Soft schema()				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	10.22	9.29	12.66	11.98
Stdd 1024	14.86	30.58	76.80	38.30
Diff 8192	3.01	3.27	8.75	8.82
Stdd 8192	7.60	11.22	29.75	21.11
1024 adv	-31.22%	-69.62%	-83.52%	-68.72%
8192 adv	-60.39%	-70.86%	-70.59%	-58.22%
D1024-S8192 adv	34.47%	-17.20%	-57.45%	-43.25%
1024: Diff $\succ$ Stdd	-1	-1	-1	-1
8192: Diff $\succ$ Stdd	-1	-1	-1	-1
D 1024 $\succ$ S 8192	-1	-1	-1	-1
Better adv	NO	NO	NO	NO
than default	NO	NO	NO	NO
architecture?	NO	YES	YES	YES

Indicator	Bachelier	BSMM
Avg RMSE	11.26	26.02
Avg adv	-0.36	-0.64
Avg arch adv	7.23	9.47

Table B.21: Softer schema results. Source: Author's own work.

F3: Softer schema				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	7.55	8.92	12.00	12.71
Stdd 1024	17.53	28.73	39.44	84.40
Diff 8192	2.97	3.12	8.86	8.78
Stdd 8192	7.59	10.86	21.74	33.66
1024 adv	-56.93%	-68.95%	-69.57%	-84.94%
8192 adv	-60.87%	-71.27%	-59.25%	-73.92%
D1024-S8192 adv	-0.53%	-17.86%	-44.80%	-62.24%
1024: Diff $\succ$ Stdd	-1	-1	-1	-1
8192: Diff $\succ$ Stdd	-1	-1	-1	-1
D 1024 $\succ$ S 8192	-1	-1	-1	-1
Better adv	NO	NO	NO	NO
than default	NO	NO	NO	NO
architecture?	NO	NO	NO	YES

Indicator	Bachelier	BSMM
Avg RMSE	10.91	27.70
Avg adv	-0.46	-0.66
Avg arch adv	5.88	9.48

Table B.22: Even softer schema results. Source: Author's own work.

F4: Even softer schema				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	18.00	7.09	13.75	12.61
Stdd 1024	19.88	23.77	116.13	56.05
Diff 8192	3.02	2.68	9.29	9.11
Stdd 8192	11.52	10.16	60.68	29.75
1024 adv	-9.46%	-70.17%	-88.16%	-77.50%
8192 adv	-73.78%	-73.62%	-84.69%	-69.38%
D1024-S8192 adv	56.25%	-30.22%	-77.34%	-57.61%
1024: Diff $\succ$ Stdd	-1	-1	-1	-1
8192: Diff $\succ$ Stdd	-1	-1	-1	-1
D 1024 $\succ$ S 8192	1	-1	-1	-1
Better adv	NO	NO	NO	NO
than default	NO	NO	NO	NO
architecture?	YES	NO	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	12.02	38.42
Avg adv	-0.34	-0.76
Avg arch adv	7.99	9.31

Table B.23: Constant results. Source: Author′s own work.

G1: Constant = 0.05				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	35.64	55.63	24.44	13.85
Stdd 1024	37.92	74.09	34.88	38.13
Diff 8192	8.51	19.62	13.66	9.61
Stdd 8192	24.94	26.00	18.68	13.23
1024 adv	-6.01%	-24.92%	-29.93%	-63.68%
8192 adv	-65.88%	-24.54%	-26.87%	-27.36%
D1024-S8192 adv	42.90%	113.96%	30.84%	4.69%
1024: Diff $\succ$ Stdd	0	0	0	-1
8192: Diff $\succ$ Stdd	-1	-1	-1	-1
D 1024 $\succ$ S 8192	1	1	1	1
Better adv	YES	YES	YES	NO
than default	NO	NO	NO	NO
architecture?	YES	NO	YES	NO

Indicator	Bachelier	BSMM
Avg RMSE	35.29	20.81
Avg adv	0.06	-0.19
Avg arch adv	37.38	16.91

Table B.24: Uniform results. Source: Author′s own work.

G2: Uniform = (-0.1, 0.1)				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	5.89	19.41	11.73	10.48
Stdd 1024	10.62	31.90	28.71	28.63
Diff 8192	4.04	7.51	10.28	9.87
Stdd 8192	5.14	11.84	15.48	13.79
1024 adv	-44.54%	-39.15%	-59.14%	-63.40%
8192 adv	-21.40%	-36.57%	-33.59%	-28.43%
D1024-S8192 adv	14.59%	63.94%	-24.22%	-24.00%
1024: Diff $\succ$ Stdd	-1	-1	-1	-1
8192: Diff $\succ$ Stdd	-1	-1	-1	-1
D 1024 $\succ$ S 8192	1	1	-1	-1
Better adv	NO	NO	NO	NO
than default	NO	NO	NO	NO
architecture?	YES	YES	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	12.04	16.12
Avg adv	-0.11	-0.39
Avg arch adv	10.78	9.87

Table B.25: Normal results. Source: Author's own work.

G3: Normal = $\mathcal{N}(0, 0.5)$				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	5.76	10.22	11.72	10.75
Stdd 1024	8.96	21.59	29.63	23.69
Diff 8192	3.96	6.95	10.56	10.67
Stdd 8192	5.44	9.61	15.28	14.28
1024 adv	-35.71%	-52.66%	-60.45%	-54.62%
8192 adv	-27.21%	-27.68%	-30.89%	-25.28%
D1024-S8192 adv	5.88%	6.35%	-23.30%	-24.72%
1024: Diff $\succ$ Stdd	-1	-1	-1	-1
8192: Diff $\succ$ Stdd	-1	-1	-1	-1
D 1024 $\succ$ S 8192	0	0	-1	-1
Better adv	NO	NO	NO	NO
than default	NO	NO	NO	NO
architecture?	NO	NO	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	9.06	15.82
Avg adv	-0.22	-0.37
Avg arch adv	7.08	10.04

Table B.26: Xavier uniform results. Source: Author's own work.

G4: Xavier uniform				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	21.01	9.13	23.38	12.43
Stdd 1024	15.21	25.92	69.69	38.85
Diff 8192	17.88	5.26	20.38	9.70
Stdd 8192	8.28	12.61	25.92	20.36
1024 adv	38.13%	-64.78%	-66.45%	-68.01%
8192 adv	115.94%	-58.29%	-21.37%	-52.36%
D1024-S8192 adv	153.74%	-27.60%	-9.80%	-38.95%
1024: Diff $\succ$ Stdd	0	-1	-1	-1
8192: Diff $\succ$ Stdd	1	-1	-1	-1
D 1024 $\succ$ S 8192	1	0	1	0
Better adv	NO	YES	YES	YES
than default	NO	YES	YES	YES
architecture?	NO	YES	YES	YES

Indicator	Bachelier	BSMM
Avg RMSE	14.41	27.59
Avg adv	0.26	-0.43
Avg arch adv	18.19	15.77

Table B.27: Xavier normal results. Source: Author's own work.

G5: Xavier normal				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	7.13	8.08	12.16	11.98
Stdd 1024	12.08	21.85	43.93	30.91
Diff 8192	3.88	5.65	10.59	10.78
Stdd 8192	6.48	10.90	18.56	18.18
1024 adv	-40.98%	-63.02%	-72.32%	-61.24%
8192 adv	-40.12%	-48.17%	-42.94%	-40.70%
D1024-S8192 adv	10.03%	-25.87%	-34.48%	-34.10%
1024: Diff $\succ$ Stdd	0	0	1	1
8192: Diff $\succ$ Stdd	0	0	1	1
D 1024 $\succ$ S 8192	-1	0	-1	-1
Better adv	YES	YES	YES	YES
than default	YES	YES	YES	YES
architecture?	NO	YES	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	9.51	19.64
Avg adv	-0.35	-0.48
Avg arch adv	6.21	10.28

Table B.28: Kaiming uniform results. Source: Author's own work.

G6: Kaiming uniform				
Learning	Bach call	Bach put	BSMM call	BSMM put
Diff 1024	16.78	9.22	31.15	12.62
Stdd 1024	15.89	26.97	72.95	40.71
Diff 8192	13.98	6.00	31.92	9.58
Stdd 8192	9.18	13.46	27.35	21.32
1024 adv	5.60%	-31.50%	-57.30%	-69.00%
8192 adv	52.59%	-55.42%	16.71%	-55.07%
D1024-S8192 adv	82.79%	-31.50%	13.89%	-40.81%
1024: Diff $\succ$ Stdd	-1	1	1	1
8192: Diff $\succ$ Stdd	-1	1	1	1
D 1024 $\succ$ S 8192	-1	1	-1	1
Better adv	NO	SAME	NO	SAME
than default	NO	SAME	NO	YES
architecture?	NO	SAME	NO	NO

Indicator	Bachelier	BSMM
Avg RMSE	13.94	30.95
Avg adv	-0.02%	-0.32%
Avg arch adv	13.65	21.07

B.0.3 Softplus(beta=*f*) beta parameter structure

During the translation main task, there were difficulties in advancing the code. While the existing bugs were not properly fixed, it was shown that the value of the beta parameter was correlated with the number of stocks in the basket and, therefore, in an attempt to improve the forecast, action was taken to validate a structure that described beta, the so-called f function as seen in subsubsection 5.2.1.

The tests were carried out only with a single payoff size to be estimated, 8192, and based on what was observed here, it was applied to the other dimensions. It is known that it would be necessary to carry out tests for at least 1024 observations to confirm the findings and validate the robustness of these values. Because there was little time and focus on translation, these were not carried out.

It starts in Table B.29, with a constant decay of the beta value from 0.1 to 0.1. It is noted that, especially in BSMM, the RMSE values are worse in differential learning (-1), while the expected behavior of it being better (1) occurs much better with Bachelier. Two seeds are used to increase the robustness of the tests, 1234 and 4321.

For this reason, the decay of the beta value is changed to 13, where most of the failures are concentrated and a slightly better result is obtained, as shown in Table B.30.

A new attempt is then made to leave the function continuous. This time, instead of starting at 3.0, it starts at 1.75 from 12 stocks. The results didn't improve (Table B.31):

Finally, a new change is made with a smooth decay starting at 3.0, but with a change in the rule starting at 11, where many failures are concentrated (between 11 and 13). Then a real test is carried out with the help of a third seed, 666, to validate a rule (Table B.32) which proves to be even better.

Table B.29: First beta tests. Source: Author's own work.

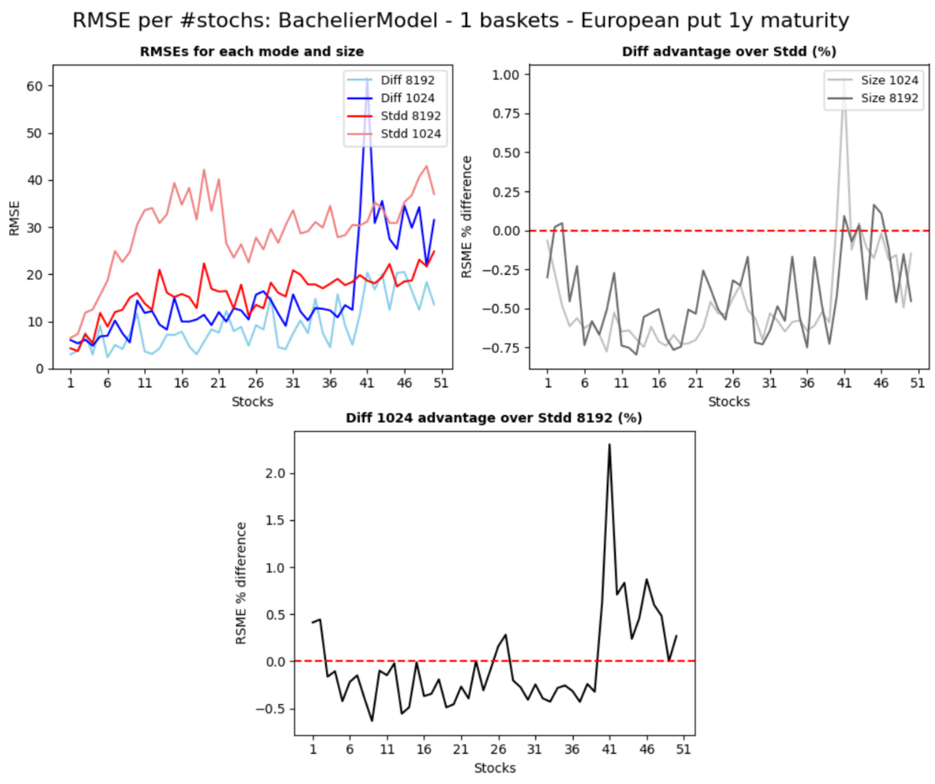


Figure B.1: European put Bachelier model (1-50 stocks).
Source: Author's own work.

Stocks	Beta	Bach 1234	Bach 4321	BSMM 1234	BSMM 4321
1	3.0	0	0	-1	1
2	2.9	1	-1	-1	1
3	2.8	1	-1	-1	-1
4	2.7	1	1	-1	-1
5	2.6	1	0	1	1
6	2.5	1	1	-1	-1
7	2.4	1	1	-1	1
8	2.3	-1	-1	-1	1
9	2.2	1	-1	0	1
10	2.1	-1	1	-1	-1
11	2.0	-1	1	-1	-1
12	1.9	1	1	-1	-1
13	1.8	-1	1	-1	-1
14	1.7	1	-1	1	1
15	1.6	-1	-1	1	1
16	1.5	-1	-1	1	0
17	1.4	-1	1	1	-1
18	1.3	1	-1	1	-1
19	1.2	1	1	1	1
20	1.1	0	1	0	1
21	1.0	1	1	1	1
Mean		0.55		0.48	

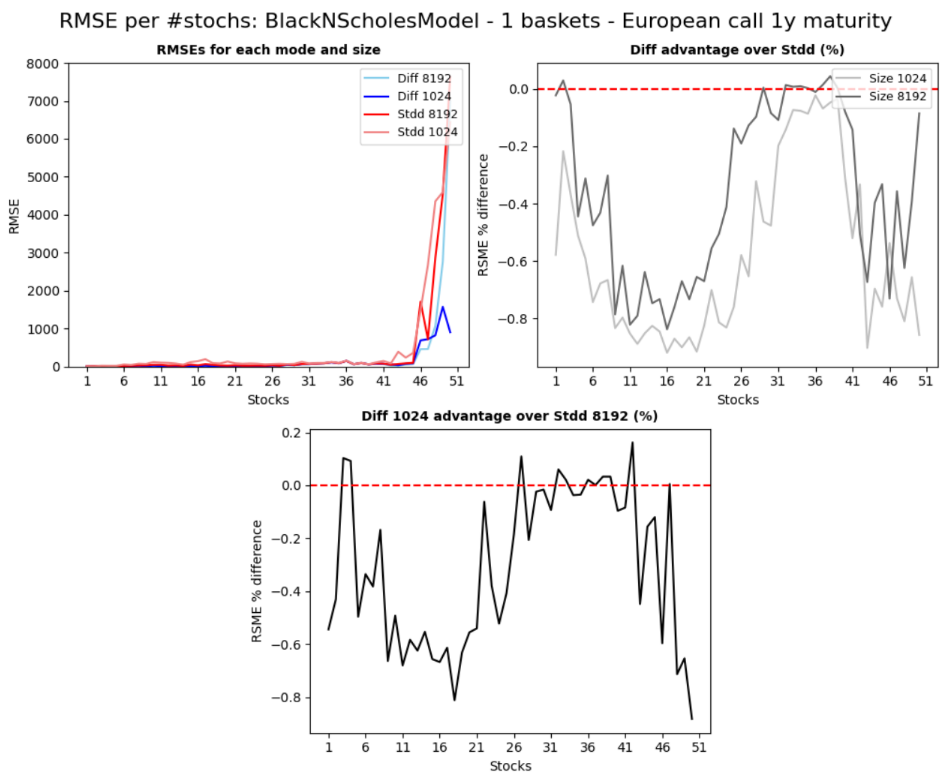


Figure B.2: European call BSMM (1-50 stocks). Source: Author's own work.

Table B.30: Second beta tests. Source: Author's own work.

Stocks	Beta	Bach 1234	Bach 4321	BSMM 1234	BSMM 4321
1	3.0	0	0	-1	1
2	2.95	-1	-1	-1	1
3	2.9	1	-1	-1	-1
4	2.85	0	1	1	-1
5	2.8	1	-1	0	1
6	2.75	1	-1	-1	-1
7	2.7	1	1	-1	-1
8	2.65	1	1	-1	1
9	2.6	1	0	1	1
10	2.55	1	1	0	-1
11	2.5	1	1	-1	-1
12	2.45	-1	1	-1	-1
13	2.4	1	1	-1	1
14	1.7	1	-1	1	1
15	1.6	-1	-1	1	1
16	1.5	-1	-1	1	0
17	1.4	-1	1	1	-1
18	1.3	1	-1	1	-1
19	1.2	1	1	1	1
20	1.1	0	1	0	1
21	1.0	1	1	1	1
Mean		0.57		0.48	

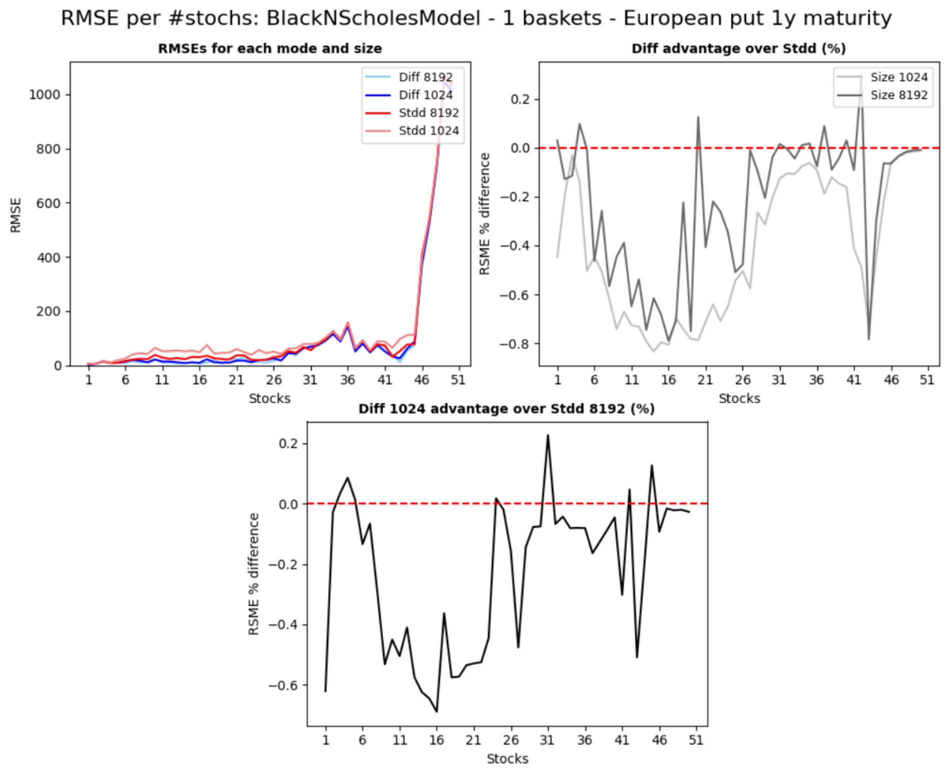


Figure B.3: European put BSMM (1-50 stocks). Source: Author's own work.

Table B.31: Third beta tests. Source: Author's own work.

Stocks	Beta	Bach 1234	Bach 4321	BSMM 1234	BSMM 4321
1	2.35	-1	-1	-1	-1
2	2.3	-1	1	-1	1
3	2.25	0	1	0	-1
4	2.2	0	1	-1	-1
5	2.15 1	-1	-1	1	
6	2.1	1	-1	0	1
7	2.05	1	1	-1	-1
8	2.0	1	1	-1	1
9	1.95	1	0	1	1
10	1.9	1	1	0	-1
11	1.85	-1	1	-1	-1
12	1.8	-1	1	-1	1
13	1.75	-1	1	-1	1
14	1.7	1	-1	1	1
15	1.6	-1	-1	1	1
16	1.5	-1	-1	1	0
17	1.4	-1	1	1	-1
18	1.3	1	-1	1	-1
19	1.2	1	1	1	1
20	1.1	0	1	0	1
21	1.0	1	1	1	1
Mean		0.52		0.48	

Table B.32: Final beta tests. Source: Author's own work.

Stocks	Beta	Bach1234	Bach4321	BS1234	BS4321	Bach66	BS666
1	3.0	0	0	-1	1	0	1
2	2.95	-1	-1	-1	1	1	1
3	2.9	1	-1	-1	-1	-1	-1
4	2.85	0	1	1	-1	0	-1
5	2.8	1	-1	0	1	1	1
6	2.75	1	-1	-1	-1	1	1
7	2.7	1	1	-1	-1	1	-1
8	2.65	1	1	-1	1	1	1
9	2.6	1	0	1	1	-1	1
10	2.55	1	1	0	-1	-1	1
11	2.5	1	1	-1	-1	1	1
12	1.9	1	1	-1	-1	-1	1
13	1.8	-1	1	-1	1	1	1
14	1.7	1	-1	1	1	1	-1
15	1.6	-1	-1	1	1	1	1
16	1.5	-1	-1	1	0	1	-1
17	1.4	-1	1	1	-1	1	1
18	1.3	1	-1	1	-1	-1	-1
19	1.2	1	1	1	1	1	-1
20	1.1	0	1	0	1	1	-1
21	1.0	1	1	1	1	-1	-1
Mean		0.62		0.50		0.62	0.57