

Using Deep Learning to Hedge Rainbow Options

Supervisor: Prof. Thibaud Vienne

Abstract

The general scope of this thesis will be to further study the application of artificial neural networks in the context of hedging rainbow options. Due to their inherently complex features, such as the correlated paths that the prices of their underlying assets take or their absence from traded markets, finding an optimal hedging strategy for rainbow options is difficult, and traders usually have to resort to models and methods they know are inaccurate. An alternative approach involving deep learning however recently surfaced in the context of hedging vanilla options [6], and researchers have started to see potential in the use of neural networks for options endowed with exotic features [5], [11] and [20].

The key to a near-perfect hedge for contingent claims might be hidden behind the training of neural network algorithms [6], and the scope of this research will be to further investigate how those innovative hedging techniques can be extended to rainbow options [20], using recent research [19], and to compare our results with those of the current models and techniques used by traders, such as *Monte-Carlo* path simulations. In order to accomplish that, we will try to develop an algorithm capable of designing an innovative and optimal hedging strategy for rainbow options using some intuition developed to hedge vanilla options [19] and price exotics [5]. But although it was shown from past literature to be potentially efficient and cost-effective, the opaque nature of an artificial neural network will make it difficult for the deep learning algorithm to be fully trusted and used as a sole method for hedging purposes, but rather as an additional technique associated with other models.

Table of contents

1	Introduction	3
2	Mathematical prerequisites	6
2.1	Agent's terminal wealth	6
2.2	Convex risk measures	8
2.3	Optimized certainty equivalents	9
2.4	Additional risk measures to be used	10
3	Neural Networks	11
3.1	Understanding key notions	11
3.1.1	Deep learning architecture	11
3.1.2	Activation functions	12
3.1.3	Recurrent neural networks	13
3.1.4	Stochastic gradient descent algorithms	13
3.2	Validity and power of the method	14
3.3	Deep learning in the context of hedging	16
4	Deep Hedging Vanillas	17
4.1	Fundamental derivatives pricing theory	17
4.1.1	Black-Scholes model	17
4.1.2	Monte-Carlo method	18
4.1.3	Heston model	20
4.2	An extensive literature review	21
4.3	Results and discussion	22
4.3.1	Brief overview of the research design	22
4.3.2	Outcome and benchmark comparison	22
4.4	Introducing Heston stochastic volatility	23
4.4.1	Context and relevance of the research	23

4.4.2	Results and conclusions	23
5	Deep Hedging Rainbows	24
5.1	Financial Theory	24
5.1.1	Defining correlation options	24
5.1.2	Multi-asset Brownian motion extension	25
5.1.3	Price and sensitivities of rainbows under Stulz	26
5.1.4	Multi-asset dynamics with jump-diffusion under Kou	28
5.2	An extensive literature review	30
5.3	Results and discussion	30
5.3.1	Brief overview of the research design	30
5.3.2	Outcome and benchmark comparison	31
5.4	Multivariate case jump-diffusion extension	31
5.4.1	Context and relevance of the research	31
5.4.2	Results and conclusions	31
6	Conclusion and discussion	32
6.1	Research outcome	32
6.2	Limits of the deep learning method	32
6.3	Further research	33
6.3.1	Considering stochastic correlation	33
6.3.2	Volatility forecasts through quantile regression	33
	Bibliography	35

Chapter 1

Introduction

The most crucial step in dealing with any asset is to be capable of determining its price, usually referred to in the literature as the *present value*. That price must reflect the reality as closely as possible, because for widely-traded instruments such as options, even the smallest price discrepancy can have severe consequences on the profits and losses made by traders. European options are often evaluated using the *Black-Scholes* pricing model [3], which calculates the price of an option as the discounted expected value of its payoffs, under a risk neutral probability measure. The model is arbitrage-free, meaning that the price of each asset in the market is unique. Under *Black-Scholes*, the price of an option is dependent on several input parameters, such as the option's time to maturity or the volatility of its underlying's spot price. The evolution of those parameters thus has an influence on the *present value* of the evaluated contingent claim, and each option reacts differently to a variation of one of its inputs. These reactions are characterized by coefficients that are commonly referred to as the *Greeks*, and as it is often preferable for the value of a portfolio to be stable over time, traders usually build theirs so that the *Greeks* are negligible. This process is referred to as *hedging*. Hedging is thus buying or selling additional instruments to make a portfolio neutral to one or several input variations, because the *Greeks* of the additional assets cancel out with those of the initial portfolio of options.

Because those sensitivities change constantly during an option's life, traders need to monitor and rebalance their portfolio to maintain their hedge accordingly, but the presence of trading costs prevents the possibility for the hedge to be realized continuously, so it is done periodically. Figuring out exactly the price and *Greeks* of an option at any moment is therefore of paramount importance in that context, because any miscalculation could result in a sub-optimal hedging strategy.

This comes down to choosing the most convenient model to begin with, as each model is endowed with a specific set of assumptions that will influence the degree to which the solution is or is not well approximated. For certain types of derivatives such as vanilla options, the present value can be calculated analytically as a closed-form solution. Pricing options with more exotic features does not allow for such, and numerical approximations, such as *Monte-Carlo* [4] simulations, are often required. Those methods have however proven to be computationally expensive and tremendously slow to run when there is some complexity involved, and in a context where the parameters of an option are updated regularly, Monte-Carlo appears to be rather unreliable.

Our study will have a particular focus on rainbow options, which are exotic derivatives relying on the performance of more than a single underlying asset. These instruments are of interest not only because of their popularity, but also because their dependence over multiple underlying assets with correlated paths makes it difficult to define an optimal hedging strategy that is in line with real-world dynamics, especially since current methods cannot account for very sudden and extreme market moves, forcing traders to use their experience and intuition to manually re-calibrate their strategy, which has gotten increasingly difficult with the growing sophistication of financial markets. This explains why traders have been looking for an innovative method that would assist them in reaching an optimal hedging strategy for rainbow options, and improvements could be foreseen through the use of deep learning and artificial neural networks [6], an architecture capable of learning key representations in the data without the need for an underlying model. The application of neural networks to hedging and pricing purposes was proven successful in several research papers, and was initiated by [15], who demonstrated an ability of deep learning in pricing futures options.

The studies of [9] then illustrated how neural networks could outperform standard hedging models for American options, and [6] presented an extensive theoretical framework for hedging vanilla options in the presence of market frictions, which was later applied by [19], who analyzed the calibration of the algorithm to various stochastic paths for the asset prices, with interesting results. Studies diving further in the use of deep learning for exotic derivatives include [20], who used the *regress later neural network* technique to price and semi-statically hedge high-dimensional contingent claims such as *Bermudan max and basket options*, which came after the works of [5], whose paper initiated research for options relying on several underlyings, namely *rainbow options*. Those were further explored by

[11], who illustrated the feasibility of the deep learning approach through the valuation of an option relying on six underlying assets, as well as its computational power when compared to classic *Monte-Carlo* simulations, showing significant improvements, both in speed and accuracy.

This paper will partly aim at extending the most recent literature made on using deep learning in hedging vanillas and pricing exotics, to the problem of hedging rainbow options in a regular market context. Beyond designing a strategy that would be in line with real-world dynamics, using deep learning could also allow us to eliminate constraints that weight heavily in the accuracy of our results, such as the absence of sudden and extreme market moves. We will therefore try to implement the algorithm in a context that is closer to market reality, to investigate the potential of deep learning in areas that could not have been explored by current methods, due to the high degree of complexity and computational power that was required.

The thesis shall be structured as follows. We will successively cover prerequisites in financial mathematics (*Chapter 2*) as well as in general deep learning concepts (*Chapter 3*), to lay down the basic foundations of our research. Then, as we begin building and testing our neural network algorithm, we will first produce and interpret the hedging strategy designed by the algorithm for European Put options, using *Black-Scholes* as a benchmark (*Chapter 4*). Having shown the robustness and efficiency of the method for vanilla options, we will then derive an adjustment of the algorithm for rainbow options (*Chapter 5*), in which we shall also test the accuracy of the method in the presence of large and unexpected market moves, to assess whether the neural networks are able to properly process such information to adapt the hedging parameters accordingly. To conclude, we shall condense our results and infer on the practical feasibility and usefulness of the method, and we shall also further discuss issues and limits encountered with deep learning algorithms applied in such context, as well as providing potential clues for further developments (*Chapter 6*).

Chapter 2

Mathematical prerequisites

In the following section, we will discuss the mathematical properties and foundations which will be essential to our investigation [2]. We will successively introduce the formula for an agent's terminal wealth, considering he holds a portfolio of derivatives and trades on its behalf. We will then introduce key definitions regarding convex risk measures and the indifference price, for which the use in our deep learning algorithm shall be explained along the way. Finally, we remind important definitions of some probability laws that will be used in this paper.

2.1 Agent's terminal wealth

We will need to consider a discrete-time financial market with trading dates comprised in the set $\{0, t_1, t_2, \dots, t_n\}$ with $t_n = T$ our finite time horizon, as well as a complete probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with $\Omega = \{\omega_1, \dots, \omega_n\}$ and \mathbb{P} the risk-neutral probability measure, which is not the observed measure, but an indetical one that is fitted for arbitrage-free pricing. Let us also define the set of all random variables over Ω to be \mathbb{Q} , such that $\mathbb{Q} := \{Q : \Omega \rightarrow \mathbb{R}\}$.

Let us denote I_i any new piece of information that becomes available at time t_i . This may include mid-prices of liquid instruments, trading signals and more. All information available up to a certain time t_i is thus given by the filtration $\mathbb{F} = (\mathcal{F}_i)$ for $i = 0 \dots n$, which is generated by the process $I = (I_i)$ for $i = 0 \dots n$.

We will then assume that m instruments exist in the market for us to hedge our contingent claim, and that their mid-prices follow an \mathcal{F} -adapted process $\mathbb{S} = (S_i)$

for $i = 0 \dots n$. Those additional instruments can either be equities or European options, and their maximum maturity is T . The portfolio of contingent claims we intend to hedge will be denoted Z . The position we will take in the hedging instrument t is given by the \mathcal{F} -adapted stochastic process $\delta = (\delta_i^t)$ for $i = 0 \dots n$ and $t = 1 \dots m$. Finally, we define the set of all such trading strategies to be \mathcal{H} .

We note the profits made from trading the hedging instrument at time i to be $(\delta \cdot S)_i$, which gives the following total gains and losses until maturity.

$$(\delta \cdot S) = \sum_{i=0}^{n-1} \delta_i \cdot (S_{i+1} - S_i)$$

Trading is costly through many ways: bid-ask spreads or liquidity constraints for example. Let us define by c_i the cost of taking a position at time t_i on any asset. The total cost of trading is then defined as the following.

$$C(\delta) = \sum_{i=0}^n c_i \cdot (\delta_i - \delta_{i-1})$$

To simplify, let us assume that no position is held in hedging instruments outside the set $\{t_1, t_2, \dots, t_{n-1}\}$. Finally, the self-financing strategy starts with an initial lump sum p_0 . The agent's terminal wealth is then defined as the following [6].

$$P_n L(Z, \delta, p_0) = p_0 - Z + (\delta \cdot S) - C(\delta) \quad (2.1)$$

The goal is to minimize said equation, being the part of our position that we were not able to hedge. In other terms, it is our final *risk exposure*. We shall see that the objective of the algorithm will be to find a *hedging strategy* δ such that the uncovered risk is minimal. Note that this equation can be extended to a *hedging strategy* on two underlyings. Indeed, in *Chapter 5* we shall study options relying on two underlying assets, and we would want to hedge against both their moves.

$$P_n L(Z, \delta_1, \delta_2, p_0) = p_0 - Z + (\delta_1 \cdot S_1) - C(\delta_1) + (\delta_2 \cdot S_2) - C(\delta_2) \quad (2.2)$$

Let us finally remark that we would be able to perform such strategy with or without trading costs, although they were usually implemented for *vanilla option* hedging in the past literature with [6], [8] and [19].

2.2 Convex risk measures

We introduce *monetary risk measures* denoted ρ , which are statistical measures used to assess the risk inherent to an investment, taking the form of a cash requirement. We then have that for any portfolio of assets X , $\rho(X)$ can be viewed as the minimal amount of cash required for the risk of the position to be in an acceptable range, in light of the agent's preferences. We will therefore materialize the aforementioned *unhedged* part of our position by said risk measure, which our algorithm will try to minimize. To guarantee that the solution we find is close-to-optimal, we choose to work with *convex risk measures*, which are classic risk measures satisfying the following properties. Once again, this minimization program can be extended to a strategy involving two underlying assets.

Definition 2.1 *We let X_1 and $X_2 \in \mathcal{X}$ be positions in assets. We let $\rho : \mathcal{X} \rightarrow \mathbb{R}$ be a convex risk measure if the following properties are verified.*

- $X_1 \geq X_2 \rightarrow \rho(X_1) \leq \rho(X_2)$ (*monotone decreasing*)
- $\rho(\beta X_1 + (1 - \beta)X_2) \leq \beta \rho(X_1) + (1 - \beta)\rho(X_2)$ (*convex*)
- $\rho(X_1 + c) = \rho(X) - c, \quad \forall c$ (*translation invariance*)

Having introduced a proper way of quantifying the risk of our portfolio then allows us to introduce the agent's optimization program. Let us denote by $\pi(X)$ the minimal lump sum required by the agent's preferences for the position in a portfolio of financial assets X to be sufficiently well hedged, \mathcal{H} the set of possible trading strategies, and $\delta \in \mathcal{H}$ the minimizing strategy for the program [6].

$$\pi(X) = \left(\inf_{\delta \in \mathcal{H}} \rho(X + (\delta \cdot S) - C(\delta)) \right) \quad (2.3)$$

We thus have that the minimal cash injection π that is required to make a portfolio X acceptable regarding an agent's preferences is equal to the minimal lump sum required to make the *hedged-portfolio* acceptable in light of those same preferences, with $\delta \in \mathcal{H}$ the hedging strategy allowing such. In (2.2), π is monotone decreasing and translation invariant. Given that the function C and the set \mathcal{H} are convex, π then is a convex risk measure [6].

Finally, as we need to consider the scenario in which having no liabilities may generate positive expected returns, which could happen under the physical mea-

sure, we decide to focus on the *indifference price* denoted $p(Z)$. This price is such that holding $p(Z)$ and a position $-Z$ gives the same utility as holding nothing.

$$\pi(p(Z) - Z) = \pi(0) \rightarrow p(Z) = \pi(-Z) - \pi(0) \quad (2.4)$$

This result is given by translation invariance, as π is a convex risk measure.

2.3 Optimized certainty equivalents

In this study, we want to use a specific class of convex risk measures denoted *optimized certainty equivalents* (or *OCEs*), to determine the minimal amount of cash required to make a portfolio of contingent claims acceptable in light of some agent's preferences. We denote X such a portfolio and $r \in \mathbb{R}$ the minimizing parameter of our program.

Definition 2.2 *We let $l: \mathbb{R} \rightarrow \mathbb{R}$ be a convex risk measure and denote it our loss function. The OCE is then defined as the following.*

$$\rho(X) = \inf_{r \in \mathbb{R}} \{r + \mathbb{E}[l(-X - r)]\}, \quad X \in \mathcal{X} \quad (2.5)$$

Following [6] intuition, the *expected shortfall* can be used as a convex risk measure for our algorithm. It can be viewed as an average or conditional *value at risk*, and shall be defined as such below.

Definition 2.2 *Considering $VaR_\gamma(X) = \inf\{m \in \mathbb{R} : \mathbb{P}(X < -m) \leq \gamma\}$, we have that the expected shortfall for any random variable X is defined as such.*

$$\rho(X) = \frac{1}{1-\lambda} \int_0^{1-\lambda} VaR_\gamma(X) d\gamma \quad (2.6)$$

Another convex risk measure that we will consider in this study is the *entropic risk measure*, and it can be defined as the following.

Definition 2.3 *For a fixed and positive λ we set $l(x) = e^{\lambda x} - \frac{1+\ln(x)}{\lambda}$. Optimizing and using (2.4) gives us the entropic risk measure.*

$$\rho(X) = \frac{1}{\lambda} \ln (\mathbb{E}[e^{-\lambda X}]) \quad (2.7)$$

Such risk measure has for long been quite popular in financial mathematics because of some interesting properties it is endowed with, as it depends on an agent's risk aversion, and can be modelled through exponential utility functions, which are defined as $U(x) = -e^{-\lambda x}$ with λ the positive *risk aversion coefficient*. We notice how the aforementioned utility function appears in the entropic risk measure (2.7), but let us solidify the relationship between the two with the following proposition [6].

Proposition 2.4 *Let $q(Z)$ be a solution of the following indifference pricing problem. Then it follows that $q(Z) = p(Z)$, the latter being the indifference price.*

$$\sup_{\delta \in \mathcal{H}} \mathbb{E}[q(Z) - Z + (\delta \cdot S) - C(\delta)] = \sup_{\delta \in \mathcal{H}} \mathbb{E}[U\{(\delta \cdot S) - C(\delta)\}]$$

This result means that if we receive $q(Z)$ of cash in exchange for letting Z go, we are not better nor worse off compared to not selling Z at all.

2.4 Additional risk measures to be used

Several other metrics were used in the past literature to quantify the performance of the deep learning approach, especially when applied to exotic derivatives [5]. We choose to define two additional measures that shall also be used as *loss functions* for our algorithm. However, unless stated otherwise, the *loss function* that is to be used for our calculations is the *expected shortfall* defined in (2.6).

Let us successively define the *mean squared error* (MSE) and the *mean absolute percentage error* (MAPE), which are to be used to evaluate the spread between the predicted value of the *hedging parameter* P , and its actual value A .

$$\text{MSE}(P) = \frac{1}{T} \sum_{i=1}^T (A_i - P_i)^2 \quad (2.8)$$

$$\text{MAPE}(P) = \frac{1}{T} \sum_{i=1}^T \left| \frac{A_i - P_i}{A_i} \right| \quad (2.9)$$

Chapter 3

Neural Networks

This chapter introduces key notions to understand the general concept behind deep learning and neural networks. First of all, we will define in clear terms how such a technology functions and what we should expect of the algorithm to produce. We shall then dive deeper into the mathematical foundations behind its utilisation, to finally detail how we will precisely try to approach the problem of deep hedging, both for vanillas (Chapter 4) and for rainbow options (Chapter 5). The neural network algorithm shall be studied using Python and the deep learning framework Keras, with Tensorflow as a backend engine [7].

3.1 Understanding key notions

3.1.1 Deep learning architecture

Deep learning algorithms are structures endowed with the ability to learn features and patterns directly from data they are trained on, in order to figure out similar patterns on resembling new data they have never seen. Through successive layers of neurons which are connected by channels to form the *neural network*, data is analyzed by the algorithm and patterns are understood.

The *training set* is defined as the set of inputs and their respective expected outputs which are initially fed to the algorithm. Through this first set, the algorithm will try and figure out what are the most fundamental and important representations behind the data. The *testing set* is then given as a way of assessing whether the neural network managed to seize and project those patterns on new but similar data. Measuring its performance is thus done through the

loss function, which compares the actual outcome of the algorithm with what we expect to obtain. The loss function that shall be used is the *expected shortfall* defined in (2.6), unless stated otherwise.

Learning is done through iteration, until the loss function is minimized. Inputs are initially fed into the network through first layers of neurons, called the *input layers*. They are then transferred to subsequent *hidden layers*, whose role is to decipher key representations of the data. Each neuron in those layers has its own importance in the algorithm, meaning that the weights attached to the inputs of each neuron, as well as their bias, is different. These parameters are randomly initialized and iteratively adjusted through *back-propagation*, which is what allows neural networks to learn by themselves, by evaluating their performance using the loss function. The *weights* are allocated to each input entering a given neuron, and the *bias* term skews the result of the neuron's output, modelling a systemic error stemming from incorrect assumptions on the model. To resume, the algorithm will iteratively rearrange its structure until the output is judged sufficiently close to what it is expected to produce.

We just introduced the feed-forward neural network, a structure where neurons only communicate horizontally. Such will form the basic element in our research and is defined as the following.

Definition 3.1 *Let $L, N_0, \dots, N_L \in \mathbb{R}, \sigma_i : \mathbb{R} \rightarrow \mathbb{R}$ and let A_i be affine functions, for any $i \in \{1, 2, \dots, L\}$. The function $\mathcal{F} : \mathbb{R}^{N_0} \rightarrow \mathbb{R}^{N_L}$ below is called a feed-forward neural network, with $\mathcal{F}_i = \sigma_i \circ A_i$ for any $i \in \{1, 2, \dots, L - 1\}$.*

$$\mathcal{F}(x) = A_L \circ \mathcal{F}_{L-1} \circ \dots \circ \mathcal{F}_1 \quad (3.1)$$

3.1.2 Activation functions

We define σ_k as the *activation function* applied to the neuron k . Such function decides on the degree of activation of the node, depending on the accuracy and relevance of its output. Denoting b the bias and w the weights associated with the inputs x gives us the following final output o_k of a given neuron k .

$$o_k = \sigma \left(\sum_{i=1}^n (w_i x_i) + b_k \right)$$

The choice for an activation function is crucial, and is made by evaluating the trade-off between simplicity of implementation and powerfulness. Indeed, complex activation functions allow nodes to learn complex structures in the data. Nonlinear functions are examples of such functions, whereas linear ones are easier to implement and faster to run. Both accumulate their share of advantages and shortcomings, so our idea will be to use one that is a mix of both.

Let us work with an activation function that resembles a linear function, while in reality being a nonlinear one, allowing for complex relationships in the data to be learned. Such is called the *rectified linear activation unit (ReLU)* [7]. That function is often referred to as *piecewise linear*, for it is half linear and half nonlinear. It is defined as the positive part of its input: $\sigma(x) = \max(0, x)$. One can therefore see that the gradient is zero when x is negative, and the neuron becomes inactive.

3.1.3 Recurrent neural networks

We note L the total number of layers and N_i the dimension of the layer i . Finally, let us consider $L-1$ hidden layers, one input and one output layer. We would want our algorithm to remember to some extent what happened in the past, because we need to consider market frictions. In that sense, we could extend our regular feed-forward neural network so that a temporal dynamic behavior can appear. This would be done through the use of *recurrent neural networks* (RNNs), where neurons use their memory regarding past information in their decision-making process. Such an algorithm is structured as an ensemble of regular feed-forward neural networks where information is passed both horizontally to the next hidden layer, and vertically to a similar hidden layer at the next time period. For the rest of this paper, we shall however restrain ourselves to *simple* neural networks, and leave *recurrent* ones for further exploration. To conclude, we shall explain the mechanism that allows the algorithm to adjust its weights and biases, to converge towards their final values.

3.1.4 Stochastic gradient descent algorithms

In our model, we will use the *Adam* algorithm [16], which is defined as a *stochastic gradient descent algorithm* with advantageous properties. As was explained earlier in that sub-section, the neural network algorithm works under a loss function that is aimed to be globally minimized, explaining why the function needed to be

differentiable. A stochastic gradient descent algorithm loops through the network and uses at each iteration the set of first partial derivatives ∇ of the loss function $l(\theta)$ that was calculated using *back-propagation*, and iteratively takes a step downhill using the gradient. This means that the next point $\theta^{(i+1)}$ taken shall always be lower than the previous one $\theta^{(i)}$, in order to reach a global minimum for the loss function. The starting point is often randomly initialized.

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla l_i(\theta^{(i)})$$

The key idea behind such a process is to quantify the step taken by the algorithm to move downhill the loss function. The step η is called the *learning rate*, and it is a small fixed hyper-parameter for regular gradient descent algorithms. For the *Adam algorithm* however, the learning rate of each input parameter is being individually and adaptively optimized, and under suitable assumptions on the loss function l as well as on the sequence $\{\eta_i\}$ for $i \in \mathbb{N}$, $\theta^{(i)}$ converges to a local minimum of l as $i \rightarrow \infty$, with dynamics for updating parameters resembling what we already saw above, but using instead the first two moments of the gradient ∇ respectively denoted m and v , with α and β being *forgetting factors*, and ϵ a small scalar preventing any division by zero.

$$m_{\theta}^* = \left(\frac{1}{1 - \alpha} \right) \left(\alpha m_{\theta}^{(i)} + (1 - \alpha) \nabla l_i(\theta^{(i)}) \right)$$

$$v_{\theta}^* = \left(\frac{1}{1 - \beta} \right) \left(\beta m_{\theta}^{(i)} + (1 - \beta) (\nabla l_i(\theta^{(i)}))^2 \right)$$

$$\theta^{(i+1)} = \theta^{(i)} - \eta \frac{m_{\theta}^*}{\sqrt{v_{\theta}^*} + \epsilon}$$

Combining the *stochastic gradient descent* algorithm with the *back-propagation* error algorithm makes the process of computing ∇ efficient, and is at the core of the deep learning approach [7].

3.2 Validity and power of the method

Having clearly defined the main concepts attached to the deep learning approach, let us now introduce mathematical proof for both the power and the validity of such method. Let us coin as $\mathcal{NN}_{\infty,e,f}^{\sigma}$ the set of all neural networks mapping

from $\mathbb{R}^e \rightarrow \mathbb{R}^f$ with activation function σ . Let us denote as $\mathcal{NN}_{M,e,f}^\sigma$ the one with at most M non-zero weights and a potentially infinite number of layers. The following theorem [6] proves the validity of the method, as it shows the algorithm is able to provide us with the minimal cash injection that the agent must supply to hedge his claim given his preferences, while the number of non-zero weights in the neural network tends towards infinity.

Theorem 3.2 *Let \mathcal{H}_M denote the subspace of all strategies δ that can be obtained with the use of neural networks belonging to $\mathcal{NN}_{M,e,f}^\sigma$ with X an asset position and ρ and π^M convex risk measures.*

$$\lim_{M \rightarrow +\infty} \pi^M(X) = \lim_{M \rightarrow +\infty} \left(\inf_{\delta \in \mathcal{H}_M} \rho(X + (\delta \cdot S) - C(\delta)) \right) \quad (3.2)$$

An important theorem [14] then illustrates that the feed-forward neural network seems to provide accurate approximations for multivariate functions, demonstrating the usefulness of the method.

Theorem 3.3 (Universal approximation [14]). *Let σ be bounded and non-constant and ρ a finite risk measure. The following statements are true.*

- $\forall \rho$ on $(\mathbb{R}^e, \mathcal{B}(\mathbb{R}^e))$ and $p \in [1, +\infty[$, the set $\mathcal{NN}_{\infty,e,f}^\sigma$ is dense in $L^p(\mathbb{R}^e, \rho)$.
- If in addition $\sigma \in C(\mathbb{R})$, then the set $\mathcal{NN}_{\infty,e,f}^\sigma$ is dense in $C(\mathbb{R}^e)$ for the topology of uniform convergence on compact sets (or compact convergence).

This last statement explains that having an activation function that is continuously differentiable on \mathbb{R} while having the set of all neural networks to be dense on the Lebesgue space (\mathbb{R}^e, ρ) ensures that the solution given by the neural network algorithm is close-to-optimal.

Properties 3.4 *Having $\{\mathcal{NN}_{M,e,f}^\sigma\}_{M \in \mathbb{N}}$ a sequence of subsets from $\{\mathcal{NN}_{\infty,e,f}^\sigma\}$, we can give the following properties $\forall M \in \mathbb{N}$.*

- $\{\mathcal{NN}_{M,e,f}^\sigma\} \subset \{\mathcal{NN}_{M+1,e,f}^\sigma\}$ and $\bigcup_{M \in \mathbb{N}} \mathcal{NN}_{M,e,f}^\sigma = \mathcal{NN}_{\infty,e,f}^\sigma$.
- $\mathcal{NN}_{M,e,f}^\sigma = \{F^\theta : \theta \in \Theta_{M,e,f}\}$, $\Theta_{M,e,f} \subset \mathbb{R}^g$ for $g \in \mathbb{N}$ (depending on M).

This sequence could be thought of as the subset of all neural networks with a fixed number of nodes and hidden layers, parametrized by a vector θ whose dimension g depends on M [19]. The following and final part of this chapter will implement all the concepts previously introduced, in the context of hedging derivatives.

3.3 Deep learning in the context of hedging

Let us remember that the aim of our deep learning algorithm is to produce the hedging strategy $\delta \in \mathcal{H}_M$ as a numerical solution for the problem of hedging *vanilla* and *rainbow* options. Theorem 3.1 provided proof of validity for the method, and Theorem 3.2 justified the usefulness and accuracy of the approach.

The feasibility of the algorithm shall then be demonstrated in the following sections by building and training the neural network in Python with the use of Tensorflow [7]. Both training and validation sets that are to be used in the development of the algorithm will be composed of underlying asset paths, generated randomly using Numpy. **The testing set will be composed of real asset paths taken from a trading software.** To build our algorithm, we will first be following the implementations in [6] and [19].

The deep learning approach to hedging any claim will follow the same structure. We begin by generating samples of asset paths endowed with specific price dynamics, which will then be fed to the algorithm. Using the *wealth* function, the network will search for the hedging strategy that minimizes the residual cash injection, materialized by the *risk measure*. By the end of that *training* period, the algorithm should have learned an approximated relationship between the spot moves and the strategy to adopt, and we can move to the *testing* phase. Here, the algorithm is only fed with underlying paths, and will try to produce the *hedging parameters* according to what was learned in the previous phase. What is produced is then compared with what the benchmark outputs.

As was introduced previously, the network shall be trained following the *Adam algorithm* [16], with $\eta = 0.0005$ as an initial learning rate that is then individually adjusted. As was explained earlier, all the parameters of the network shall be randomly initialized from a range depending on the size of the preceding layer, increasing the pace at which the algorithm is able to determine the minimum of the loss function [12]. For efficiency purposes, we use a *batch size* of 500, which is the number of samples that are processed before the structure of the algorithm is updated. Finally, the *ReLU* activation function [7] is to be applied to our hidden layers, unless stated otherwise. The complete code can be found and used on a *GitHub* page in the following repository: [thibaultcoo/rainbow-deep-hedging](https://github.com/thibaultcoo/rainbow-deep-hedging).

Chapter 4

Deep Hedging Vanillas

The aim of this section is to demonstrate an application of deep neural networks to the most popular class of derivatives, European Vanilla options. After having introduced groundwork concepts of derivatives theory about pricing methods under different sets of assumptions, we will review how the problem has been approached in the literature, and then produce and interpret the results of our algorithm in the case of a European Put option, to finally assess the powerfulness and accuracy of the general method by comparing our results with the benchmark proposed at the beginning of the section, under two different stochastic price paths.

4.1 Fundamental derivatives pricing theory

4.1.1 Black-Scholes model

The *Black-Scholes* equation for pricing European options [3] is widely seen as one of the most popular pieces of research that was produced in recent financial mathematics, for its efficiency and usefulness under its restrictive set of assumptions. Indeed, given continuous rebalancing and assuming the complete set of assumptions is verified, the payoffs of an option can be perfectly replicated, hence allowing for a complete hedge over any risk associated with the contingent claim.

Definition 4.1 *Given all relevant assumptions are verified, the Black-Scholes price of a European Put option is given by the following equation.*

$$V(t, S_t) = e^{-r(T-t)}\mathcal{N}(-d_2)K - e^{-q(T-t)}\mathcal{N}(-d_1)S_t \quad (4.1)$$

Let \mathcal{N} be the cumulative distribution function of a standard normal random variable, r and q the constant parameters for the interest and dividend rate, σ the underlying's volatility, T the option's maturity, K the strike price and S_t the underlying's price at time t . The input parameters for \mathcal{N} are described below.

$$d_1 = \frac{1}{\sigma\sqrt{T-t}} \left(\ln \left(\frac{S_t}{K} \right) + \left(r + \frac{\sigma^2}{2} \right) (T-t) \right)$$

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

This analytical solution is obtained by solving the *Black-Scholes* parabolic partial differential equation [3], assuming S_t follows a *geometric Brownian motion*.

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0 \quad (4.2)$$

From either (4.1) or (4.2) emerges a key hedging parameter, considering we want to hedge the *put option* against the moves of the underlying's price. Such coefficient is denoted δ , and represents the sensitivity of the option's price to any move of the underlying. It is derived from *Black-Scholes* that to perfectly *delta-hedge* the position, one must trade δ unit of the underlying.

$$\delta = \frac{\partial V}{\partial S} = \mathcal{N}(d_1) - 1 \quad (4.3)$$

The benchmark will hedge the *put option* periodically with that parameter. Using *Black-Scholes* however comes at a cost, one of which is the lack of accuracy that the method tends to display, and another being the inability of the model to efficiently evaluate complex options, which we will focus on in *Chapter 5*.

4.1.2 Monte-Carlo method

Using *Monte-Carlo* simulations [4] is a wide-spread solution to deal with those shortcomings. The idea behind *Monte-Carlo* is that a large number of risk-neutral price paths for the underlying asset are generated, following a specific dynamic. For each of those paths, the option's payoff at maturity is computed, and the final value of the option is calculated as the discounted average of those payoffs, which approximates the true price of the option.

The accuracy of the method relies on the *Law of Large Numbers*, which states that the sample average converges almost surely to the expected value. This method allows to fit in any stochastic process for our underlying with potentially interesting features that are more in-line with real-world dynamics, such as the presence of jumps in the underlying's prices. Another benefit is its ability to deal with complexity in options. We will later study the use of *Monte-Carlo* for options relying on several underlying assets, illustrating the accessibility and popularity of the method for any type of exotics, even though it is computationally heavy and takes an incredible amount of time to run, the problem being the lack of more efficient alternatives. So, as an input for our deep learning algorithm in this section, we will run *Monte-Carlo* simulations on an underlying asset for two specific price dynamics separately, *Black-Scholes* and *Heston*. We will then compare the hedging strategies that come along with those dynamics, and compare them with what the benchmark produced for the same underlying.

Under the *Black-Scholes* framework, the dynamics of the underlying asset is governed by the *geometric Brownian motion* defined below [2], with μ_t and σ_t being \mathcal{F}_t -adapted processes, and B_t being a Brownian motion.

$$dS_t = \mu S_t dt + \sigma S_t dB_t$$

The *discretized Black-Scholes* dynamics will be used as a first set of inputs for our algorithm, assuming that returns are log-normally distributed with unconstrained borrowing and lending at rate r . We finally assume one can buy and sell any amount of the underlying asset in a market without any trading costs. When the price dynamics are evaluated under the risk-neutral measure \mathbb{P} , we have that $\mu = 0$ [2]. We shall use the log transform to generate our paths, with $s_t = \ln(S_t)$.

$$ds_t = -\frac{\sigma^2}{2}dt + \sigma dB_t$$

The time discretization of such path then takes the following form under \mathbb{P} , with the time interval $[0, T]$ defined in *Chapter 2*, and using N time steps with $i = 0..n$.

$$s_t = s_{t-1} - \frac{\sigma^2}{2} \frac{T}{N} + \sigma \sqrt{\frac{T}{N}} Z_i \quad (4.4)$$

In the above, Z_i are independently distributed standard normal random variables.

4.1.3 Heston model

To allow for a more compelling analysis of the efficiency of our deep learning architecture, let us introduce another model for which the volatility of the underlying is considered as a stochastic process, in order to get us closer to real market dynamics. We choose to work with the *Heston* model [13], under which the underlying's dynamics S_t are defined as below.

$$dS_t = rS_t dt + \sqrt{v_t} S_t dB_t^s, \quad S_0 \geq 0$$

Heston is a *stochastic volatility* model, so we must also define the *Cox-Ingersoll-Ross* dynamics followed by the underlying's volatility v_t , with ξ its own volatility, θ its mean and κ its mean-reversion rate.

$$dv_t = \kappa(\theta - v_t)dt + \xi\sqrt{v_t}dB_t^v, \quad v_0 \geq 0$$

Let us denote B_t^s and B_t^v two Brownian motions, whose correlation will be defined as ρ , where r denotes the risk-free rate, κ is the mean-reversion pace of the process, ξ is the volatility of the variance and θ is the long-term variance.

As for *Black-Scholes*, let us use the discretized log-transformed paths for *Heston*, so that it can be applied to the context of periodical hedging. We are still using the time interval $[0, T]$, with N time steps and $s_t = \ln(S_t)$ for $i = 0..n$.

$$s_t = s_{t-1} + r - \frac{v_{t-1}}{2} \frac{T}{N} + \sqrt{v_{t-1}} \sqrt{\frac{T}{N}} Z_i^S \quad (4.5)$$

We perform the same transform with the dynamics of the volatility. We previously used $Z_i^S \sim \mathcal{N}(0, 1)$, and to account for the relationship between the spot and volatility's dynamics, we introduce the following formula to characterize the jiggling of the volatility: $Z_i^V = \rho Z_i^S + \sqrt{1 - \rho^2} Z_i$, where Z_i^V and $Z_i \sim \mathcal{N}(0, 1)$. Let the following be the dynamics of the volatility, with $\varrho_t = \ln(v_t)$ for $i = 0..n$.

$$\varrho_t = \varrho_{t-1} + \frac{1}{v_{t-1}} \left(\kappa(\theta - v_{t-1}) - \frac{\xi^2}{2} \right) \frac{T}{N} + \left(\frac{\xi}{\sqrt{v_{t-1}}} \right) \sqrt{\frac{T}{N}} Z_i^V \quad (4.6)$$

Those paths will be generated in Python using Numpy, for both our *training* and *validation* sets. The *testing* phase will be realized using real data. Those random paths will be used to evaluate the robustness of our artificial neural network,

which shall then be compared to the hedging parameters generated using *Black-Scholes's* partial differential equation.

To resume, we will on one hand generate stochastic price paths for the underlying, feed them to the neural network algorithm so that it can decipher their key patterns, and produce hedging parameters accordingly. On the other hand, we will compute the *Black-Scholes* hedging parameters, which will allow us to compare and thus evaluate the performance of the deep learning architecture. Such reasoning shall also be observed for when we will study *multi-asset options*, the sole difference being the growing complexity in the methods used.

4.2 An extensive literature review

Several interesting papers have emerged since [15] demonstrated an ability of artificial neural networks to price futures options with great accuracy. The works of [9] extended such studies to *delta* and *vega* hedging problems for vanilla options, whose novelty was to introduce a training set of *price changes* instead of *simple prices*, which resulted in more accurate hedging ratios, as the sensitivities produced by the algorithm were better than when using the standard model.

The most important piece of research in the field of *deep hedging vanillas* is widely seen as being [6], where the authors laid down extensive mathematical foundations to the use of deep learning towards vanilla options hedging. By using *convex risk measures* as loss functions for the algorithm in a context with trading costs, the paper demonstrates that given the trajectories of all hedging instruments, with payoff samples and associated weights, deep learning techniques can be used to compute close-to-optimal hedging parameters, for any transaction cost structure and risk measure. The algorithm was tested under regular *Brownian* dynamics, as well as with *Heston* dynamics, the latter allowing for trading in both the stock and a *variance swap*. Beyond using the theoretical notions defined in the paper, we have also been influenced by the general architecture of their algorithm for our own research, although under *Heston* we chose not to introduce the *variance swap* for simplicity.

The need to verify the robustness of the method was then further explored by [19], whose intuition was to test the algorithm under different stochastic dynamics, with some introducing *jump diffusion* in the price of the assets, with *Bates*

and *Variance-Gamma* dynamics, although the resulting parameters under those dynamics were far from being satisfying. An interesting loss function defined as a mix of two expected shortfalls was however introduced, with promising results.

The paper of [8] finally demonstrated that fully *delta-hedging* is sub-optimal, and that depending on how the current holding of underlying assets compares to the quantity recommended by said *delta-hedging*, one should either under-hedge or over-hedge. By providing a loss function that varies with the first two moments of the distribution of the hedging cost, the authors illustrated an ability of *reinforcement learning* to outperform regular *delta-hedging* with trading costs, and laid down interesting ideas for further developments, such as the introduction of other moments of the hedging cost distribution, to better capture its dynamics.

4.3 Results and discussion

4.3.1 Brief overview of the research design

This section will be devoted to study the hedging strategy produced by the neural network for an At-The-Money European Put option with initial spot price $S_0 = 100$, using *discretized Black-Scholes* as a benchmark. A *simple neural network* with $L = 3$ and $L_i = \{1, 100, 100, 1\}$ shall be used in our research. Two *loss functions* are to be successively used in the following sub-sections: the *mixed expected shortfall* as defined by [19], as well as the *mean squared error*. Both were introduced in *Chapter 2*. The input data for the first two sets shall be taken from the *discretized* price dynamics of the underlying defined in (4.4), **and the testing set shall comprise real data extracted from a trading software.**

4.3.2 Outcome and benchmark comparison

need to do that
need to do that

4.4 Introducing Heston stochastic volatility

4.4.1 Context and relevance of the research

Assuming constant volatility is one of the most substantial drawbacks encountered when working under *Black-Scholes*, as it reduces considerably the approximation for the hedging parameters of an option. Therefore, we would prefer to work under *Heston* dynamics, which considers the volatility of the asset as a stochastic process. We will illustrate how successful is the artificial neural network at retrieving the fundamental patterns in such new data. The architecture of the algorithm studied in the preceding section is to be used again, as we shall only change the input data, being derived from the log-transformed discretized price paths for the asset s and its volatility ϱ , defined in (4.5) and (4.6).

4.4.2 Results and conclusions

need to do that
need to do that

Chapter 5

Deep Hedging Rainbows

In the following paragraphs, we will start focusing on the application of artificial neural networks in the context of hedging derivatives relying on more than one underlying asset. The structure will be the same as the preceding section. We shall first theoretically introduce rainbow options and the benchmarks that shall be used to evaluate the deep learning approach. Then, we will introduce some key aspects that were previously tackled in the recent literature regarding exotics in general, to finally illustrate the ability or inability of such an algorithm to correctly produce an optimal hedging strategy for a rainbow option. The final paragraphs of this chapter will further explore the potential of deep learning by applying the method in the presence of large and unexpected moves in the underlying's prices.

5.1 Financial Theory

5.1.1 Defining correlation options

One of the main challenges in portfolio management is the handling of risk exposure, to an asset class or a market sector for example. Although partly done through *hedging*, another fundamental strategy that is heavily used to reduce risk exposure is known as *diversification*, which deals with how capital is allocated in the first place in such a way as to minimize the risk of the portfolio.

The key idea behind *diversification* is that we consider the *correlation* between the movements of several asset prices. It is often perceived as preferable for the assets of a portfolio to be uncorrelated, as the plunge of one's price has lower

chances of affecting the price of the other assets in our portfolio, hence offering protection. One can therefore see the potential of speculating on *correlation*, and several of the most traded derivatives on the market are purely based on such.

Rainbow options are correlation options. As was previously introduced, they are exotic derivatives relying on the performance of several underlying assets, each being a *color* of the rainbow. Several different types of payoff structures for the rainbow option can exist on the market, allowing for any investor with a specific view on the correlation of some assets to find an interest in trading the option. For example, a basket of two underlying stocks allowing the buyer to purchase the worst performing one can be perceived as an ideal protection, as long as the correlation between both assets is high, and is called a *worst-of call option*. The latter is the most popular of the class, and shall be studied for the rest of the section. The payoff Π of a *worst-of call option* on n underlying assets is given by the following equation, with S_i and K_i being the spot and strike prices of the underlying asset i at maturity T .

$$\Pi = \max \left(0, \min_{1 \leq i \leq n} (S_i(T)) - K \right) \quad (5.1)$$

As was already introduced in *Chapter 4*, multi-asset options are too complex for a closed-form *present value* to be easily computed, and analytical approximations using *Monte-Carlo* are accessible solutions for pricing and hedging purposes. The following sub-sections will lay out the research design for the use of artificial neural networks as a potentially efficient alternative method for such purposes.

5.1.2 Multi-asset Brownian motion extension

In the setting where the price dynamics of two assets are to be generated, one has to consider the correlation in their movements. Whereas we could be tempted to proceed as in *Chapter 4* while modelling *Heston* co-movements of the spot and the volatility, here we must remark that the influence is bilateral. To take such into account, we naturally choose to work with an extension of the *geometric Brownian motion* defined earlier for single asset dynamics. A *multidimensional geometric Brownian motion* can be specified through the following system of stochastic differential equations, assuming S^i with $i = \{1, 2\}$ our two assets.

$$dS_t^i = \mu_i S_t^i dt + \sigma_i S_t^i dB_t^i \quad (5.2)$$

Considering the *correlation* between those two stochastic processes, we define by \mathcal{M} the *covariance matrix* associated with the two Brownian motions.

$$\mathcal{M}_{ij} = \rho_{ij}\sigma_i\sigma_j$$

By recalling that a *Brownian motion* B with zero mean and said *covariance* can be written as AB , with A the *Cholesky* factor of \mathcal{M} , such that $AA^T = \mathcal{M}$, we can rewrite the system defined in (5.2) as the following.

$$dS_t^i = \mu_i S_t^i dt + \sum_{j=1}^2 A_{ij} S_t^i dB_t^j$$

Finally, and to allow for practical implementation, we decide to *discretize* such system. Let us define the following algorithm for simulating multidimensional *geometric Brownian motions* with the price log-transformed, such that $s_t = \ln(S_t)$.

$$s_t^i = s_{t-1}^i - \frac{\sigma^2}{2} \frac{T}{N} + \sqrt{\frac{T}{N}} \left(\sum_{j=1}^2 A_{ij} Z_k^j \right) \quad (5.3)$$

Those are the *discretized* price dynamics for the two underlying assets on which our exotic option is to be studied. We notice that in such context, the correlation coefficient ρ is constant, which remains unrealistic but makes computations easier. We recall that the price dynamics are evaluated under \mathbb{P} , so that $\mu_i = 0$ [2].

5.1.3 Price and sensitivities of rainbows under Stulz

An analytical solution for the price of a *worst-of-two call option* was introduced by [22], along with an intuitive formula to compute its sensitivity to first-order price moves of its two underlying assets. We shall successively demonstrate those results. The latter will allow us to compute the two *delta-hedging* parameters according to the two spot prices at any moment. In its *training phase*, the neural network will be fed with a three-dimensional matrix filled with input data, which can be seen as being two-dimensional matrices for each simulation composed of the two stock price paths. Let us now lay out the theoretical foundations that led to the pricing and hedging formulas for said European *rainbow* options.

The assumptions required are the same as those defined for *Black-Scholes* in the previous chapter, with the two assets following the stochastic differential system

defined below. Let us notice that those dynamics are different than the *Multi-asset geometric Brownian motion*. We will study this difference later.

$$\begin{cases} dS_t^i = \mu_i S_t^i dt + \sigma_i S_t^i dB_t^i & \text{with } i \in \{1, 2\} \\ dB_t^1 dB_t^2 = d\rho \end{cases}$$

We assume that ρ , μ_i and σ_i for $i \in \{1, 2\}$ are constants. Under the *no-arbitrage* condition, finding the value of the option comes down to computing the value of a portfolio P which replicates its payoffs. So, following *Ito's Lemma*, we derive the dynamics of P as a function of the spot prices, as well as the time to maturity.

$$\begin{aligned} dP(S_1, S_2, \tau) &= \frac{\partial P}{\partial S_1} dS_1 + \frac{\partial P}{\partial S_2} dS_2 - \frac{\partial P}{\partial \tau} dt \\ &+ \frac{1}{2} \left(\frac{\partial^2 P}{\partial S_1^2} S_1^2 \sigma_1^2 + \frac{\partial^2 P}{\partial S_2^2} S_2^2 \sigma_2^2 + 2 \frac{\partial^2 P}{\partial S_1 \partial S_2} S_1 S_2 \rho \sigma_1 \sigma_2 \right) dt \end{aligned}$$

Given that the above defined partial differential equation is verified, P is self-financing. By noting that when either one of the assets has price 0 the option is worthless, and using the constraint defined in (5.1), we can find the analytical solution to the price R of a *worst-of two call* option [22], with strike price K .

$$R = S_1 \mathcal{N}_2(\eta_1, \beta_1, \rho_1) + S_2 \mathcal{N}_2(\eta_2, \beta_2, \rho_2) - K e^{-r\tau} \mathcal{N}_2(\gamma_1, \gamma_2, \rho)$$

There, we have that $\mathcal{N}_2(\eta, \beta, \theta)$ is the bivariate cumulative standard normal distribution, η and β being the two upper limits of integration, and θ being the correlation coefficient. We also detail the simplifications that were made above.

$$\eta_i = \frac{1}{\sigma_i \sqrt{\tau}} \left(\ln \left(\frac{S_i}{K} \right) + \left(r + \frac{\sigma_i^2}{2} \right) \tau \right) \quad \gamma_i = \eta_i - \sigma_i \sqrt{\tau}$$

$$\sigma^2 = \sigma_1^2 + \sigma_2^2 - 2\rho\sigma_1\sigma_2 \quad \rho_i = \frac{\rho\sigma_j - \sigma_i}{\sigma}$$

$$\beta_i = \frac{1}{\sigma \sqrt{\tau}} \left(\ln \left(\frac{S_j}{S_i} \right) - \frac{\sigma^2 \sqrt{\tau}}{2} \right)$$

Finally, one can compute the partial derivative of the price of the *rainbow* option with respect to the two assets, so as to define the *delta-hedging* parameters [22].

$$\begin{aligned}\delta_i = \frac{\partial R}{\partial S_i} = & \mathcal{N}_2(\eta_i, \beta_i, \rho_i) + \mathcal{N}_1\left(\frac{\eta_i - \rho_i \beta_i}{\sqrt{1 - \rho_i^2}}\right) e^{-\frac{1}{2}\beta_1^2} \frac{1}{\sqrt{k}} \\ & - \left(\frac{S_j}{S_i}\right) \mathcal{N}_1\left(\frac{\eta_j - \rho_j \beta_j}{\sqrt{1 - \rho_j^2}}\right) e^{-\frac{1}{2}\beta_2^2} \frac{1}{\sqrt{k}}\end{aligned}$$

With $j \in \{1, 2\}$, $j \neq i$ and $k = \min\{\ln(S_j)\}$. The design of our further research will be the following. The input data that will be fed into the algorithm will consist of the paths taken by the two assets. The algorithm will try to learn key features in the data, so that when introduced to new paths, the network is able to produce coherent and optimal hedging parameters. We will then compare those results with our benchmark, to assess if the algorithm can be used as a substitute *rainbow* hedging method.

The computation of the cumulative bivariate normal probability $\mathcal{N}_2(\eta, \beta, \theta)$ is to be approximated. For the rest of this paper, we will consider the algorithm proposed by [10], whose surprisingly accurate method grants us four-decimal correct approximations. The detailed application can be found in the code.

5.1.4 Multi-asset dynamics with jump-diffusion under Kou

The final and most exciting part of our research will be to evaluate the performance of our deep learning algorithm when fitted with stochastic paths for the two underlying assets endowed with random and unexpected large price moves. These sudden trends are characterized by *jumps*, and following the work of [18] on multivariate jump-diffusion models for asset pricing, we can derive the following *stochastic differential equation* for two asset prices, and remark the similarity with the system defined in (5.2) for $i, j \in \{1, 2\}$, $i \neq j$.

$$dS_t^i = \mu_i S_t^i dt + \sum_{j=1}^2 A_{ij} dB_t^j + d\left(\sum_{k=1}^{N(t)} (V_k^i - 1)\right) \quad (5.4)$$

We adopted the same notations as previously for the *Cholesky* matrix A . The final term between parentheses characterizes the additional jump part, described by a *Poisson* process $N(t)$, where jumps are denoted V_k . We consider two types of jumps, the first one being an individual jump that affects only one of the two

assets, and is defined with the *Poisson* rate λ_i for $i \in \{1, 2\}$. The second being a common jump that affects both our assets, with λ_c being the corresponding *Poisson rate*. The final rate λ of our *Poisson* process is therefore $\lambda = \lambda_c + \lambda_1 + \lambda_2$. This illustrates how complex it would be to consider more than two underlyings.

The jump V_k is such that its log-transform has the following distribution, with \mathcal{M} the covariance matrix, μ_c the vector for the means of the two assets, and μ_i and σ_i respectively the mean return and volatility of asset i . Let us remark that our *individual* and *common* jumps follow respectively one-dimensional and two-dimensional *asymmetric Laplace distributions* that we will denote \mathcal{AL} [18].

$$(\ln(V_k^1), \ln(V_k^2)) = \begin{cases} \mathcal{AL}_2(\mu_c, \mathcal{M}) & \rightarrow \text{probability} = \frac{\lambda_c}{\lambda} \\ (\mathcal{AL}_1(\mu_1, \sigma_1^2), 0)' & \rightarrow \text{probability} = \frac{\lambda_1}{\lambda} \\ (0, \mathcal{AL}_1(\mu_2, \sigma_2^2))' & \rightarrow \text{probability} = \frac{\lambda_2}{\lambda} \end{cases}$$

Computation is made easier using the following relationship, where μ_i is the mean vector of the assets, \mathcal{M}_i is the covariance matrix, \mathcal{Z} is an exponential law and i is the number of assets [17]. We remark that for *individual* jumps, \mathcal{AL}_1 is a scalar, and when *common* jumps are considered, \mathcal{AL}_2 is a two-entry array.

$$\mathcal{AL}_i(\mu_i, \mathcal{M}_i) = \mu_i \mathcal{Z} + \sqrt{\mathcal{Z}} \mathcal{N}_i(0, \mathcal{M}_i)$$

From then on, we can derive the discrete dynamics of the two asset paths, considering Z_k^i a cumulative standard normal distribution and N a *Poisson* process with rate λ . All three are assumed to be independent of the jumps themselves.

$$s_t^i = s_{t-1}^i - \frac{\sigma^2}{2} \frac{T}{N} + \sqrt{\frac{T}{N}} \left(\sum_{j=1}^2 A_{ij} Z_k^j \right) + N \left(\lambda \frac{T}{N} \right) \Lambda_i \quad (5.5)$$

We chose to denote by Λ_i the distribution for the log-transform of the jump, which could be either individual or common to both assets. Let us remark that if the jump is individual to i , then $\Lambda_j = 0$. Λ is a two entry vector that impacts both assets at every period, based on the probability distribution defined above.

The *Python* implementation will then be straightforward. We will build a module that is capable of producing the joint dynamics of our two assets over N time steps, and we will then loop for a given number of simulations to produce the *training set* that is to be given as inputs for the algorithm.

5.2 An extensive literature review

The few and recent papers that tackled the use of artificial neural networks in pricing and hedging problems for exotics options have illustrated the potential of the method in such a context. In his study of *Bermudian* options, [20] demonstrated an interesting technique to semi-statically hedge high-dimensional and path-dependant claims, by defining an upper and lower bound for the price of the option. Beyond illustrating the simplicity and efficiency of the model for both pricing and hedging purposes, the neural networks used are flexible, as they have been tested for several types of options with linear and nonlinear payoff structures, whose outcome was demonstrated to be at least as good as the benchmark, using *MSE* as a loss function.

Some other developments regarding *rainbow options* were seen with [5], who demonstrated that the algorithm could not produce satisfying *present values* for such options, although it was able to price an option as a *Monte-Carlo* method would. The reason is that the network was not able to correctly decipher the pricing function for the option, contrarily to what was demonstrated by [14]. The reason advanced by [5] was the lack of a qualitative data set to be fed to the algorithm, an issue that was further studied in [11], where the author successfully managed to value basket options on six underlyings, with prodigious pace compared to *Monte-Carlo* simulations.

5.3 Results and discussion

5.3.1 Brief overview of the research design

In this section, we will tackle the problem of producing coherent hedging parameters for a *worst-of call option* on two underlying assets with same initial spot price $S_0^i = 100$, and whose dynamics are defined in (5.2). As a benchmark, we shall use the analytical formula defined above for *delta* computation of said rainbow option. The same architecture as in the preceding chapter shall be used. The loss function chosen is the *mean squared error*. The input data that we load into the deep learning algorithm will be comprised of stochastic paths for the two assets as defined in (5.2). **The final data used to verify the performance of the algorithm will be taken from a trading software. Only paths will be generated this time**, and the algorithm will have to produce coherent hedging parameters by itself, based on the patterns it will have learned on previous *training* and

validation sets. For the time being, we choose not to change the current number of *epochs*, *learning rate* and *batch size*, respectively set at 10, 0.005 and 500.

5.3.2 Outcome and benchmark comparison

need to do that
need to do that

5.4 Multivariate case jump-diffusion extension

5.4.1 Context and relevance of the research

While it makes sense for our deep learning algorithm to be tested for hedging purposes under regular market dynamics, it comes as even more interesting to verify its robustness under large and unexpected moves in the underlying asset prices. Indeed, it is during those rare and unpredictable events that the hedging parameters produced are more likely to be *off the chart*, forcing traders to manually re-calibrate their strategies. This sub-section will be devoted to running our algorithm in such a context, by providing the artificial neural networks with asset paths generated using the method defined above [18], with the dynamics introduced in (5.5). The benchmark remains the *discrete delta-hedge* [22].

5.4.2 Results and conclusions

need to do that
need to do that

Chapter 6

Conclusion and discussion

6.1 Research outcome

what we wanted to demonstrate
how we did it and what were the results
for both vanilla and rainbow options

6.2 Limits of the deep learning method

This study was concentrated on studying vanillas and fairly restricted exotic options, as we believe that despite the remarkable computational power of artificial networks, they would most likely fail to capture more complex relationships, considering for example a *worst-of ten call option*, which would require the manipulation and approximation of extremely intertwined dynamics. Dealing with such issues may be realized through a larger *training* set and an increased variety in the generated price dynamics, considering for example a stochastic correlation matrix between the underlyings. But then again, managing to hedge single exotic derivatives fails to help the trader who wishes to hedge an entire portfolio.

The final and most inherent shortcoming to working with artificial neural networks is the *black-box* problem. As introduced earlier, those algorithm approximate non-linear relationships in the input data during the *training* phase [14], and then use such functions on new data during the *testing* phase. Those functions remain invisible to the user, so one might find it difficult to entirely trust the outcome of the network, especially in the context of hedging derivatives, where

the stakes are very high. The deep learning model can then at best be used jointly with another more classic and reliable method, which will at least reduce the degree to which traders have to rely on their intuition to re-calibrate their strategies. The algorithm could also offer clues for innovative and potentially efficient new strategies, reinforcing its position as a research tool for traders.

6.3 Further research

Reflecting on this research, several improvements could be foreseen in the context of *hedging rainbow* options, particularly through two major refinements: the necessity to consider *stochastic correlations* between the colors of the rainbow, and finally the possibility to implement *trading signals* in the form of *forecasts* for the volatility and the correlations of the underlyings.

6.3.1 Considering stochastic correlation

The high degree of complexity that implementing those methods in such context would require refrained us from wanting to do so, but the need to further train the algorithm in a context that aspires to imitate real market dynamics will inevitably lead us to such future developments, given that some significant study regarding the specific architecture that the algorithm should possess be made. However, some changes that would be easier to implement could be for example to discretely change ρ at some point in the paths of the two assets.

We could also consider a correlation *trading signal* that would encompass the first two moments of the distribution for the correlation between our two assets. Using such might let the algorithm observe the appearance of specific patterns before episodes of correlation moves and understand their consequences on the hedging parameters, in order to adjust the strategy consequently.

6.3.2 Volatility forecasts through quantile regression

Following the work of [24], we propose to implement quantile regression to determine volatility forecasts for the returns of our underlyings, and use those as trading signals, introduced as additional inputs to our deep learning algorithm. This method relies on the analysis of stock return autocorrelation on several

sub-parts of the overall distribution, based on which we can infer a general autocorrelation pattern which we will use to determine future returns volatility levels.

This method was proven robust, for it does not assume a fixed conditional distribution of stock returns, an assumption that truly harmed other simpler models. Indeed, it was shown by [24] that autocorrelation on both extremes of the distribution is different. When stock prices are increasing, their future returns appeared as negatively correlated with their past returns, which is also true contrariwise. Using such might allow the algorithm to refine its hedging strategy, by considering deeper dynamics than those observable with current models.

Bibliography

- [1] An Yunbi, Suo Wulin (2003), *The performance of option pricing models on hedging exotic options*, Working Paper, Queen's University.
- [2] Back Kerry (2005), *A course in Derivatives Securities: Introduction to Theory and Computation*, Springer Finance Textbook, 372 pages.
- [3] Black Fisher, Scholes Myron (1973), *The pricing of options and corporate liabilities*, Working Paper, The University of Chicago Press, The Journal of Political Economy, Vol. 81, No. 3 (May - Jun., 1973), pp. 637-654.
- [4] Boyle Phelim (1977), *Options: A Monte Carlo approach*, Working Paper, Journal of Financial Economics, Volume 4, Issue 3, May 1977, pp. 323-338.
- [5] Brostrom Axel, Kristiansson Richard (2018), *Exotic derivatives and deep learning*, Master's Thesis, KTH Royal Institute of Technology.
- [6] Buehler Hans, Gonon Lukas, Teichmann Josef, Wood Ben (2018), *Deep hedging*, Working Paper, JP Morgan, ETH Zürich.
- [7] Chollet François (2021), *Deep learning with Python*, Manning, 384 pages.
- [8] Cao Jay, Chen Jacky, Hull John, Poulos Zissis (2020), *Deep hedging of derivatives using reinforcement learning*, Working Paper, Journal of Financial Data Science, 3, pp. 10-27.
- [9] Carverhill Andrew, Cheuk Terry (2003), *Alternative Neural Network Approach for Option Pricing and Hedging*, Working Paper, The University of Hong Kong, School of Business.
- [10] Drezner Zvi (1978), *Computation of the bivariate normal integral*, Mathematics of Computation, 32 (January 1978), pp. 277-279.
- [11] Ferguson Ryan, Green Andrew (2018), *Deeply learning derivatives*, Working Paper, Scotiabank.

- [12] He Kaiming, Zhang Xiangyu, Ren Shaoqing, Sun Jian (2015), *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, Working Paper, Microsoft Research.
- [13] Heston Steven (1993), *A closed-form solution for options with stochastic volatility with applications to bond and currency options*, The Review of Financial Studies. 6 (2), pp. 327-343.
- [14] Hornik Kurt (1991), *Approximation capabilities of multilayer feedforward networks*, Working Paper, Neural Networks 4, no. 2, pp. 251-257.
- [15] Hutchinson James, Lo Andrew, Poggio Tomaso (1994), *A nonparametric approach to pricing and hedging derivative securities via learning networks*, Working Paper, The Journal of Finance Vol. 49 No. 3, Massachusetts Institute of Technology.
- [16] Kingma Diederik, Lei Ba Jimmy (2015), *Adam: a method for stochastic optimization*, Conference paper, OpenAI, University of Amsterdam, University of Toronto.
- [17] Kotz Samuel, Kozubowski Tomasz, Podgorski Krzysztof (2003), *The Laplace distribution and generalizations*, Textbook, Boston, MA, pp. 239-272.
- [18] Kou Steven (2008), *Jump-diffusion models for asset pricing in financial engineering*, Survey article, J.R. Birge and V. Linetsky (Eds.), Handbooks in OR MS, Vol. 15, pp. 73-116.
- [19] Kozira Michal (2018), *Deep learning approach to hedging*, Master's Thesis, Oxford University.
- [20] Lokeshwar Vikranth, Bhardwaj Vikram, Jain Shashi (2019), *Neural network for pricing and universal static hedging of contingent claims*, Working Paper, Cornell University.
- [21] Ruder Sebastian (2019), *An overview of gradient descent optimization algorithms*, Working Paper, Insight Center for Data Analytics, NUI Galway.
- [22] René Stulz (1982), *Options on the minimum or the maximum of two risky assets: Analysis and Applications*, Working Paper, Journal of Financial Economics, University of Rochester, pp. 161-185.
- [23] Ye Ziqun (2013), *The Black-Scholes and Heston models for option pricing*, Master's Thesis, University of Waterloo.
- [24] Zhao Yixiu (2020), *Stock returns, quantile autocorrelation and volatility forecasting*, Working Paper, International Review of Financial Analysis.