Pricing and Hedging With Neural Networks

Rugile Ulianskaite

Supervisor: Dr. Huy Chau

The University of Manchester

Abstract

This dissertation explores how neural networks can price and hedge financial derivatives in incomplete markets with extreme risks. Traditional models like Black-Scholes rely on idealized assumptions: complete markets, frictionless trading, and continuous price movements that rarely hold in the real world. While early applications of neural networks focused mainly on predicting option prices at a single point in time, this project investigates whether neural networks can also effectively learn dynamic hedging strategies in challenging market conditions.

We combine a strong mathematical foundation, covering Lebesgue spaces, functional analysis, risk measures, and utility theory, with practical model development. Using the deep hedging framework, we train neural networks to approximate hedging strategies while minimizing four different loss functions, each with different properties and sensitivity to risk. The models are tested in two contrasting environments: an idealized Black-Scholes market and a heavy-tailed market generated by a jump-diffusion process. We evaluate each model's pricing accuracy, hedging error, and training stability.

Our findings reveal that while all models perform well in stable markets, risk-aware loss functions, specifically Conditional Value-at-Risk and entropic risk, provide a clear advantage when managing extreme risks. In heavy-tailed settings, models trained with the Value-at-Risk loss function struggle, producing large hedging errors, whereas those using Conditional Value-at-Risk and entropic risk achieve lower tail errors and greater robustness. Surprisingly, the simple mean qquared error loss remains competitive in both environments, especially in pricing accuracy and training efficiency.

The results highlight both the strengths and challenges of deep hedging. The approach avoids the need to specify an equivalent martingale measure and adapts to different market conditions. Nevertheless, it is not free from modelling choices as network architecture and loss function selection play a critical role in shaping the learned hedging strategies. This project demonstrates the potential of combining financial risk theory with modern machine learning and the importance of careful model design when navigating the complexities of real-world markets.

Contents

Chapter 1

Introduction

Pricing and hedging financial derivatives - a problem at the heart of financial mathematics - is essential for managing risk and making investment decisions. For decades, models like Black-Scholes have provided analytical solutions under idealized assumptions: continuous trading, no market frictions, and complete markets. However, in real-world markets, traders encounter discrete-time trading, liquidity constraints and market frictions. In the face of such limitations, model risk becomes a frequent pitfall when financial models rely on incorrect or overly simplistic assumptions, particularly in market conditions with sudden jumps and illiquidity.

This has caused a quickly growing interest in data-driven approaches, particularly using neural networks. Pricing and hedging with such models does not require the market to behave according to specific assumptions: instead, they learn directly from data. Early work by Hutchinson et al. (1994) showed that neural networks could price derivatives as well as traditional models could. Since then, over a hundred papers have explored the applications of neural networks to pricing and hedging, covering everything from implied volatility surface estimation to exotic option valuation. Ruf and Wang (2019) provide a comprehensive review of this literature, highlighting the main directions, such as data leakage and comparisons with classical models as a benchmark. Despite this, most of the early research focuses on making predictions of option prices at a single point in time rather than how to adjust a hedge over time in real and imperfect markets.

This changed when Buehler et al. (2019) introduced the deep hedging framework for pricing and hedging. This model-free approach is designed to handle incomplete markets and incorporate trading costs, making it one of the first successful dynamic hedging models. In this project, we explore how the choice of loss function - including mean squared error (MSE) as well as risk measures Value-at-Risk (VaR), Conditional Value-at-Risk (CVaR), and entropic risk - affects the performance of hedging strategies learned through this framework. We test these methods both in idealized settings such as Black-Scholes and in more realistic environments with sudden jumps and heavy-tailed losses, to better understand the strengths and limitations of neural network-based hedging.

The dissertation is organized as follows. We begin by building a strong foundation of the mathematical results in Lebesgue spaces and functional analysis, essential for understanding the concepts in deep learning and hedging in Chapter 2. Having built an understanding of these concepts, we then apply them in Chapter 3 to explore the mathematics behind neural networks, justify their effectiveness by proving the existence of a universal approximator, and understand the learning process of a neural network. In Chapter 4, we review the relevant concepts in financial mathematics, introducing the reader to risk-neutral pricing of assets, the basics of stochastic calculus and the Black-Scholes model. In the last part of Chapter 4, we introduce the risk measures Value-at-Risk (VaR), Conditional Value-at-Risk (CVaR), and entropic risk and focus on risk measurement, distinguishing between convex and coherent risk measures, associating risk measures with acceptance sets, and exploring utility theory. Lastly, we review the dual representation of risk measures as well as the effects of semicontinuity on them.

In Chapter 5, we bring all the content of the previous chapters together in a study of the deep hedging framework. We examine the assumptions on the market, formulations of the optimization problem, and methods of risk management used in the problem. Moreover, we restate the optimization problem in terms of neural networks and analyze convergence results to understand the theoretical effectiveness of the framework. Chapter 6 contains a numerical implementation of the deep hedging framework. We present the computational setting with four models, each trained with a different loss function. The loss functions are chosen to explore two properties: convexity (by selecting risk measures that are non-subadditive, coherent, or convex) and risk-awareness (by varying the degree to which each loss function accounts for extreme losses and tail risks). We test the models in an idealized Black-Scholes market and in a more realistic setting with sudden jumps and heavy tails to investigate the robustness of each model.

Chapter 2

Lebesgue Spaces

Before delving into the mathematics behind neural networks and financial markets, we need to understand concepts essential in these fields. In this chapter, we state the key results from Lebesgue spaces and functional analysis, which we will rely on in subsequent chapters.

We present the definitions and results for a general vector space X over $\mathbb{R}$. In later chapters, we will often use $X = \mathbb{R}^{d_0}$ for $d_0 \in \mathbb{N}$ as the vector space. Now, we recall the concepts of normed linear spaces and Lebesgue spaces. For those interested in more context on L^p spaces, see Chapters 3.1, 3.2 and Appendix B in Castillo and Rafeiro (2016).

Definition 2.0.1 (Normed Linear Spaces) *A vector space X is a normed linear space over the field of real numbers if it is endowed with a norm, which is a function $\|\cdot\| : X \to [0, \infty)$ that satisfies the following conditions:*

(i) $\|\cdot\|$ *is positive definite:* $\|x\| = 0 \iff x = 0 \; \forall x \in X$*;*

(ii) $\|\cdot\|$ *is homogeneous:* $\|\alpha x\| = |\alpha|\|x\| \; \forall \alpha \in \mathbb{R}, x \in X$*;*

(iii) $\|\cdot\|$ *satisfies the triangle inequality:* $\|x + y\| \leq \|x\| + \|y\| \; \forall x, y \in X$*.*

Definition 2.0.2 (Lebesgue space) *The Lebesgue or L^p space, $L^p(X, \mathcal{F}, \mu)$, with $p \geq 1$ is a normed linear space of $\mathcal{F}$-measurable functions that are Lebesgue p-integrable with respect to the measure μ:*

$$\|f\|_p = \left(\int_X |f|^p d\mu(x) \right)^{\frac{1}{p}} < \infty.$$

If μ is a probability measure, we say that $L^p(X, \mathcal{F}, \mu)$ is a space of functions with a finite p-th moment:

$$\mathbb{E}[|f|^p] = \int_X |f|^p d\mu(x) < \infty.$$

Example 2.0.3 *Let $X = [0, 1]$ with μ as the Lebesgue measure. Then, the function $f(x) = x^2$ has a finite second moment, that is, the function belongs to $L^2([0, 1], \mathcal{F}, \mu)$ because $\int_0^1 |f(x)|^2 dx = \int_0^1 x^4 dx = \frac{1}{5} < \infty$.*

It is important to state the following fundamental integral inequality as it is a central result for the study of Lebesgue spaces.

Lemma 2.0.4 (Hölder's Inequality) *Let $p, q > 0$ be conjugate exponents, that is,*

$$\frac{1}{p} + \frac{1}{q} = 1$$

and consider the functions $f \in L^p(X, \mathcal{F}, \mu)$ and $g \in L^q(X, \mathcal{F}, \mu)$. Then, $fg \in L^1(X, \mathcal{F}, \mu)$ and

$$\int_X |fg| d\mu \leq \|f\|_p \|g\|_q.$$

Note that taking $p = q = 2$ leads us to a result that is already familiar - the Cauchy-Schwartz inequality. For the proof of this result or more details, see Chapter 3.2 in Castillo and Rafeiro (2016).

Now, we review some results about L^p spaces from functional analysis. These results are invaluable in understanding the effectiveness of neural networks and concepts in financial mathematics. A *functional* is a specific type of function that takes another function from the space X as its input. If x is a functional, we denote it by $*$, that is, $x^* : X \to \mathbb{R}$. If such a functional is linear and bounded, it belongs to a functional space X^* - the dual space of X.

Theorem 2.0.5 (Hahn-Banach Separation Theorem) *Let Y be a linear subspace of a normed linear vector space X. If Y is not dense in X, then there exists a nonzero functional $\Lambda \in X^*$, $\Lambda \neq 0$ such that $\Lambda(y) = 0 \ \forall y \in Y$.*

This result shows that, if a subspace cannot approximate all functions in a larger space, then we can express this limitation by defining a functional. Indeed, if Y is not dense in X, there are points in X that Y cannot "reach". The functional Λ then captures there points by assigning them nonzero value in X but vanishing on Y.

Example 2.0.6 *Consider the normed linear space $X = \mathbb{R}^2$ and let Y be the subspace of all vectors parallel to $\begin{pmatrix}1\\1\end{pmatrix}$. Clearly, Y is not dense in X, so for all $\begin{pmatrix}x_1\\x_2\end{pmatrix} \in Y$ the functional $\Lambda \begin{pmatrix}x_1\\x_2\end{pmatrix} = x_1 - x_2$ vanishes. Indeed, if this vector is parallel to $\begin{pmatrix}1\\1\end{pmatrix}$, then $x_1 = x_2$ and so the functional is zero. However, for $\begin{pmatrix}x_1\\x_2\end{pmatrix} \notin Y$, $x_1 \neq x_2$ and so Λ is nonzero everywhere else on X.*

Theorem 2.0.7 (Riesz Representation Theorem for L^p Spaces) *Let $(X, \mathcal{F}, \mu)$ be a σ-finite measure space[1] and let $1 < p, q < \infty$, where p and q are conjugate exponents. Then, there is an isometric isomorphism[2] τ from $[L^p(X, \mathcal{F}, \mu)]^*$ onto $L^q(X, \mathcal{F}, \mu)$.*

Further, $\forall x^* \in [L^p(X, \mathcal{F}, \mu)]^*$, there is a unique $g \in L^q(X, \mathcal{F}, \mu)$ such that $\forall f \in L^p(X, \mathcal{F}, \mu)$, we have

$$x^*(f) = \int_X fg d\mu,$$

with $\|x^*\| = \|g\|_q$.

Theorem 2.0.7 guarantees that any functional from the dual space $L^p(X, \mathcal{F}, \mu)$ can be associated with a function in $L^q(X, \mathcal{F}, \mu)$ without losing any properties such as norm or structure. This ensures that we can reach any space $L^q(X, \mathcal{F}, \mu)$ from any space $L^p(X, \mathcal{F}, \mu)$ for $1 < p < \infty$ using isometric isomorphisms.

Observe that the conjugate exponents in Theorem 2.0.7 p and q take values in $(1, \infty)$, however, in Definition 2.0.2, we see that L^p is also defined for $p = 1$. Indeed, we can have $L^1(X, \mathcal{F}, \mu)$ with the 1-norm. It is then natural to ask whether we can use Theorem 2.0.7 for $p = 1$. Notice that the conjugate exponent of $p = 1$ is

$$\frac{1}{p} + \frac{1}{q} = 1. \quad \iff \quad \frac{1}{q} = 0 \quad \iff \quad q \to \infty$$

We again refer to Castillo and Rafeiro (2016) to understand the notion of an L^∞ space.

[1] A σ-finite measure allows us to divide the set X into countable subsets each of which have a finite measure.
[2] These two properties describe a distance preserving, bijective function.

Definition 2.0.8 (L^∞ Space) *$L^\infty(X, \mathcal{F}, \mu)$ is a normed linear space of $\mathcal{F}$-measurable, essentially bounded functions. That is, $\forall f \in L^\infty(X, \mathcal{F}, \mu)$, f has a bounded essential supremum*

$$\|f\|_\infty = ess\sup f = \inf\{M \geq 0 : |f(x)| \leq M \quad \mu - almost\ everywhere\}.$$

Hence, we only require f to be bounded on the majority of its domain, with the exception of sets with zero measure.

Now, we can state the following result.

Lemma 2.0.9 *Let $(X, \mathcal{F}, \mu)$ be a σ-finite measure space. Then, $\forall x^* \in [L^1(X, \mathcal{F}, \mu)]^*$, there is a unique function $g \in L^\infty(X, \mathcal{F}, \mu)$ such that*

$$x^*(f) = \int_X fgd\mu.$$

Furthermore, $\|x^*\| = \|g\|_\infty$ and the map $x^* \mapsto g$ is an isometric isomorphism $[L^1(X, \mathcal{F}, \mu)]^* \to L^\infty(X, \mathcal{F}, \mu)$.

The Riesz Representation Theorem 2.0.7 and Lemma 2.0.9 appear to be very similar: Theorem 2.0.7 allows us to find functions connecting Lebesgue spaces for any $1 < p, q < \infty$, while Lemma 2.0.9 only permits connecting results between L^∞ and the dual space of L^1.

We do not prove the Hahn-Banach Separation Theorem 2.0.5, the Riesz Representation Theorem 2.0.7, or Lemma 2.0.9. A more detailed discussion can be found in Chapter 4 of Friedman (1982).

Lastly, we state a different version of the Riesz Representation Theorem concerning continuous duals. Note that $C(X)$ denotes the space of continuous functions $f : X \to \mathbb{R}$.

Theorem 2.0.10 (Riesz Representation Theorem for Continuous Duals) *Let $x^* : C(X) \to \mathbb{R}$ be a positive linear functional. There exists a unique monotone function $\mu : X \to \mathbb{R}$ such that*

$$x^*(f) = \int_X f(x)d\mu(x),$$

where the integral is the Riemann-Stieltjes integral.

In this, the Riemann-Stieltjes integral generalizes the Riemann integral generalizes the Riemann integral by introducing a weighting function $\mu(x)$, called the integrator, which determines the "size" of each subinterval. Instead of using uniform interval lengths, the integral accounts for variations in $\mu(x)$. For details and a proof of the Riesz Representation Theorem for Continuous Duals, see del Rio et al. (2017).

Chapter 3

Neural Networks

3.1 Introduction to Neural Networks

Mathematical modelling is fundamental to most applied mathematics problems because it allows us to represent and analyze real-life situations. Consider a scenario where data is collected on the size of a house and its distance to the city center. We aim to predict the market price of the house based on these two features. This task can be framed as a problem of function approximation, where we aim to find a function that maps each data point x, here consisting of size and distance, to a corresponding output y, the house price. We can generalize our findings by using the approximated function to make predictions on previously unseen data.

Although they may seem complicated at first, neural networks are simply powerful function approximators inspired by the structure of the human brain. Like the brain, neural networks consist of layers of interconnected artificial neurons. Just as the human brain learns from experience, neural networks approximate and adjust functions to recognize patterns and relationships between data.

3.2 How Do Neural Networks Work

After explaining the motivation, we now move on to the mechanics of neural networks. This section explains the basic elements of a neural network, including its layers, activation functions, and mathematical structure, and demonstrates how these elements work together to create powerful function approximators.

A neural network is a parametric machine learning model designed to learn the function f, which maps entries from the input space $\mathcal{X}$ to entries from the output space $\mathcal{Y}$ as accurately as possible. A neural network aims to find the optimal mapping for predicting $\mathcal{Y}$ based on $\mathcal{X}$. This is done through layered processing of the input data. Each layer contains connected nodes called neurons that transform these inputs.

We can find three types of layers in a neural network. The *input layer* receives the data. The data is then passed on to *hidden layers*, which do the heavy lifting of learning the patterns in the data. Lastly, the *output layer* produces the final predictions.

As in Higham and Higham (2019), let $L \in \mathbb{N}$ be the total number of layers in a neural network except for the input. As we move through the network, we refer to the current layer as l. For each layer $l = 0, ..., L$, let $N_l \in \mathbb{N}$ be the number of neurons in that layer. Denote by F^l the output of layer l. The layer $l = 1$ is the input layer, so we define its output as the input data, i.e. $F^1 := x \in \mathbb{R}^{N_1}$, and the output is denoted by $F^L := F(x) \in \mathbb{R}^{\mathbb{N}_L}$.

Example 3.2.1 *Let us revisit the scenario introduced in Section 3.1, in which we aim to collect data about the size of a house and its distance from the city centre in order to predict the market price of the house. Our input data is then $n \in \mathbb{N}$ pairs of observations that take real values, $(x_1, x_1'), ..., (x_n, x_n') \in \mathcal{X} := \mathbb{R}^2$. Therefore, the input layer has two neurons, that is, $N_1 = 2$.*

Since we are interested in the market price of the house, our output is a single real number. That is, we seek $y_1, ..., y_n \in \mathcal{Y} := \mathbb{R}$, and so the ouput layer has one neuron, $N_L = 1$. Taking $L = 3$, with $N_2 = 3$, we get a simple neural network, illustrated in Figure 3.1.

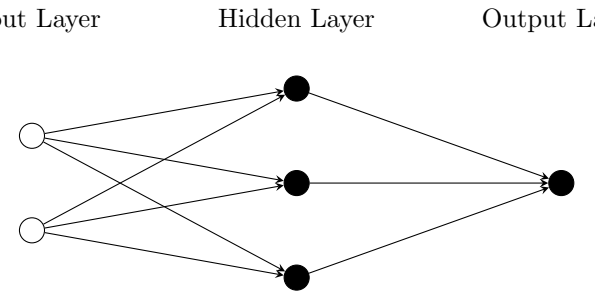


Figure 3.1: Structure of a simple neural network with one hidden layer.

After the data passes through the input layer and flows through the hidden layers, each neuron applies an *activation function* to find the output that will be passed on to the next layer. An activation function allows the neural network to model non-linear relationships and patterns, making predictions more accurate. Without introducing nonlinearity like this, the entire network would be equivalent to a single linear transformation, regardless of how deep it is. There are many choices for an activation function. For illustrative purposes, we use the historically popular sigmoid activation function

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

The sigmoid function is convenient for examples since we can express its derivative as $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. For those interested in exploring other activation functions, refer to Chapters 6.2 and 6.3 in Goodfellow et al. (2016).

Having introduced activation functions, we can now define outputs for subsequent layers. The output of the layer l, for $l = 2, ..., L$ is defined recursively as

$$F^l := \sigma\left(W^{[l]} F^{l-1} + b^{[l]}\right).$$

In this equation, $W^{[l]} \in \mathbb{R}^{N_l \times N_{l-1}}$ represents a weight matrix that transforms the output passed on to a neuron in layer l from another neuron in layer $l-1$. We then add biases $b^{[l]} \in \mathbb{R}^{N_l}$. The final output in layer L is obtained by applying this transformation through all layers. Adopting notation from Buehler et al. (2019), we define the neural network as follows.

Definition 3.2.2 (Neural Network) *For any $l = 1, ..., L$, let $W_l : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$ be an affine function, given by $F^{l-1} \mapsto F^l := W^{[l]} F^{l-1} + b^{[l]}$ for weight matrices $W^{[l]} \in \mathbb{R}^{N_l \times N_{l-1}}$ and bias vectors $b^{[l]}$. A function $F : \mathbb{R}^{N_1} \to \mathbb{R}^{N_L}$ defined as*

$$F(x) := F^L = W_L \circ F^{L-1} \circ \cdots \circ F^1 \quad \text{with} \quad F^l = \sigma \circ W_l \quad \text{for } l = 1, ..., L-1$$

is called a feed forward neural network. Here, the activation function is applied componentwise.

Example 3.2.3 *Revisiting the setup in Example 3.2.1, we note that for the two dimensional inputs, we have $F^1 \in \mathbb{R}^2$. The hidden layer has three neurons, $F^2 \in \mathbb{R}^3$, so the weights and*

biases used to transform the input data are $W^{[2]} \in \mathbb{R}^{3\times 2}$ *and* $b^{[2]} \in \mathbb{R}^3$*. The affine function for this layer is then* $F^1 \mapsto F^2 = W^{[2]}F^1 + b^{[2]}$*.*

The output is a single value, so $F^3 \in \mathbb{R}$ *and the affine function is given by* $F^2 \mapsto F^3 = W^{[3]}F^2 + b^{[3]}$*, where* $W^{[3]} \in \mathbb{R}^{1\times 3}$ *and* $b^{[3]} \in \mathbb{R}$*.*

3.3 Why Do Neural Networks Work

With a general understanding of how neural networks work, we want to provide a theoretical basis for their effectiveness. The Universal Approximation Theorem guarantees that, in theory, even simple neural networks with just one hidden layer can model any continuous function with sufficient accuracy, assuming there are enough neurons. This result highlights the flexibility and potential of neural networks as a powerful and versatile machine learning tool. We state and reprove the result in Hornik (1991), while adopting the notation used in Buehler et al. (2019) to align with our context of pricing and hedging.

Theorem 3.3.1 (Universal Approximation Theorem) *Denote by* $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ *the set of neural networks with the activation function* σ *mapping* $\mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$*. Suppose* σ *is bounded and non-constant. The following statements hold:*

(i) For any finite measure μ *on* $(\mathbb{R}^{d_0}, \mathcal{B}(\mathbb{R}^{d_0}))$ *and* $1 \le p < \infty$*, the set* $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ *is dense in* $L^p(\mathbb{R}^{d_0}, \mu)$*.*

(ii) If, in addition, $\sigma \in C(\mathbb{R})$*, then* $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ *is dense in* $C(\mathbb{R}^{d_0})$ *for the topology of uniform convergence on compact sets.*

This result tells us that neural networks are dense in both $L^p(\mathbb{R}^{d_0}, \mu)$ and $C(\mathbb{R}^{d_0})$ under the given conditions. Here, *dense* means that we can find the required function by adjusting the parameters of a neural network.

In Theorem 3.3.1 (i), we consider the space $L^p(\mathbb{R}^{d_0}, \mu)$ as in Definition 2.0.2. In this context, neural networks can approximate any function in the L^p space to any accuracy in an "average" sense, where accuracy is measured by the L^p norm.

Theorem 3.3.1 (ii) suggests that, if σ itself is continuous, for every function f, continuous in $\mathbb{R}^{d_0}$, there is a neural network that approximates $f(x)$ to an arbitrary accuracy for all x in a compact subset[1]. This is a stronger result than part (i) because it guarantees uniform convergence on compact sets rather than average approximation.

This result is crucial for understanding why neural networks work in practice as it proviides a theoretical foundation for their effectiveness. To fully grasp this, we now state the proof of the Universal Approximation Theorem. For this, we need the following lemma from Hornik (1991).

Lemma 3.3.2 *If* σ *is bounded and non-constant, then* σ *is discriminatory. That is, there does not exist a finited signed measure* α *on* $\mathbb{R}^{d_0}$ *such that*

$$\int_{\mathbb{R}^{d_0}} \sigma(Wx + b)d\alpha = 0 \quad \text{for all } W \in \mathbb{R}^{d_1\times d_0}, b \in \mathbb{R}^{d_1}.$$

Proof of Theorem 3.3.1. *Suppose that* σ *is bounded and non-constant.*

(i) First, note that $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ *is a linear subspace of* $L^p(\mathbb{R}^{d_0}, \mu)$*. Indeed, for any functions* $f, g \in \mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$*, we have that* $af + bg \in \mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ *for constants* a, b*, so* $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ *itself is a linear space. Now,* σ *is bounded by assumption, i.e.* $|\sigma(x)| \le M$ *for some constant* $M > 0$ *and all* $x \in \mathbb{R}$*, hence, for any input,* σ *restricts the outputs of each neuron to* $[-M, M]$*. This means that any* $f \in \mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ *is bounded. Because* f *is bounded*

[1] In a finite-dimensional space such as $\mathbb{R}^{d_0}$, compact can be taken to mean closed and bounded.

and the measure μ is finite, we have $\int_{\mathbb{R}^{d_0}} |f|^p d\mu < \infty$ and so, for all $f \in \mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$, $f \in L^p(\mathbb{R}^{d_0}, \mu)$.

Theorem 2.0.5 tells us that, if for some μ $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ is not dense in $L^p(\mathbb{R}^{d_0}, \mu)$, there is a nonzero linear functional $\Lambda \in [L^p(\mathbb{R}^{d_0}, \mu)]^*$ that vanishes on $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$.

By Theorem 2.0.7 and Lemma 2.0.9, we have that Λ is of the form $\Lambda(f) = \int_{\mathbb{R}^{d_0}} f g d\mu$ for some $g \in L^q(\mathbb{R}^{d_0}, \mu)$ for a conjugate exponent q of p, $\forall p, q, \geq 1$.

Now, take some Borel set B and define the set function $\alpha(B) = \int_B g d\mu$, the value of which can be viewed as a weighted contribution of g over the set B with respect to measure μ. Since $\mathbf{1}_B \in L^p(\mathbb{R}^{d_0}, \mu)$, we can use Hölder's Inequality 2.0.4 to see that for all $B \in \mathcal{B}(\mathbb{R}^{d_0})$ we have

$$|\alpha(B)| = \left| \int_{\mathbb{R}^{d_0}} \mathbf{1}_B g d\mu \right| \leq \|\mathbf{1}_B\|_p \|g\|_q. \tag{3.1}$$

We find the p-norm of $\mathbf{1}_B$ with respect to μ:

$$\|\mathbf{1}_B\|_p = \left(\int_{\mathbb{R}^{d_0}} |\mathbf{1}_B|^p d\mu \right)^{\frac{1}{p}} = \left(\int_B 1 d\mu \right)^{\frac{1}{p}} = (\mu(B))^{\frac{1}{p}}$$

and since we have a finite measure space, $(\mu(B))^{\frac{1}{p}} \leq \left(\mu(\mathbb{R}^{d_0})\right)^{\frac{1}{p}}$. We substitute this in (3.1) to get the following result:

$$|\alpha(B)| \leq \left(\mu(\mathbb{R}^{d_0})\right)^{\frac{1}{p}} \|g\|_q < \infty.$$

This bound shows that α is finite for all Borel sets and, since α is linear and countably additive, it is a finite signed measure on $\mathbb{R}^{d_0}$. Thus, using α, we can rewrite the functional as

$$\Lambda(f) = \int_{\mathbb{R}^{d_0}} f g d\mu = \int_{\mathbb{R}^{d_0}} f d\alpha.$$

Recall that any function f in our set of neural networks is of the form $f(x) = \sigma(Wx + b)$. Since Λ vanishes on $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$, we note that for all $W \in \mathbb{R}^{d_1 \times d_0}$, $b \in \mathbb{R}^{d_0}$

$$\int_{\mathbb{R}^{d_0}} \sigma(Wx + b) d\alpha(x) = 0.$$

(ii) Suppose that $\sigma \in C(\mathbb{R}^{d_0})$. Similarly to the begining of the proof, we note that $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ is a linear subspace of $C(\mathbb{R}^{d_0})$. Suppose that for some compact subset $X \subseteq \mathbb{R}^{d_0}$, $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ is not dense in $C(X)$. Once again, Theorem 2.0.5 gives that there exists some nonzero linear functional Λ on $C(X)$ such that Λ vanishes on $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$.

Then, using Theorem 2.0.10 we can proceed in the same way as above to find a finite signed measure α on $\mathbb{R}^{d_0}$ such that

$$\Lambda(f) = \int_{\mathbb{R}^{d_0}} f(x) \ d\alpha(x) \text{ for all } f \in C(X).$$

Since our functional vanishes on $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$, for all $W \in \mathbb{R}^{d_1 \times d_0}, b \in \mathbb{R}^{d_0}$ we have

$$\int_{\mathbb{R}^{d_0}} \sigma(Wx + b) \ d\alpha(x) = 0.$$

In both cases, we arrive at the conclusion that

$$\int_{\mathbb{R}^{d_0}} \sigma(Wx + b) \ d\alpha(x) = 0 \text{ for all } W \in \mathbb{R}^{d_1 \times d_0}, b \in \mathbb{R}^{d_0}. \tag{3.2}$$

Since σ is bounded and non-constant, Lemma 3.3.2, implies that σ is discriminatory, which contradicts (3.2). Therefore, we conclude that $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ is dense in both $L^p(\mathbb{R}^{d_0}, \mu)$ and $C(\mathbb{R}^{d_0})$.
□

The Universal Approximation Theorem is a result that is foundational in understanding the potential of neural networks as powerful function approximators. However, it has limitations that should be considered in practice. It is important to note that this result is existential rather than constructive: while it guarantees the existence of a network that can approximate any function, it does not provide guidance on how to find this network. Moreover, the assumption that there are arbitrarily many neurons is problematic as this can be unachievable in practice because of computational cost.

3.4 The Challenge of Optimization

The versatility of neural networks comes from their ability to adapt functions through layers of interconnected neurons. This is not automatic: we still need to find the best values for the network's parameters, which brings us to the main challenge in machine learning: optimization.

Indeed, to find the weights and biases that allow a neural network to approximate functions accurately, we aim to minimize the *loss function*. The loss function measures how well the neural network's predictions align with the true labels. It quantifies the error between the target values and the network's outputs, guiding the learning process.

At each step, the optimization algorithm adjusts the weights and biases to reduce the loss value and improve the network's predictions. In this way, the model learns from its mistakes: larger error causes larger updates, pushing the network towards better approximations.

We explain optimization relying on Higham and Higham (2019). Consider the setup as in Section 3.2, and let Θ_{∞,d_0,d_1} be the parameter space containing all weights and biases for networks mapping from $\mathbb{R}^{d_0}$ to $\mathbb{R}^{d_1}$. Then, $\theta \in \Theta_{\infty,d_0,d_1}$ is the vector in $\mathbb{R}^p$ for some $p \in \mathbb{N}$, representing the weights and biases used in a network. The *loss function* is then a mapping $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$, where the codomain is all possible evaluations of the network's performance quantified by a real number. Our task then becomes a minimization problem, namely, finding the parameters in

$$\inf_{\theta \in \Theta_{\infty,d_0,d_1}} \mathcal{L}(\theta), \tag{3.3}$$

where the loss $\mathcal{L}$ is chosen based on the problem at hand. A commonly used loss function for regression problems is the mean squared error:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{2}\|y_i - F_i^L\|_2^2 = \frac{1}{N}\sum_{i=1}^{N} C_{x_i},$$

where y_i are targets and F_i^L the network's output corresponding to the ith input x_i.

One of the most established methods for solving the problem in (3.3) is gradient descent, in which we iteratively compute the vectors $\theta \in \mathbb{R}^p$ until they converge to a vector minimizing the loss function. At each step of this process, we choose some small perturbation $\Delta\theta$ such that the new parameter vector $\theta + \Delta\theta$ reduces the loss function. To try and simplify the problem at hand, we try to express the loss function as a Taylor series:

$$\mathcal{L}(\theta + \Delta\theta) = \mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)^T\Delta\theta + \frac{1}{2}\Delta\theta^T\nabla^2\mathcal{L}(\theta)\Delta\theta + \dots$$

We want the chosen perturbation to be small, which ensures that $\|\Delta\theta\|^n \to 0$ as $n \to \infty$. Hence, we can omit the quadratic term and all terms of higher order from the Taylor expansion.

Ignoring higher order terms, we want to minimize the loss in $\mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)^T\Delta\theta$. To do this, we subtract the largest value of $\nabla\mathcal{L}(\theta)^T\Delta\theta$ from $\mathcal{L}(\theta)$. Recalling the Cauchy-Schwartz inequality, we know that for $\nabla\mathcal{L}(\theta), \Delta\theta \in \mathbb{R}^p$, the inner product satisfies $|\nabla\mathcal{L}(\theta)^T\Delta\theta| \leq \|\nabla\mathcal{L}(\theta)\|_2\|\Delta\theta\|_2$. Expanding this equation gives

$$-\|\nabla\mathcal{L}(\theta)\|_2\|\Delta\theta\|_2 \leq \nabla\mathcal{L}(\theta)^T\Delta\theta \leq \|\nabla\mathcal{L}(\theta)\|_2\|\Delta\theta\|_2,$$

so the smallest $\nabla\mathcal{L}(\theta)^T\Delta\theta$ can get is $-\|\nabla\mathcal{L}(\theta)\|_2\|\Delta\theta\|_2$, which is true when $\Delta\theta = -\nabla\mathcal{L}(\theta)$. Hence, we set the update of the parameter vector to be

$$\theta \to \theta - \eta\nabla\mathcal{L}(\theta), \tag{3.4}$$

where $\eta > 0$ is a small scalar called the *learning rate*. This value is chosen according to the specific model at hand.

Example 3.4.1 *We illustrate gradient descent by considering the loss function* $\mathcal{L}(\theta) = 2\theta^2 + 3\theta + 1$. *Since this is a quadratic function, we know it is convex, and its critical point will be the global minimum. Now, the derivative is* $\nabla\mathcal{L}(\theta) = 4\theta + 3$. *Choose learning rate* $\eta = 0.2$. *Then, from equation 3.4 we get the update rule*

$$\theta_{k+1} = \theta_k - 0.2(4\theta_k + 3) = 0.2\theta_k - 0.6.$$

Consider the initial guess $\theta_0 = 0$. *We iteratively compute:*

$$\theta_1 = 0.2\theta_0 - 0.6 = -0.6$$

$$\theta_2 = 0.2\theta_1 - 0.6 = -0.72$$

$$\theta_3 = 0.2\theta_2 - 0.6 = -0.744$$

$$\vdots$$

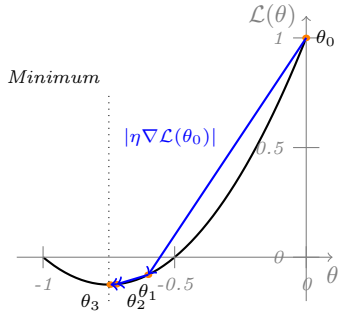


Figure 2 : Gradient descent iterations

The initial guess moves toward decreasing gradient and approaches the minimum value, as illustrated in Figure 2. Since this is a simple quadratic function, we can find analytically that the minimum is given by $\theta = -0.75$. *Just after three iterations, our choice of initial guess and learning rate gives an approximation* 0.006 *away from the minimum.*

Assume now that the loss is defined as

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} C_{x_i}, \quad \text{where } C_{x_i} := \frac{1}{2}\|y_i - F_i^L\|_2^2.$$

Then, the gradient of the loss function is given by:

$$\nabla\mathcal{L}(\theta) := \frac{1}{N}\sum_{i=1}^{N}\nabla C_{x_i}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\nabla\left(\frac{1}{2}\|y_i - F_i^L\|_2^2\right),$$

If the dataset is large, calculating the partial derivatives over all N points can become computationally expensive. For this reason, we sometimes implement *stochastic gradient descent*, in which we use the partial derivative of a training point chosen randomly from the N data points, making the update $\theta \to \theta - \eta\nabla C_{x_i}(\theta)$. However, this does not guarantee a reduction in the loss function for very small learning rates.

It is also common to use a method which finds a middle ground between the two extremes by using either one or all training points. For using a single point, we select $m \in \mathbb{N}$ training points $\{k_1, \ldots, k_m\}$ uniformly at random from the N for $m << N$. Then, these points $\{x_{k_i}\}_{i=1}^m$ are called a *minibatch* and the update rule becomes

$$\theta \to \theta - \frac{1}{m}\eta\sum_{i=1}^{m}\nabla C_{x_{k_i}}(\theta).$$

Both stochastic and minibatch gradient descent address the computational challenges of handling large datasets or parameter sets. Stochastic gradient descent reduces the computational power required by updating parameters based on a single randomly chosen data point, which allows for quick updates but introduces noise into the optimization process. Minibatch gradient descent is a compromise, using a subset of the data to find updates, which allows for both computational efficiency and gradient stability.

Despite the merits of these gradient descent methods, training neural networks often becomes problematic when the optimization problems are non-convex. Unlike convex problems with a single global minimum, non-convex problems may have local minima, saddle points or constant parts of the function. This makes the optimization tricky since the gradient descent solution may converge to points other than the global minimum where the gradient is zero. This means that neural network training does not always guarantee convergence to the global minimum of the loss function. Thus, although neural networks are powerful tools, their dependence on the convexity of the problem can introduce uncertainties.

3.5 The Mechanics of Learning: Backpropagation

When using a neural network, the main goal is to take the input x and allow the information to propagate through the hidden layers to yield an output y. The information flow through the network layers is called *forward propagation*. Sending the information forward in this way lets us compute the value of the loss function easily. However, finding the gradient using a forward pass alone can be computationally expensive and inefficient.

To address this problem, neural networks use a method known as *backpropagation*, an efficient algorithm that calculates the gradient of the loss function by propagating the errors backwards through the layers. Instead of recalculating the gradient of each layer separately, backpropagation uses the chain rule to reuse the previous result and reduce computational complexity.

We review backpropagation adapting notation from Higham and Higham (2019). Recall that in Section 3.4, we defined the loss as

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|y_i - F_i^L\|_2^2 = \frac{1}{N}\sum_{i=1}^{N} C_{x_i}.$$

We can use the fact that this is a linear combination of terms over training points and compute the partial derivatives of each term in the sum individually.

For simplicity of calculations, define $z^{[l]} = W^{[l]}F^{l-1} + b^{[l]}$ for layers $l = 2, 3, \ldots, L$, so that we can write $F_l = \sigma(z^{[l]})$[2]. Then, $z_j^{[l]}$ is the weighted input for neuron j at layer l.

We fix a training point x_i and drop the index i for simplicity of notation. Let $\delta^{[l]} \in \mathbb{R}^{N_l}$, where the jth entry $\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}$ is the "error" or sensitivity of C to the weighted input for neuron j at layer l.

Backpropagation consists of four steps, through which we aim to find the partial derivatives of the loss function with respect to the weights $W^{[l]} = (w_{jk}^{[l]})$ and biases $b_j^{[l]}$ for $j = 1, \ldots, N_l$ and $k = 1, \ldots, N_{l-1}$. We outline the four steps below.

1. **Find** F^L:

   Denote the input $F^1 = x \in \mathbb{R}^{N_1}$ and use a *forward pass* through the network to evaluate the output F^L:

   $$F^1 \to z^{[2]} \to F^2 \to z^{[3]} \to \cdots \to z^{[L]} \to F^L.$$

[2]Recall that $\sigma : \mathbb{R} \to \mathbb{R}$, so we perform computations componentwise. In particular, for vectors $x, y \in \mathbb{R}^{N_l}$, the componentwise product is $x \circ y$ with $(x \circ y)_i = x_i y_i$.

2. **Calculate** $\delta^{[L]}$:

   Notice that after we have calculated F^L and $z^{[L]}$, the "error" of the Lth layer can be computed directly using the chain rule:

$$\begin{aligned}\delta^{[L]} &= \frac{\partial C}{\partial z_j^{[L]}} = \frac{\partial C}{\partial F_j^L}\frac{\partial F_j^L}{\partial z_j^{[L]}} = \frac{\partial\left(\frac{1}{2}\|y - F^L\|_2^2\right)}{\partial F_j^L}\frac{\sigma(z_j^{[L]})}{\partial z_j^{[L]}} \\ &= \frac{\partial}{\partial F_j^L}\left(\frac{1}{2}\sum_{k=1}^{N_L}\left(y_k - F_k^L\right)^2\right)\sigma'(z_j^{[L]}) = (F_j^L - y_j)\sigma'(z_j^{[L]}).\end{aligned}$$

   Hence, we have $\delta^{[L]} = \sigma'(z^{[L]}) \circ (F^L - y)$ immediately available.

3. **Find** $\delta^{[L]}$ **for** $l = L-1, ..., 2$:

   First, note that since $z^{[l]} = W^{[l]}F^{l-1} + b^{[l]}$, we can express the relationship between the weighted input for a neuron j at layer l, $z_j^{[l]}$, and the weighted input for neuron k at layer $l+1$, $z_k^{[l+1]}$, as a sum of the activations from the previous layer, $\sigma(z_r^{[l]})$, scaled by the corresponding weights $w_{kr}^{[l+1]}$ that connect the neurons in layer l to neuron k in layer $l+1$. This sum is then shifted by the bias of neuron k:

$$z_k^{[l+1]} = \sum_{r=1}^{N_l}\left(w_{kr}^{[l+1]}\sigma(z_r^{[l]})\right) + b_k^{[l+1]}.$$

   Then, the derivative becomes

$$\frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = w_{kr}^{[l+1]}\sigma'(z_j^{[l]}).$$

   Now, the derivative for the lth layer can be calculated using the derivative for the $(l+1)$th layer. Indeed, we have

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}} = \sum_{k=1}^{N_{l+1}}\frac{\partial C}{\partial z_k^{[l+1]}}\frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = \sum_{k=1}^{N_{l+1}}\delta_k^{[l+1]}\frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = \sum_{k=1}^{N_{l+1}}\delta_k^{[l+1]}w_{kj}^{[l+1]}\sigma'(z_j^{[l]}).$$

   Hence, we can use $\delta^{[l]} = \sigma'(z^{[l]}) \circ \left((W^{[l+1]})^T\delta^{[l+1]}\right)$ to perform a *backwards pass* and find $\delta^{[l]}$ for all $l = L-1, ..., 2$:

$$\delta^{[L]} \to \delta^{[L-1]} \to \cdots \to \delta^{[3]} \to \delta^{[2]}.$$

4. **Find partial derivatives with respect to weights and biases**:

   Note that $z_j^{[l]} = \left(W^{[l]}\sigma(z_j^{[l]})\right)_j + b_j^{[l]}$ and, since $z_j^{[l-1]}$ does not depend on $b_j^{[l]}$, we have $\frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = 1$. Then, using the chain rule, the partial derivative with respect to bias for all $l = 2, ..., L$ is

$$\frac{\partial C}{\partial b_j^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}}\frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}}\cdot 1 = \delta_j^{[l]}.$$

   Noting that $\frac{\partial z_r^{[l]}}{\partial w_{jk}^{[l]}} = 0$ for all $r \neq j$, the partial derivative with respect to the weights is

$$\frac{\partial C}{\partial w_{jk}^{[l]}} = \sum_{r=1}^{N_l}\frac{\partial C}{\partial z_r^{[l]}}\frac{\partial z_r^{[l]}}{\partial w_{jk}^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}}\frac{\partial z_j^{[l]}}{\partial w_{jk}^{[l]}} = \delta_j^{[l]}\frac{\partial\left(\sum_{k=1}^{N_{l-1}}(w_{jk}^{[l]}F_k^{l-1}) + b_j^{[l]}\right)}{\partial w_{jk}^{[l]}} = \delta_j^{[l]}F_k^{l-1},$$

   so we get the partial derivatives $\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]}$ and $\frac{\partial C}{\partial w_{jk}^{[l]}} = \delta_j^{[l]}F_k^{l-1}$, components of which are available to us through the forward pass in step 2 and backward pass in step 3.

The four steps above: forward pass, error computation, backward pass and the calculation of gradients, give us an algorithm for iteratively improving the network's performance in an efficient way that accounts for the discrepancies in each layer. With these gradients, we can use optimization algorithms, such as stochastic gradient descent, to update the network parameters and minimize the loss function, enabling generalization to unseen data.

Chapter 4

Foundations of Financial Concepts

4.1 Historical Foundations and Modern Relevance

According to Akyıldırım and Soner (2014), one of the earliest records of financial engineering dates back to the 6th century BC, when the ancient Greek philosopher Thales of Miletus participated in a rudimentary form of call option trading. Expecting the olive harvest to be plentiful, Thales speculated on future demand by securing the rights to use local oil presses. While the breakthroughs of modern finance would not yet arrive for over two millennia, this example illustrates the ancient origins of financial concepts. By the 17th century, merchants were practicing simplistic options trading to hedge against risks in their trade. However, it was only through the works of Blaise Pascal and Pierre de Fermat, who shaped probability theory through their letters detailing mathematical approaches to gambling, that the foundations of modern financial mathematics were beginning to form.

Drawing from the rich history of finance by Cesa (2017), we review the developments of modern financial mathematics. The event that marks the birth of this field is Louis de Bachelier's thesis *Theory of Speculation*, which introduced Brownian motion to finance. Building on this, Kiyoshi Itô published the paper *On Stochastic Differential Equations* in 1951, presenting Itô's Lemma, which allowed the construction of stochastic differential equations needed for derivative valuation. In the 1970s, Black, Scholes, and Merton developed a model for pricing European options, revolutionizing the financial industry and accelerating the growth of the options market. Later advancements, such as the 1993 Heston model, Monte Carlo methods, and computational finance, have expanded the theory of financial mathematics, allowing for machine learning applications in pricing and hedging complex instruments.

Financial mathematics is essential in modern finance, providing pricing, hedging, and risk management methods. Mathematical tools and models bridge the gap between theoretical concepts and real-world markets. However, their reliance on assumptions makes it necessary to develop further and refine the field of mathematical finance.

4.2 Financial Instruments and Their Role in Markets

In order to discuss topics in financial mathematics, it is important to understand the core concepts. In this chapter, we briefly review financial derivatives and payoffs. Before defining financial derivatives, we first establish the framework in which they are traded.

Definition 4.2.1 (Financial Market) *A financial market consists of a probability space* $(\Omega, \mathcal{F}, \mathbb{P})$,

a filtration $\{\mathcal{F}_t\}_{t\geq 0}$, representing the flow of information, and a collection of tradable assets $S = (S^0, S^1, \ldots, S^d)$, where S^0 is a risk-free asset and $S^1, \ldots, S^d$ are risky assets.

The asset prices are adapted to the filtration $\{\mathcal{F}_t\}_{t\geq 0}$, while trading strategies are represented by portfolio processes δ.

In a financial market, we often trade *financial derivatives*, contracts whose value depends on an underlying asset, such as stocks, bonds, commodities, or currencies. They are also known as *contingent claims* because their value depends on the underlying asset's performance. There are many types of derivatives, but the more widely used ones are options.

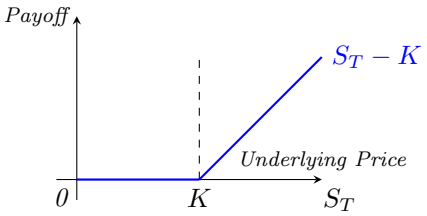
Definition 4.2.2 (Option) *Options are financial contracts that grant the holder the right, but not the obligation, to buy (call option) or sell (put option) an underlying asset at a specified price, also known as the strike price, on or before a predetermined expiration date, often called maturity.*

Other common derivatives include *forward contracts* and *future contracts*, both of which are agreements to trade an asset at a set price on a specific future date. Forwards are traded in over-the-counter markets, while futures are traded on an exchange. We refer to Hull (2021) for a more detailed review of various financial derivatives.

The value of derivatives results from their payoff structures, which define the financial loss or gain at the predetermined time of settlement, otherwise known as maturity. This can be visualized using a payoff diagram.

Example 4.2.3 (European Call Option Payoff) *The owner of a European call option on the underlying asset (e.g. stock) with value S_t at time t, has the right, but not the obligation, to buy the underlying asset at the expiration date or maturity T at the fixed strike price K. The corresponding payoff function and diagram are given by*

$$C(S_T, T) = (S_T - K)^+ = \begin{cases} S_T - K & \text{if } S_T > K, \\ 0 & \text{otherwise.} \end{cases}$$



While European options are exercised only at maturity, some options offer more flexibility. For example, American options can be exercised at any time before the expiration date. The owner of such an option holds the right, but not the obligation, to buy or sell the underlying asset at the strike price at any point up to the expiration date. Once again, the interested reader is referred to Hull (2021).

Derivatives enable risk management and efficient allocation of capital, thus playing a crucial role in financial markets. They allow market participants to hedge against adverse price movements and speculate on future market conditions.

4.3 Arbitrage Theory and Risk-Neutral Pricing

Arbitrage theory is at the core of financial mathematics, especially in pricing derivatives. In essence, arbitrage allows us to enforce consistency across markets by eliminating price inconsistencies. Intuitively, arbitrage opportunities allow market participants to profit without taking on risk. Price changes quickly eliminate such opportunities to make instantaneous and risk-free profits in financial markets. More rigorously, as stated in Föllmer and Schied (2004):

Definition 4.3.1 (Arbitrage Opportunity) *Consider a financial market over the time interval* $[0, T]$ *with* $d + 1$ *assets. Let* $S_0 = (S_0^0, \ldots, S_0^d) \in \mathbb{R}_{\geq 0}^{d+1}$ *be the vector of asset prices at the initial time* $t = 0$*, where* $S_0^i \geq 0$ *denotes the price of the* i*th asset. We fix a probability space* $(\Omega, \mathcal{F}, \mathbb{P})$*.*

A portfolio $\delta \in \mathbb{R}^{d+1}$*, where* δ^i *represents the number of shares of asset* i*, is called an arbitrage opportunity if:*

1. *the initial cost is non-positive:* $\delta \cdot S_0 \leq 0$*;*
2. *the terminal payoff is almost surely non-negative:* $\delta \cdot S_T \geq 0$ $\mathbb{P}$*-a.s.;*
3. *there is a strictly positive gain with positive probability:* $\mathbb{P}(\delta \cdot S_T > 0) > 0$*.*

Before we can price derivatives in a way that does not permit arbitrage, we need to understand how trading strategies behave over time. Define a self-financing condition as in Föllmer and Schied (2004).

Definition 4.3.2 (Self-Financing Trading Strategy) *A trading strategy* $\delta = (\delta_t)_{t=1}^{T-1}$ *is self-financing if for all trading times* $t = 1, ..., T - 1$ *we have*

$$\delta_t \cdot S_t = \delta_{t+1} \cdot S_t.$$

That is, for a self-financing strategy, the investor does not add or remove cash so that any change in portfolio value is due to the price movement of the assets only.

This condition ensures that any change in the portfolio's value over time is driven only by the market. It is an essential assumption for defining fair pricing.

An important tool for understanding fair pricing is the risk-neutral measure, which gives a mathematical framework for evaluating the price of financial instruments in a consistent and arbitrage-free way. Formally, as defined in Föllmer and Schied (2004):

Definition 4.3.3 (Risk-Neutral Measure) *A probability measure* $\mathbb{Q}$ *is called a risk-neutral measure (or equivalent martingale measure) if, under* $\mathbb{Q}$*, the discounted price process* $\left(\frac{S_t^i}{(1+r)^t}\right)_{t\geq 0}$ *of each tradable asset* S^i *is a martingale with respect to the natural filtration* $(\mathcal{F}_t)_{t\geq 0}$*. That is, for all* $t \geq 0$*,*

$$\mathbb{E}_{\mathbb{Q}}\left[\frac{S_{t+1}^i}{(1+r)^{t+1}} \,\middle|\, \mathcal{F}_t\right] = \frac{S_t^i}{(1+r)^{t+1}}.$$

Under this measure, the current value of any asset is the expected value of its discounted future price. This leads to the pricing formula

$$S_0^i = \mathbb{E}_{\mathbb{Q}}\left[\frac{S_T^i}{1+r}\right],$$

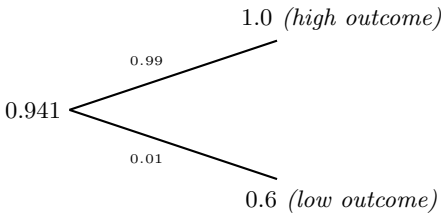
which gives the arbitrage-free price of asset i for $i = 0, ..., d$.

To illustrate the relationship between the artificial risk-neutral measure $\mathbb{Q}$ and the real-world probability measure $\mathbb{P}$, we consider an example adapted from McNeil et al. (2016).

Example 4.3.4 (Basic Binary Model) *Consider a one-period asset with maturity* $T = 1$ *year that pays either £*1 *or £*0.6 *at maturity. Suppose that undrr the real-world probability measure* $\mathbb{P}$*, the higher payoff occurs with probability* 0.99 *and the lower payoff occurs with probability* 0.01*. The risk-free interest rate is* $r = 0.05$*.*

Suppose that the current market price of the asset is $S_0 = 0.941$. The price of a risk-free bond maturing at time $T = 1$ is $B_0 = (1 + r)^{-1} = 0.952$.

We can view this as a collection of tradable assets given by $S = (B_0, S_0)$. The payoff for S_0 is illustrated on the right.



The expected discounted value of the asset under $\mathbb{P}$ is

$$\mathbb{E}_{\mathbb{P}}\left[\frac{S_1}{1+r}\right] = \frac{1}{1.05}(0.99 \cdot £1 + 0.01 \cdot £0.6) = £0.949.$$

Note that $0.949 > S_0 = 0.941$ shows that the asset's current market price is lower than its expected discounted value under the real-world measure. This price difference can be thought of as the compensation investors receive for taking on uncertainty.

The risk-neutral probability of the lower outcome, denoted q, is determined by ensuring that the discounted expected value under $\mathbb{Q}$ matches the market price, such that investing in the bond becomes a fair bet:

$$S_0 = \mathbb{E}_{\mathbb{Q}}\left[\frac{S_1}{1+r}\right] = \frac{1}{1.05}\left((1-q) \cdot £1 + q \cdot £0.6\right).$$

Solving this gives $q = 0.03$, which is higher than the real-world probability of the lower outcome, 0.01. This reflects the fact that under the risk-neutral measure, adverse outcomes are weighted more heavily to account for risk preferences.

The risk-neutal measure leads to one of the most fundamental results in financial mathematics, which links arbitrage-free markets and the existence of a risk-neutral measure. As stated in Föllmer and Schied (2004):

Theorem 4.3.5 (Fundamental Theorem of Asset Pricing) *Consider the set of equivalent risk-neutral measures*

$$\mathcal{P} = \{\mathbb{Q} : \mathbb{Q} \text{ is a risk-neutral measure with } \mathbb{Q} \approx \mathbb{P}\}.$$

Then, a market model is arbitrage-free if and only if $\mathcal{P} \neq \emptyset$.

By Theorem 4.3.5, the absence of arbitrage implies the existence of a risk-neutral measure in an arbitrage-free market. In practice, we often also want prices to be unique. This leads us to a deeper result.

Theorem 4.3.6 (Second Fundamental Theorem of Asset Pricing) *An arbitrage-free market model is complete if and only if there exists exactly one risk-neutral probability measure.*

4.4 Black-Scholes: The Formula That Changed Finance

Before creating any pricing or hedging model for derivatives, we need to understand and describe how the underlying asset behaves over time. This raises a fundamental question: how can we model the price of a stock in a way that reflects the market dynamics yet allows for mathematical formulation?

Example 4.4.1 *To illustrate the challenge at hand, consider the price movements of Standard & Poor's 500 index (S&P500) over the last five years. Figure 4.1 shows a plot of the daily closing prices from March 14, 2020, to March 14, 2025.*

Figure 4.1: S&P 500 Index Daily Closing Prices.

The price fluctuations seem random, drifting gradually or spiking suddenly. At first glance, the data seems chaotic, unpredictable and difficult to model mathematically.

Despite this seemingly random behavior, we must make assumptions about the behavior of stock prices over time when we attempt to create a financial model. We do not treat prices as deterministic trends. Instead, we make randomness an intrinsic part of the model, which leads us to stochastic processes. Brownian motion, named after a botanist Robert Brown, was initially used to describe the movement of pollen particles in liquid. This process models small and unpredictable fluctuations, which addresses the challenge presented in Example 4.4.1. As defined in Durett (2019):

Definition 4.4.2 (Brownian Motion) *A one-dimensional Brownian motion is a real-valued stochastic process* $\{B_t\}_{t\geq 0}$ *on some probability space* $(\Omega, \mathcal{F}, \mathbb{P})$ *that satisfies the following:*

(i) *The path of* $\{B_t\}_{t\geq 0}$, $t \mapsto B_t$, *is almost surely continuous.*

(ii) $\{B_t\}_{t\geq 0}$ *has independent increments, that is,* $\forall\ 0 \leq t_1 < t_2$, *the increment* $B_{t_2} - B_{t_1}$ *is independent of* $\{B_s | s \in [0,1]\}$.

(iii) $\{B_t\}_{t\geq 0}$ *has stationary increments, that is,* $\forall\ 0 \leq t_1 < t_2$, *we have* $B_{t_2} - B_{t_1} \sim B_{t_2 - t_1} \sim N(0, t_2 - t_1)$.

Brownian motion allows us to capture the chaotic way a stock price evolves over time, however, it is still not a good mathematical representation for price. Stock prices tend to grow over time, yet Brownian motion has a zero mean. This is addressed by generalized Brownian motion, $X_t = \mu_t + \sigma B_t$, $t \geq 0$, where $\{B_t\}_{t\geq 0}$ is a Brownian motion. This modification introduces drift μ, which models a deterministic trend over time, and volatility, σ, which scales the randomness of the process. While generalized Brownian motion is an improved model of the underlying asset's price, it still allows negative prices. Further, the variance is constant, whereas the volatility should scale with the price. Geometric Brownian motion ensures that the prices remain nonnegative and that fluctuations increase as the stock price grows.

Definition 4.4.3 (Geometric Brownian Motion) *Given a Brownian motion* $\{B_t\}_{t\geq 0}$, *drift* μ *and volatility* σ, *a stock price process* S_t *that follows a geometric Brownian motion satisfies the stochastic differential equation (SDE)*

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

We have arrived at an equation that models stock prices: the geometric Brownian motion SDE. However, this presents yet another challenge since introducing stochasticity in differential equations demands a set of mathematical tools different from those required for regular differential

equations. Unlike classical calculus, in which differentiation and integration follow well-behaved and familiar rules, Brownian motion introduces an issue - its paths behave in a fractal-like way, and so they are nowhere differentiable.

To address this, we introduce a type of calculus that takes randomness into account - stochastic calculus. As briefly mentioned in Section 4.1, stochastic calculus was first introduced in the 1950s by Kiyoshi Itô and soon became invaluable in derivative valuation. The key result is Itô's Lemma, which presents a way to express the differential of a function $f(S_t)$ where S_t is a stochastic process. For better understanding, we state the result and a simplified version of its proof, as in Itô (1951).

Lemma 4.4.4 (Itô's Lemma) *Let $B_t(\omega)$ be a Brownian motion on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with $\omega \in \Omega$, and let $f(x)$ be a twice continuously differentiable function. Then, we have*

$$\int_a^b f'(B_t(\omega))dB_t(\omega) = f(B_b(\omega)) - f(B_a(\omega)) - \frac{1}{2}\int_a^b f''(B_t(\omega))dt. \tag{4.1}$$

Sketch of Proof of Lemma 4.4.4. *For simplicity, assume that $a = 0$ and $b = 1$. The general case follows by a change of limits.*

We first discretize the interval. Define a partition of $[0, 1]$ with n subintervals: $t_k = \frac{k}{n}$ for $k = 0, 1, \ldots, n$. We decompose

$$f(B_1(\omega)) - f(B_0(\omega)) = \sum_{k=1}^{n} \left[f(B_{t_k}(\omega)) - f(B_{t_{k-1}}(\omega)) \right].$$

Defining $\Delta B_k := B_{t_k}(\omega) - B_{t_{k-1}}(\omega)$ and using a Taylor expansion around $B_{t_{k-1}}(\omega)$, we get

$$f(B_{t_k}(\omega)) = f(B_{t_{k-1}}(\omega)) + f'(B_{t_{k-1}}(\omega))\Delta B_k + \frac{1}{2}f''(B_{t_{k-1}}(\omega))\left(\Delta B_k\right)^2 + R_k,$$

where R_k is a higher-order remainder term. Now, summing over $k = 0, ..., n$, we have

$$f(B_1(\omega)) - f(B_0(\omega)) = \sum_{k=1}^{n} f'(B_{t_{k-1}}(\omega))\Delta B_k + \frac{1}{2}\sum_{k=1}^{n} f''(B_{t_{k-1}}(\omega))\left(\Delta B_k\right)^2 + \sum_{k=1}^{n} R_k.$$

Now, define the following:

- *The stochastic integral approximation: $I_1 = \sum_{k=1}^{n} f'(B_{t_{k-1}}(\omega))\Delta B_k$;*
- *The integral approximation of the drift term: $I_2 = \sum_{k=1}^{n} f''\left(B_{t_{k-1}}(\omega)\right)\frac{1}{n}$;*
- *The error in approximating variance: $I_3 = \sum_{k=1}^{n} f''\left(B_{t_{k-1}}(\omega)\right)\left(\left(\Delta B_k\right)^2 - \frac{1}{n}\right)$;*
- *The higher-order remainder term: $I_4 = \sum_{k=1}^{n} \delta(n, k, \omega)\left(\Delta B_k\right)^2$.*

$$f\left(B_1(\omega)\right) - f\left(B_0(\omega)\right) = I_1 + \frac{1}{2}I_2 + I_3 + I_4. \tag{4.2}$$

Now, we show that $I_4 \to 0$ in probability. The term I_4 involves a function $\delta(n, k, \omega)$ that vanishes uniformly as $n \to \infty$, given that f'' is continuous. We can choose a sufficiently large N for any $\delta > 0$ such that $n > N$ implies $\mathbb{P}\left(|\delta(n, k, \omega)| < \delta\right) > 1 - \delta$. Defining

$$\delta^*(n, k, \omega) = \begin{cases} \delta(n, k, \omega) & \text{if } |\delta(n, k, \omega)| < \delta \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad I_4^* = \sum_{k=1}^{n} \delta^*(n, k, \omega)\left(\Delta B_k\right)^2,$$

we have $\mathbb{P}(I_4 = I_4^*) > 1 - \delta$ and by expectation calculations, $\mathbb{E}\left[(I_4^*)^2\right] \leq n\delta^2\frac{1}{n} = \delta^2$, so $I_4 \xrightarrow{\mathbb{P}} 0$.

Now, we show that $I_3 \to 0$ in probability. By continuity of f'', we can choose a sufficiently large M for any $\delta > 0$ such that $\mathbb{P}\left(f''\left(B_{t_{k-1}}(\omega)\right) < M\right) > 1-\delta$. We define a truncated version of f'' to ensure boundedness and approximate I_3:

$$\mu_M(x) = \begin{cases} f''(x) & \text{if } |f''(x)| \le M \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad I_{3,M} = \sum_{k-1}^{n} \mu_M\left(B_{t_{k-1}}(\omega)\right)\left(\left(\Delta B_k\right)^2 - \frac{1}{n}\right).$$

Since $B_t(\omega)$ is a Brownian motion, $\Delta B_k \sim N(0, \frac{1}{n})$, so $\mathbb{E}\left[\left(\Delta B_k\right)^2\right] = \frac{1}{n}$. Using this, we obtain

$$\mathbb{E}\left[(I_{3,M})^2\right] \le M^2 \sum_{k-1}^{n} \mathbb{E}\left[\left(\left(\Delta B_k\right)^2 - \frac{1}{n}\right)^2\right] = \frac{M^2}{n}\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\left(x^2-1\right)^2 dx \to 0.$$

Thus, as $n \to \infty$, $I_3 \xrightarrow{\mathbb{P}} 0$. Similarly, using continuity of f'', it can be shown that $I_2 \to \int_0^1 f''\left(B_t(\omega)\right) dt$ and $I_1 \to \int_0^1 f'\left(B_t(\omega)\right) dB_t(\omega)$.

Putting this back into (4.2) *and rearranging yields the result* (4.1). □

To see how Itô's Lemma 4.4.4 applies to the model of a stock price in Definition 4.4.3, we adapt an example from Wilmott et al. (1995).

Example 4.4.5 *Suppose $f(S_t, t)$ is a smooth function of the stock price S_t that satisfies the SDE $dS_t = \mu S_t dt + \sigma S_t dB_t$. Note that $dt \to 0$, $(dB_t)^2 \to dt$ and $dt dB_t = o(dt) \to 0$. Also,*

$$(dS_t)^2 = \mu^2 S_t^2 (dt)^2 + \sigma^2 S_t^2 (dB_t)^2 + 2\mu\sigma S_t^2 dt dB_t \to \sigma^2 S_t^2 dt.$$

Applying a Taylor expansion, we get Itô's Lemma for S_t:

$$\begin{aligned} df &= f(S_t + dS_t,\ t + dt) - f(S_t, t) \\ &= \frac{\partial f}{\partial S}\cancelto{\to}{dS_t} \sigma S_t dB_t + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\cancelto{\to}{(dS_t)^2}^{\sigma^2 S_t^2 dt} + \frac{\partial f}{\partial t}\cancelto{\to}{dt}^{0} + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\cancelto{\to}{(dt)^2}^{0} + \frac{\partial^2 f}{\partial t \partial S}\cancelto{\to}{dt dS_t}^{0} + \dots \\ &= \left(\frac{\partial f}{\partial t} + \mu S_t \frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 f}{\partial S^2}\right) dt + \sigma S_t \frac{\partial f}{\partial S} dB_t. \end{aligned}$$

Now that we have built the mathematical framework for describing the evolution of a stock price over time, we want to construct a pricing model. Before Fischer Black, Myron Scholes and Robert Merton developed the Black-Scholes model, there were attempts to model the price of an option, however, all of them had significant drawbacks. As the authors outlined in Black and Scholes (1973), the previous models required an arbitrary discount rate, making the model overly dependent on the choice of r. Further, previous attempts relied on utility functions[1], making such models inherently subjective as no single utility function can represent all market participants. Pricing remained inconsistent until Black and Scholes (1973) outlined a framework assuming continuous trading and a frictionless market and constructing a self-financing portfolio that eliminates risk.

We now come to the question: under what conditions can we construct a portfolio that eliminates risk? By Black and Scholes (1973), the Black-Scholes model relies on six key assumptions.

1. The risk-free interest rate r is constant and known.
2. The stock prices are modelled by geometric Brownian motion as in Definition 4.4.3.
3. The underlying asset pays no dividends.
4. We use a European-style option that can only be exercised at maturity.
5. The markets are frictionless, so there are no transaction costs or taxes.

[1] Utility functions represent how much risk an investor is willing to take. We discuss utility functions in Subsection 4.5.2.

6. Trading happens continuously, and there are no short-selling or borrowing restrictions.

All of these assumptions play a role in deriving the Black-Scholes equation. However, some can be relaxed to reflect real market dynamics better, creating modified models. For instance, we can loosen assumptions 1, 3, 4, and 5 and adapt the model to incorporate interest rates that vary with time, dividends, different options, or transaction costs. However, assumptions 2 and 6 are structurally necessary. The assumption that S_t follows a geometric Brownian motion is fundamental in deriving the Black-Scholes partial differential equation (PDE). Also, the existence of a self-financing hedging strategy, a condition crucial to the model, relies on continuous and unrestricted trading.

Let us follow the derivation of the Black-Scholes PDE as in Black and Scholes (1973). Let the function $V(S_t, t)$ represent the price of a European-style option with maturity T and strike price K. Applying Lemma 4.4.4 to S_t, modelled by geometric Brownian motion, as in Example 4.4.5, we get

$$dV_t = \left(\frac{\partial V}{\partial t} + \mu S_t \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2}\right) dt + \sigma S_t \frac{\partial V}{\partial S} dB_t.$$

Now, we attempt to eliminate risk - hedge - by buying the option and short-selling Δ units of the stock S_t, resulting in the portfolio $\Pi_t = V_t - \Delta_t S_t$. The change in the value of the portfolio is

$$\begin{aligned} d\Pi &= dV_t - \Delta_t dS_t \\ &= \left(\frac{\partial V}{\partial t} + \mu S_t \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2}\right) dt + \sigma S_t \frac{\partial V}{\partial S} dB_t - \Delta_t \left(\mu S_t dt + \sigma S_t dB_t\right) \\ &= \mu S_t \left(\frac{\partial V}{\partial S} - \Delta_t\right) dt + \sigma S_t \left(\frac{\partial V}{\partial S} - \Delta_t\right) dB_t + \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2}\right) dt. \end{aligned}$$

We chose $\Delta_t = \frac{\partial V}{\partial S}$ because this specific choice means the term dB_t cancels out, eliminating randomness and making the portfolio risk-free. We have

$$d\Pi_t = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2}\right) dt. \tag{4.3}$$

Since the portfolio is now risk-free, it must grow at the risk-free rate r, hence we have $d\Pi_t = r\Pi dt$. Substituting (4.3) and $\Pi_t = V_t - \Delta_t S_t$, we finally arrive at the Black-Scholes PDE, the formula that transformed finance:

$$\frac{\partial V}{\partial t} + rS_t \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2} - rV_t = 0 \tag{4.4}$$

Deriving the analytic solutions to the Black-Scholes PDE (4.4) involves solving the backwards parabolic PDE or using numerous substitutions to transform it into a heat conduction equation. We omit this for brevity and state the solutions for the European call and put options $C(S_t, t)$ and $P(S_t, t)$, respectively. Define

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} \quad \text{and} \quad d_2 = d_1 - \sigma\sqrt{T-t}.$$

Then, if $\Phi(x)$ is the cumulative distribution function for the normal distribution, that is $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}s^2} ds$, the values of the call and put options are given by

$$C(S,t) = S\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2), \tag{4.5}$$

$$P(S,t) = Ke^{-r(T-t)}\Phi(-d_2) - S\Phi(-d_1). \tag{4.6}$$

For exact substitutions to arrive to the solutions given in (4.5) and (4.6), see Black and Scholes (1973) for a concise approach as well as Chapters 4 and 5 in Wilmott et al. (1995) for an in-depth explanation.

As with any model, its usefulness depends on how much the assumptions agree with reality. While the Black-Scholes model transformed the study of mathematical finance, real financial markets contain complexities and dynamics that this model overlooks.

As mentioned when presenting the assumptions for the model, many constraints can be relaxed. For instance, in real financial markets, volatility is not constant, so the Black-Scholes model fails to consider the dynamic risk present in markets. This is often addressed by using stochastic volatility models, such as the Heston model. Moreover, many companies that have their stocks traded publicly pay dividends, which reduce the stock price as soon as they are paid. As a result, the model overestimates the price of call options, which is adjusted for in the Black-Scholes-Merton model. Non-European-style options, such as American options, can be priced using numerical methods.

The Black-Scholes framework laid the groundwork for such adjustments. However, some limitations require a new modelling approach and cannot be fixed through minor adjustments. One of the fundamental assumptions of the model mentioned above is that the Black-Scholes framework assumes that the market is complete; that is, every contingent claim can be exactly replicated by trading a portfolio. In a complete market, no investor faces unhedgeable risk, so every payoff can be achieved using the available hedging instruments. In a real market, some risks, such as liquidity constraints, restrictions on short selling or borrowing, market discontinuities, and so on, cannot be eliminated through trading. This creates hedging errors and risks that the model cannot account for.

4.5 Risk Measures in Financial Decision-Making and Utility Theory

Risk is something that is encountered every day in the financial world. It determines how investors create portfolios, how banks decide to approve loans, and how traders protect themselves against sudden market swings. Every financial choice carries some uncertainty, so understanding and managing risk is essential for decision-makers. Risk measurement becomes quantitative in financial mathematics, helping anticipate losses, adjust investment strategies, and maintain stability. While people often associate risk with ups and downs in market prices, a more thorough approach is to view risk as the uncertainty about financial outcomes, variability in returns, or deviation from expected outcomes.

People have been dealing with financial uncertainty for centuries, making it difficult to pinpoint when risk measurement was born. Modern methods such as derivatives, credit models, and regulatory frameworks, such as Basel III, shape how risk is measured in mathematical finance. Since hedging is about reducing or managing risk in investment portfolios, accurately measuring risk is central to understanding pricing and hedging strategies. However, risk is not only about the mathematical framework but also about how people make decisions. Utility theory explores why some investors chase big rewards despite high risks while others prefer playing it safe. Understanding these human elements helps us create balanced strategies that manage uncertainty effectively in the constantly changing financial world.

4.5.1 Risk Measures: Quantifying Risk

In order to effectively evaluate and compare risk, we study risk measures. A risk measure is a function that quantifies the risk of a financial position by assigning a numerical value to its potential losses. Consider a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Let $\mathcal{X}$ be the space of all integrable random variables on Ω. If $X \in \mathcal{X}$ is a financial position, then a risk measure links X to a real number that specifies how risky holding the position X is.

When measuring risk, it is not enough to assign numerical values to potential losses, as this number also needs to align with real market dynamics. A good risk measure should reflect that

spreading investments out should reduce risk, respond appropriately to changes in the value of a portfolio and facilitate comparing different investment options.

Example 4.5.1 (Value-at-Risk) *Consider a risk measure popular throughout the evolution of mathematical finance, Value-at-Risk (VaR). For a confidence level $\alpha \in (0,1)$ and a financial position $X \in \Omega$, we define the Value-at-Risk as in Föllmer and Schied (2004):*

$$VaR_\alpha(X) = \inf_{m \in \mathbb{R}} \{\mathbb{P}(-X \leq m) \geq \alpha\}.$$

Value-at-Risk at level α tells us the smallest amount of capital we need to add to a financial position X, so the chance of incurring a loss greater than this amount is no more than $1-\alpha$. Despite the common usage of VaR, it has significant limitations. For instance, it does not always reward diversification and ignores extreme losses beyond the specified confidence level.

For risk measures to reflect reality better, Artzner et al. (1999) introduce coherent risk measures.

Definition 4.5.2 (Coherent Risk Measure) *A coherent risk measure is a mapping $\rho : \mathcal{X} \to \mathbb{R}$ which satisfies the following axioms for all financial positions $X, Y \in \mathcal{X}$ and all constants $\lambda, m \in \mathbb{R}$.*

(i) ***Translation (cash) invariance****: adding $m \in \mathbb{R}$ units of risk-free capital reduces the risk by m:*

$$\rho(X+m) = \rho(X) - m. \tag{4.7}$$

(ii) ***Subadditivity****: risk does not increase disproportionately when combining portfolios:*

$$\rho(X+Y) \leq \rho(X) + \rho(Y). \tag{4.8}$$

(iii) ***Positive homogeneity****: scaling a position also scales its risk:*

$$\rho(\lambda X) = \lambda \rho(X). \tag{4.9}$$

(iv) ***Monotonicity****: a riskier position should not have a lower risk measure:*

$$X \leq Y \text{ (a.s.)} \implies \rho(X) \geq \rho(Y). \tag{4.10}$$

Example 4.5.3 (Conditional Value-at-Risk is Coherent) *Conditional Value-at-Risk (CVaR) is a commonly used risk measure. It is also known as Expected Shortfall (ES) and Average Value-at-Risk (AVaR) and is given by*

$$CVaR_\alpha(X) = \frac{1}{\alpha}\int_0^\alpha VaR_s ds = \mathbb{E}[-X \mid -X \geq VaR_\alpha(X)].$$

CVaR represents expected loss given that the losses are larger than VaR at level α (see Example 4.5.1). Unlike VaR, which only gives a single quantile, it accounts for the entire tail risk. We can easily check the axioms using the integral representation to verify that CVaR is a coherent risk measure.

The main issue with coherent risk measures is that they assume positive homogeneity (4.9), which is unrealistic realistic in a real-world market. In Artzner et al. (1999), the authors state that if positions are not large enough so that the time needed to liquidate them depends on their sizes, we should consider the consequences of lack of liquidity when measuring risk. For illustration, suppose an investor has $X = £10M$ worth of an illiquid bond for which the $\text{CVaR}_{0.99}$ is 10% of the portfolio value. So, the worst 1% of scenarios or the left-side tail of the distribution results in an expected loss of $£1M$. Now, if we double the position to $2X = £20M$, by positive homogeneity, we have $\rho(2X) = 2\rho(X) = £2M$, however, the bond is illiquid, so selling a large volume causes the price to drop 25%, The trader then loses $0.25 \times £20M = £5M$, which is more than double than what was predicted by $\text{CVaR}_{0.99}$ according to the positive homogeneity axiom. This shows the need for even more robust risk measures. In their paper, Föllmer and Schied (2002) introduce convex risk measures which generalize coherent risk measures by dropping positive homogeneity (4.9) and subadditivity (4.8), and replace them with convexity.

Definition 4.5.4 (Convex Risk Measure) *A mapping $\rho : \mathcal{X} \to \mathbb{R}$ is a convex risk measure if it satisfies the monotonicity* (4.10) *and the translation invariance* (4.7) *axioms. In addition, for all $X, Y \in \mathcal{X}$ and $\lambda \in \mathbb{R}$, ρ must satisfy:*

***Convexity**: diversification does not increase risk:*

$$\rho(\lambda X + (1-\lambda)Y) \leq \lambda\rho(X) + (1-\lambda)\rho(Y). \tag{4.11}$$

Example 4.5.5 (Entropic Risk Measure is Convex) *The entropic risk measure quantifies risk using an exponential penalty, making it especially useful for handling uncertainty in portfolio optimization. In Föllmer and Schied (2004), the entropic risk measure is given by*

$$\rho(X) = \frac{1}{\beta}\log\mathbb{E}\left[e^{-\beta X}\right],$$

where $\beta > 0$ represents the investor's risk preferences. When β is large, the investor is highly risk averse and ρ heavily penalizes uncertain or negative outcomes. The entropic risk measure is convex.

Suppose $X \leq Y$ (a.s.). Since the function $e^{-\beta X}$ is decreasing, we have

$$e^{-\beta X} \geq e^{-\beta Y} \quad \Longrightarrow \quad \frac{1}{\beta}\log\mathbb{E}\left[e^{-\beta X}\right] \geq \frac{1}{\beta}\log\mathbb{E}\left[e^{-\beta Y}\right],$$

so ρ satisfies monotonicity (4.10)*. Now, take $m \in \mathbb{R}$. We have*

$$\rho(X+m) = \frac{1}{\beta}\log\mathbb{E}\left[e^{-\beta X-\beta m}\right] = \frac{1}{\beta}\left(\log e^{-\beta m} + \log\mathbb{E}\left[e^{-\beta X}\right]\right) = \frac{1}{\beta}\log\mathbb{E}\left[e^{-\beta X}\right] - m = \rho(X) - m.$$

Hence, ρ is cash invariant as in (4.7)*. Lastly, for $\lambda \in [0,1]$, we can use Jensen's inequality to get monotonicity* (4.10)

$$\begin{aligned}\rho(\lambda X + (1-\lambda)Y) &= \frac{1}{\beta}\log\mathbb{E}\left[e^{-\beta(\lambda X+(1-\lambda)Y)}\right] \leq \frac{1}{\beta}\mathbb{E}\left[\log e^{-\beta\lambda X-\beta(1-\lambda)Y}\right] \\ &= \frac{1}{\beta}\lambda\log\mathbb{E}\left[e^{-\beta X}\right] + \frac{1}{\beta}(1-\lambda)\log\mathbb{E}\left[e^{-\beta Y}\right] = \lambda\rho(X) + (1-\lambda)\rho(Y),\end{aligned}$$

and so ρ is convex.

4.5.1.1 Acceptance Sets and Capital Requirements

Having grasped the idea of risk measures, we come to a concept of acceptance sets as defined in Föllmer and Schied (2002).

Definition 4.5.6 (Acceptance Set) *Each risk measure ρ is associated with an acceptance set, given by*

$$A_\rho = \{X \in \mathcal{X} : \rho(X) \leq 0\}.$$

Conversely, for any given acceptance set we can define a risk measure

$$\rho_A(X) = \inf_{m\in\mathbb{R}}\{m + X \in A\}.$$

The acceptance set A_ρ contains all financial positions that are considered "acceptable" under the risk measure ρ. Since, for any financial position, $\rho(X)$ measures the risk or expected loss, a positive value indicates that we expect to incur a loss. Looking back at the definition of A_ρ, $\rho(X) \leq 0$ indicates that we accept financial positions for which we expect negative losses, that is, profits.

Similarly, looking at how ρ_A is defined in Definition 4.5.6, we see that

$$m + X \in A \quad \Longleftrightarrow \quad \rho(X+m) \leq 0 \quad \Longleftrightarrow \quad \rho(X) \leq m$$

by the cash invariance axiom (4.7). This reflects that $\rho(X)$ can be viewed as a capital requirement or the smallest amount of money we can add to the position X to make it acceptable.

Example 4.5.7 *Given a confidence lebel $\alpha \in (0,1)$, suppose a financial position $X \in \mathcal{X}$ is considered acceptable if $\mathbb{P}(X < 0) \leq \alpha$. The associated risk measure is defined by*

$$\rho_A(X) = \inf_{m \in \mathbb{R}} \{\mathbb{P}(X + m < 0) \leq 1 - \alpha\},$$

which coincides with $VaR_\alpha(X)$. Note that in Example 4.5.1, Value-at-Risk is represented in its distribution-based form because it represents the α-quantile of $-X$. Here, Value-at-Risk is given in its capital requirement form - minimum capital required to ensure that the probability of exceeding losses is below $1 - \alpha$.

Risk measures not only quantify possible losses, but they also help shape important decisions like pricing investments and choosing hedging strategies. By assigning numerical values to risk and linking these numbers to acceptance sets, we create a practical framework that allows us to judge whether a financial position is safe enough or if it needs extra capital to cover potential losses. However, good risk measurement is not just about mathematics - it is also about understanding human behavior and recognizing that people and institutions often see and respond to risk very differently.

4.5.2 Utility Theory: The Link Between Risk and Reward

Risk measures link financial mathematics and decision theory, providing us with methods for optimal decision-making when outcomes are uncertain. In finance, we encounter two main types of decision theory: descriptive decision theory, which studies how decisions are *actually made*, explaining investor behavior and risk aversion, and normative decision theory, which explores how decisions *should be made* based on utility maximization and used in asset pricing. We focus on the latter, in particular, utility theory.

Utility theory is a powerful way to approach financial decision-making in the presence of uncertain outcomes. In essence, utility theory helps describe an investor's approach to risk and the trade-offs they are willing to accept. Initially introduced by von Neumann and Morgenstern through an axiomatic approach, this theory gives us a structured way to evaluate and compare uncertain outcomes. The main idea is simple: investors do not just look at expected profits - they also care about the risk involved. This is where utility functions become useful, turning personal risk preferences into something measurable. We define a utility function as in Föllmer and Schied (2004).

Definition 4.5.8 (Utility Function) *Suppose $\mathcal{M}$ is a fixed set of Borel probability measures on the fixed interval $S \subset \mathbb{R}$. Then, a function $u : S \to \mathbb{R}$ is called a utility function if it is strictly concave, strictly increasing and continuous on S. A von Neumann-Morgenstern representation*

$$U(\mu) = \int_S u d\mu = \mathbb{E}_\mu \left[u(S)\right],$$

where $\mu \in \mathcal{M}$ is called an expected utility representation.

Intuitively, a utility function represents an individual's preference over wealth. The utility function u is strictly increasing, showing that more money is better and concave, representing risk aversion. If u is normalized, $\mathbb{E}\left[u(X)\right]$ represents a certain value of an uncertain future outcome X. To express an investor's preferences more rigorously, we say that a risk measure ρ imposes a *preference order* $\succeq$ on random variables, namely

$$X \succeq Y \quad \iff \quad \rho(X) \leq \rho(Y).$$

Then, as explained in Ben-Tal and Teboulle (2007), for an individual with a utility function u, the preference order under the von Neurmann and Morgernstern's expected utility theory is

$$X \succeq Y \quad \iff \quad \mathbb{E}\left[u(X)\right] \geq \mathbb{E}\left[u(Y)\right].$$

Notice that the inequality changes direction when we switch between risk measures and expected utility. This expresses the preference for a smaller loss; in the case where a risk measure is convex, this also reflects the concavity of the utility function.

For a random variable $X \in \mathcal{X}$, Ben-Tal and Teboulle (2007) define the *certainty equivalent* as

$$C_u(X) = u^{-1}\left(\mathbb{E}\left[u(X)\right]\right).$$

This certainty equivalent allows us to quantify how much an investor is willing to pay to eliminate risk. The difference between the expected value and the certainty equivalent is known as the risk premium, representing the cost of uncertainty.

While the certainty equivalent provides a useful way to measure an investor's risk-adjusted valuation of uncertain outcomes, it has some limitations. One drawback is that it directly depends on the utility function, making it less flexible in expressing diverse risk preferences in different contexts. Moreover, since it strictly follows the form of the given utility function, it may not adapt well to constraints, such as capital requirements or regulations.

To address these issues, Ben-Tal and Teboulle (2007)[2]introduce the Optimized Certainty Equivalent (OCE), a more general and flexible extension of the certainty equivalent.

Definition 4.5.9 (Optimized Certainty Equivalent (OCE)) *Let u be a utility function and consider the financial position $X \in L^\infty(\Omega, \mathcal{F}, \mathbb{P})$. The Optimized Certainty Equivalent (OCE) of an uncertain outcome X is defined by the map $S_u : L^\infty \to \mathbb{R}$,*

$$S_u(X) = \sup_{w \in \mathbb{R}} \{w + \mathbb{E}\left[u(X - w)\right]\}.$$

Rather than just taking the inverse of a utility function, OCE works by finding the ideal balance between immediate gains and future wealth. By doing this, OCE naturally explains a broad spectrum of investors' attitudes to risk, making it more flexible than the certainty equivalent for optimizing portfolios and hedging.

The idea behind OCE is straightforward: investors constantly choose between spending some wealth immediately and saving the remainder for the uncertain future. Suppose an investor expects to receive $£X$ in the future, but instead of waiting, they can consume a part of it immediately. If they decide to consume $£w$ now, they are left with $£(X - w)$ for the future, which is uncertain. However, the remaining wealth still holds value, which is captured by their expected utility $\mathbb{E}\left[u(X - w)\right]$. Hence, the total present value of their wealth is given by $w + \mathbb{E}\left[u(X - w)\right]$.

The OCE reflects that the investor wants to allocate their wealth in the best possible way, optimally splitting between what they consume now and what they save for the uncertain future. The OCE is then defined as the best possible value of this trade-off - the maximum amount that represents the sure value of X today. In other words, the investor looks for the optimal value of w, making the current allocation as favorable as possible.

We adapt an example from Ben-Tal and Teboulle (2007).

Example 4.5.10 (Piecewise linear utility function) *To see how the OCE connects to Conditional Value-at-Risk, we start with a simple piecewise linear utility function. We define*

$$u(t) = \begin{cases} \gamma_2 t & \text{if } t \leq 0, \\ \gamma_1 t & \text{if } t > 0, \end{cases}$$

[2]While OCE was originally introduced in their 1986 paper *Expected Utility, Penalty Functions and Duality in Stochastic Nonlinear Programming*, this paper focuses on programming and precedes the introduction of both coherent and convex risk measures. We use their 2007 paper *An Old-New Concept of Convex Risk Measures: The Optimized Certainty Equivalent*, which frames the OCE as a risk measure and targets an audience in finance.

where $0 \leq \gamma_1 < 1 < \gamma_2$. Notice that $u(t)$ is concave if $\gamma_2 > \gamma_1$. In practical modeling, we often work with piecewise linear utility functions that are concave but not strictly because they still capture risk-averse behavior in a weaker form. Here, risk-aversion is expressed through asymmetry - the function penalizes losses more than it does rewards.

Denoting $[t]_+ = \max\{0, t\}$, we can rewrite the utility function as $u(t) = \gamma_1[t]_+ - \gamma_2[-t]_+$, which gives the OCE

$$S_u(X) = \sup_{w\in\mathbb{R}}\{w - \gamma_2\mathbb{E}[-w - X]_+ - \gamma_1\mathbb{E}[X + w]_+\}.$$

Now, define a risk measure by $\rho(X) := -S_u(X)$, which gives

$$\rho(X) = \inf_{w\in\mathbb{R}}\{w - \gamma_2\mathbb{E}[-w - X]_+ - \gamma_1\mathbb{E}[X + w]_+\}.$$

Consider a special case where $\gamma_1 = 0$, so the investor only focuses on losses, and $\alpha = \frac{1}{\gamma_2} \in (0, 1)$, so γ_2 controls risk aversion. We can simplify the risk measure to

$$\rho(X) = \inf_{w\in\mathbb{R}}\left\{w + \frac{1}{\alpha}\mathbb{E}[-w - X]_+\right\}.$$

This is a representation formula for Conditional Value-at-Risk, $CVaR_\alpha(X)$.

We have just seen that choosing a piecewise linear utility allows us to recover the coherent and convex risk measure CVaR from the OCE. Naturally, we want to question whether the Optimized Certainty Equivalent has a connection with risk measures. We state a general result and its proof from Buehler et al. (2019) which shows that, in fact, OCE generates a family of convex risk measures.

Lemma 4.5.11 *Let $u : \mathbb{R} \to \mathbb{R}$ be a utility function, satisfying the conditions stated in Definition 4.5.8. Then, the risk measure defined by*

$$\rho(X) := -S_u(X) = \inf_{w\in\mathbb{R}}\{-w - \mathbb{E}[u(X - w)]\}$$

is a convex risk measure.

Proof of Lemma 4.5.11. *Let $X, Y \in \mathcal{X}$.*

1. ***Monotonicity**. We assume $X \leq Y$ (a.s.). Since u is strictly increasing, we have $\forall\, w \in \mathbb{R}$,*

$$u(X - w) \leq u(Y - w) \quad \Longrightarrow \quad -\mathbb{E}[u(X - w)] \geq -\mathbb{E}[u(Y - w)].$$

Subtracting w from both sides and taking the infimum gover $\mathbb{R}$ gives

$$\rho(X) = \inf_{w\in\mathbb{R}}\{-w - \mathbb{E}[u(X - w)]\} \geq \inf_{w\in\mathbb{R}}\{-w - \mathbb{E}[u(Y - w)]\} = \rho(Y).$$

2. ***Cash Invariance**. Fix $m \in \mathbb{R}$. We have*

$$\rho(X + m) = \inf_{w\in\mathbb{R}}\{-(w - m) - m - \mathbb{E}[u(X - (w + m))]\}.$$

Set $w' := w - m$, then

$$\rho(X + m) = \inf_{w'\in\mathbb{R}}\{-w' - m - \mathbb{E}[u(X - w')]\} = \rho(X) - m.$$

3. ***Convexity**. Fix $\lambda \in [0, 1]$. By the convexity of $-u$ and by splitting w into $w_1, w_2 \in \mathbb{R}$, we have*

$$\begin{aligned}
\rho(\lambda X + (1 - \lambda)Y) &= \inf_{w\in\mathbb{R}}\{-w - \mathbb{E}[u(\lambda X + (1 - \lambda)Y - w)]\} \\
&= \inf_{w_1,w_2\in\mathbb{R}}\{-\lambda w_1 - (1 - \lambda)w_2 + \mathbb{E}[u(\lambda(X - w_1) + (1 - \lambda)(Y - w_2))]\} \\
&\leq \inf_{w_1,w_2\in\mathbb{R}}\{-\lambda w_1 - \lambda\mathbb{E}[u(X - w_1)] - (1 - \lambda)w_2 - (1 - \lambda)\mathbb{E}[u(Y - w_2)]\} \\
&\leq \lambda \inf_{w_1\in\mathbb{R}}\{-w_1 - \mathbb{E}[u(X - w_1)]\} + (1 - \lambda) \inf_{w_2\in\mathbb{R}}\{-w_2 - \mathbb{E}[u(Y - w_2)]\} \\
&= \lambda\rho(X) + (1 - \lambda)\rho(Y).
\end{aligned}$$

□

4.5.3 Dual Representation of Convex Risk Measures

While utility theory connects risk measures to individual preferences, convex risk measures also have a form useful for optimization: the dual or robust representation. This form represents the worst-case expected loss over all probability measures, which fits perfectly with machine learning.

The following result from Föllmer and Schied (2002) generalizes a result from Artzner et al. (1999), originally presented for coherent risk measures.

Theorem 4.5.12 (Robust (Dual) Representation of Convex Risk Measures) *Suppose $\mathcal{X}$ is the space of all real-valued functions on a finite set Ω and $\mathcal{P}$ is a set of probability measures on $(\Omega, \mathcal{F})$. Then, $\rho : \mathcal{X} \to \mathbb{R}$ is a convex measure of risk if and only if there exists a penalty function $\alpha : \mathcal{P} \to (-\infty, \infty]$ such that for $X \in \mathcal{X}$*

$$\rho(X) = \sup_{\mathbb{Q} \in \mathcal{P}} \{\mathbb{E}_{\mathbb{Q}}[-X] - \alpha(\mathbb{Q})\}.$$

The penalty function satisfies $\alpha(\mathbb{Q}) \geq -\rho(0)$ for any $\mathbb{Q} \in \mathcal{P}$, and it is convex and lower semi-continuous on $\mathcal{P}$.

We can think of $\alpha(\mathbb{Q})$ as the equivalent of a regularization term often used in machine learning. It controls how much each probability measure $\mathbb{Q}$ contributes to the overall risk. Instead of treating all possible scenarios equally, α assigns a penalty to each probability measure. If $\mathbb{Q}$ makes a risky position appear safer, $\alpha(\mathbb{Q})$ is smaller, ensuring that the measure contributes to the risk. However, if $\mathbb{Q}$ is further from the real-world probability measure $\mathbb{P}$, $\alpha(\mathbb{Q})$ is significantly higher so that the model relies on $\mathbb{Q}$ less. Moreover, α is the minimal penalty function that represents ρ, that is, if there exists another penalty function α^*, we then must have $\alpha(\mathbb{Q}) \leq \alpha^*(\mathbb{Q})$ for all $\mathbb{Q} \in \mathcal{P}$. We state the proof of this result to build intuition.

Proof of Theorem 4.5.12. *Let us define, for each $\mathbb{Q} \in \mathcal{P}$,*

$$\alpha(\mathbb{Q}) := \sup_{X \in \mathcal{X}} \{\mathbb{E}_{\mathbb{Q}}[-X] - \rho(X)\}.$$

This function measures how much worse the expectation under $\mathbb{Q}$ is compared to ρ accross all X. This α is well-defined and convex, since it is a supremum of affine functions[3]. We now show that for some $Y \in \mathcal{X}$

$$\rho(Y \sup_{\mathbb{Q} \in \mathcal{P}} \{\mathbb{E}_{\mathbb{Q}}[-Y] - \alpha(\mathbb{Q})\}.$$

1. *First, use the definition of α to find an upper bound. Since, by definition of supremum,*

$$\alpha(\mathbb{Q}) \geq \mathbb{E}_{\mathbb{Q}}[-Y] - \rho(Y) \quad \Longrightarrow \quad \mathbb{E}_{\mathbb{Q}}[-Y] - \alpha(\mathbb{Q}) \leq \rho(Y),$$

   taking the supremum over all $\mathbb{Q} \in \mathcal{P}$ we get

$$\sup_{\mathbb{Q} \in \mathcal{P}} \{\mathbb{E}_{\mathbb{Q}}[-Y] - \alpha(\mathbb{Q})\} \leq \rho(Y).$$

2. *For the lower bound, suppose $\rho(Y) > m$ for some $m \in \mathbb{R}$. We want to show that*

$$m < \sup_{\mathbb{Q} \in \mathcal{P}} \{\mathbb{E}_{\mathbb{Q}}[-Y] - \alpha(\mathbb{Q})\}.$$

   For this, we use an acceptance set, $A_\rho = \{X \in \mathcal{X} : \rho(X) \leq 0\}$. This is convex and closed because $\mathcal{X}$ is finite-dimensional and ρ is convex and lower semicontinuous.

   Then, since we picked Y such that $\rho(Y) > m$, $Y - m \notin A_\rho$. Indeed, by cash invariance (4.7), we have $\rho(Y - m) = \rho(Y) - m > 0$. By Theorem 2.0.5, we can separate $Y - m$ from

A_ρ using a linear functional. This function is just the expectation under some $\mathbb{Q} \in \mathcal{P}$. This gives

$$\mathbb{E}_\mathbb{Q}[-(Y-m)] < \inf_{X \in A_\rho} \mathbb{E}_\mathbb{Q}[-X] =: \alpha(\mathbb{Q}) \quad \Longrightarrow \quad \mathbb{E}_\mathbb{Q}[-Y] - \alpha(\mathbb{Q}) > m.$$

We have found a probability measure such that the above holds, hence, this also holds for the supremum $\mathcal{P}$, hence

$$\sup_{\mathbb{Q} \in \mathcal{P}} \{\mathbb{E}_\mathbb{Q}[-Y] - \alpha(\mathbb{Q})\} \geq m.$$

Since this is true for all $m < \rho(Y)$, we have

$$\rho(Y) \leq \sup_{\mathbb{Q} \in \mathcal{P}} \{\mathbb{E}_\mathbb{Q}[-Y] - \alpha(\mathbb{Q})\}.$$

Since α is defined as a supremum of continuous functions, it is lower semicontinuous. As discussed above, it is also convex. Moreover, for any $\mathbb{Q} \in \mathcal{P}$, we can take $X = 0$ to obtain $\alpha(\mathbb{Q}) \geq \mathbb{E}_\mathbb{Q}[0] - \rho(0) = -\rho(0)$.

□

The finite probability space Ω is assumed in the framework described in Section 5.2 for simplicity. For the proof of the result for non-discrete probability spaces, see Section 4.2 in Föllmer and Schied (2004).

We now derive the robust representation of the convex risk measure entropic risk, adapted from McNeil et al. (2016).

Example 4.5.13 (Dual Representation of Entropic Risk) *Let ρ_β be the entropic risk measure as in Example 4.5.5 and let $X \in \mathcal{X}$. We show that the dual representation of ρ_β is given by*

$$\rho_\beta(X) = \sup_{\mathbb{Q} \in \mathcal{P}} \left\{ \mathbb{E}_\mathbb{Q}[X] - \frac{1}{\beta} H(\mathbb{Q}|\mathbb{P}) \right\}, \quad \text{where } H(\mathbb{Q}|\mathbb{P}) = \begin{cases} \mathbb{E}_\mathbb{Q}\left[\log \frac{d\mathbb{Q}}{d\mathbb{P}}\right] & \text{if } \mathbb{Q} << \mathbb{P}^4, \\ \infty & \text{otherwise.} \end{cases}$$

We will prove

$$\frac{1}{\beta} \log \mathbb{E}_\mathbb{P}\left[e^{\beta X}\right] = \sup_{\mathbb{Q} << \mathbb{P}} \left\{ \mathbb{E}_\mathbb{Q}[X] - \frac{1}{\beta} H(\mathbb{Q}|\mathbb{P}) \right\}.$$

Denote by Z the Radon-Nikodym derivative, i.e. $Z := \frac{d\mathbb{Q}}{d\mathbb{P}}$. Since $\mathbb{Q} << \mathbb{P}$, Z exists and satisfies the following:

(i) $Z \geq 0$,

(ii) $\mathbb{E}_\mathbb{Q}[Z] = 1$,

(iii) $\mathbb{E}_\mathbb{Q}[X] = \mathbb{E}_\mathbb{P}[XZ]$,

(iv) $H(\mathbb{Q}|\mathbb{P}) = \mathbb{E}_\mathbb{P}[Z \log Z]$.

So, in terms of Z, we want to show

$$\frac{1}{\beta} \log \mathbb{E}_\mathbb{P}\left[e^{\beta X}\right] = \sup_{Z \geq 0, \mathbb{E}_\mathbb{P}[Z]=1} \left\{ \mathbb{E}_\mathbb{P}[XZ] - \frac{1}{\beta} \mathbb{E}_\mathbb{P}[Z \log Z] \right\}. \tag{4.12}$$

We use the variational formula (see Chapter 3 in Föllmer and Schied (2004)), which states that $\forall X \in \mathcal{X}$,

$$\log \mathbb{E}_\mathbb{P}\left[e^{\beta X}\right] \sup_{Z \geq 0, \mathbb{E}_\mathbb{P}[Z]=1} \{\beta \mathbb{E}_\mathbb{P}[XZ] - \mathbb{E}_\mathbb{P}[Z \log Z]\}.$$

Dividing both sides by $\beta > 0$ then gives (4.12). Substituting $\mathbb{E}_\mathbb{Q}[X] = \mathbb{E}_\mathbb{P}[XZ]$ and $H(\mathbb{Q}|\mathbb{P}) = \mathbb{E}_\mathbb{P}[Z \log Z]$ in the right-hand-side then yields the result.

[3] An affine function is that of the form $f(x) = ax + b$. An affine function is linear and so it is both convex and concave. The supremum of convex functions is then also convex.

When optimizing machine learning models, we often use the Mean Squared Error (MSE) as the loss function, which is insufficient to capture decision-making in the presence of risk. We want to control for extreme losses rather than just mean losses. Using convex measures of risk as loss functions addresses this problem. When we restate convex risk measures in their robust form, we introduce a penalty function, which penalizes the distributions $\mathbb{Q}$ based on their attitude towards risk.

The entropic risk measure is an intuitive example for this concept because of the form its penalty function takes. In Example 4.5.13, the penalty function is the Kullback-Leibler (KL) divergence or the relative entropy, $\alpha(\mathbb{Q}) = H(\mathbb{Q}|\mathbb{P})$, as the name of the risk measure hints. For some $\omega \in \Omega$, the Radon-Nikodym derivative $\frac{d\mathbb{Q}}{d\mathbb{P}}(\omega)$ is the relative likelihood of the outcome ω under $\mathbb{Q}$, compared to $\mathbb{P}$. It explains how to adjust expectations or probabilities under $\mathbb{P}$ to reflect the new measure. When we the expected value, $H(\mathbb{Q}|\mathbb{P})$ then tells us how surprising is an outcome as represented by $\mathbb{Q}$, when we expect the real probability to be as measured by $\mathbb{P}$. The penalty here increases when $\mathbb{Q}$ moves away from $\mathbb{P}$ in relative entropy. That is, if the distribution $\mathbb{Q}$ is close to $\mathbb{P}$, α assigns it a lower penalty, making it more influential on the risk measure. Hence, $\alpha(\mathbb{Q})$ acts as a regularization term, discouraging the model from overvaluing extreme tail-loss scenarios unless they are sufficiently likely under $\mathbb{P}$.

The penalty term ensures that the model prioritizes avoiding large losses rather than just maximizing returns. Adjusting the returns for risk in this way makes hedging more effective.

4.5.4 Semicontinuity and Convergence Properties

An important consideration when applying risk measures in practice, especially in machine learning, is semicontinuity.

Definition 4.5.14 (Semicontinuity) *A function f is lower semicontinuous if, for every sequence $x_n \to x$, we have*

$$\liminf_{n\to\infty} f(x_n) \geq f(x).$$

A function f is upper semicontinuous if, for every sequence $x_n \to x$, we have

$$\limsup_{n\to\infty} f(x_n) \leq f(x).$$

Intuitively, we can view semicontinuous functions as functions that can only have discontinuities in one direction. This is illustrated in Figure 4.2. A lower semicontinuous function can jump down but not up, so there are no sudden increases. It is essential for optimization problems, particularly ensuring that a minimizer exists. On the other hand, an upper semicontinuous function can jump up but not down, so there are no sudden increases. It appears in duality results discussed in Subsection 4.5.3 and is important in ensuring that limits preserve inequalities.

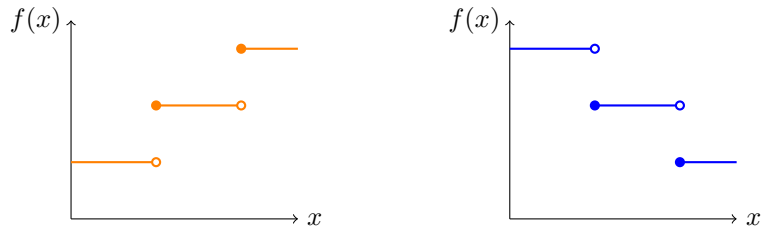


Figure 4.2: Upper (left) and lower (right) semicontinuous functions.

[4] $\mathbb{Q}$ is absolutely continuous with respect to $\mathbb{P}$. The Radon-Nikodym derivative $\frac{d\mathbb{Q}}{d\mathbb{P}}$ exists only under this condition.

4.5.4.1 Semicontinuity and The Fenchel-Legendre Transform

The Fenchel-Legendre transform, also known as the convex conjugate, is fundamental in convex duality and variational representations, as we have already seen in Example 4.5.13. It reflects how well a function can be approximated by linear functionals. In Föllmer and Schied (2004), it is defined as follows.

Definition 4.5.15 (Fenchel-Legendre Transform) *For a function $f : \mathbb{R} \to \mathbb{R} \cup \{\infty\}$, the Fenchel-Legendre transform is given by*

$$f^*(y) = \sup_{x \in \mathbb{R}} \{yx - f(x)\}, \quad y \in \mathbb{R}.$$

The convex conjugate tells us how well you can approximate the function f from below using lines of slope y.

Example 4.5.16 *Consider the convex function $f(x) = x^2$, as shown in Figure 4.3. The left plot shows a function a single line with slope $y = 1$, tangent to f at $x = 0.5$. This line is the best linear lower bound on f with the slope 1, that is $f^*(1) = \sup_{x \in \mathbb{R}} \{x - x^2\} = x - 0.25$. On the right, we can see several such linear functionals for various values of y. They form an outer boundary of the linear functionals f^*, which reconstructs our function f.*

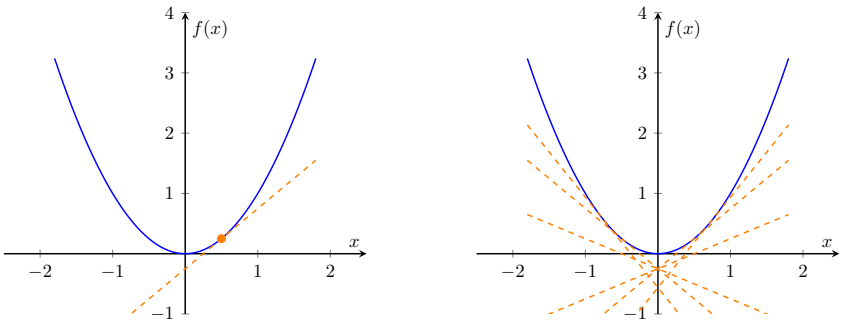


Figure 4.3: Visualisation of the Fenchel-Legendre transform, showing $f^*(1)$ (left), and the lines $f^*(y)$ for $y = -1.5, -1, -0.5, 0.5, 1, 1.5$ (right).

The example illustrated above shows how a convex function is completely determined by its linear estimators. Formally, this is given by the Fenchel-Moreau theorem, which states that a convex function can be reconstructed as the supremum of such linear estimators. We do not explore this result in detail and refer the interested reader to Föllmer and Schied (2004) Appendix Section A.1.

The Fenchel-Legendre transform is then given by $\rho^*(\ell) = \sup_{X \in \mathcal{X}} \{\ell(X) - \rho(X)\}$, which represents the best linear overestimator of the risk measure. Since $\mathcal{X}$ is finite-dimensional, ρ^* is then lower semicontinuous and convex in the same way as argued in the proof of Theorem 4.5.12, and so the penalty function also has these properties.

The Fenchel–Moreau theorem applied to ρ gives back its robust representation:

$$\rho(X) = \sup_{\mathbb{Q} \in \mathcal{P}} \{\mathbb{E}_{\mathbb{Q}}[-X] - \alpha(\mathbb{Q})\}.$$

It defines the risk as the worst-case expected loss minus the penalty of each measure $\mathbb{Q}$ relative to the real-life probability measure.

Moreover, the set $\mathcal{P}$ is compact in the discrete case, so the supremum is attained. That is, there exists a worst-case measure that contributes to the optimal value.

Finally, the lower semicontinuity of α plays an important role: it ensures the stability of the optimization problem. Due to the lower semicontinuity of α, the supremum might be infinite, which makes optimization unreliable numerically, such as in neural network training.

4.5.4.2 Fatou and Lebesgue Properties

Further, the Fatou property of the risk measure ρ helps in establishing convergence. We state the property as in Föllmer and Schied (2004).

Lemma 4.5.17 (Fatou Property of Convex Risk Measures) *Suppose $\rho : \mathcal{X} \to \mathbb{R}$ is a convex risk measure. Then, for any bounded sequence (X_n) in $\mathcal{X}$ such that $X_n \to X$ pointwise a.s., we have*

$$\rho(X) \leq \liminf_{n \uparrow \infty} \rho(X_n).$$

The Fatou property of ρ is a form of lower semicontinuity that ensures that the risk of the limit, $\rho(X)$, is not worse than the limit of the risks $\rho(X_n)$. This property plays a role in explaining the effectiveness of the deep hedging framework.

The implications of the Fatou property are strengthened by those of the Lebesque property, explained in the following result from Föllmer and Schied (2004).

Lemma 4.5.18 (Lebesgue property of convex risk measures) *Let $\rho : \mathcal{X} \to \mathbb{R}$ be a convex risk measure. Then, if (X_n) is a bounded sequence in $\mathcal{X}$ such that $X_n \to X$ pointwise, we have*

$$\rho(X_n) \to \rho(X).$$

The Lebesgue property is a form of full continuity, that is, both upper and lower semicontinuity, which we have due to our assumptions that Ω is finite and ρ is defined on a finite set.

Chapter 5

The Deep Hedging Framework

5.1 The Evolution of Hedging: From Theory to Data-Driven Approaches

At first glance, Chapter 4 seems to have given us everything required for hedging. If markets behaved as neatly as the classical finance approaches suggest, nothing else would be needed. However, reality rarely presents us with the perfect conditions. The assumptions dictating the soundness of traditional hedging models do not always hold in practice.

In a complete market, every contingent claim can be perfectly hedged, so choosing the right portfolio can eliminate any risk. This idea was at the heart of classical finance in Section 4.4, where market completeness was discussed as a key assumption for the Black-Scholes model. Indeed, numerous popular pricing models rely on the ability to trade without frictions continuously and replicate any derivative. As Buehler et al. (2019) point out, the pricing and risk of portfolio derivatives are linear under complete market assumptions. That is, if we hedge two contingent claims separately, the total hedge is equal to the sum of the two hedges for separate claims. However, pricing and hedging are never linear and depend on interactions between positions. Further, trading comes with costs, liquidity is not always guaranteed, and trades can move prices, introducing market impact. Moreover, firms often have internal risk limits that restrict how much risk a trader or the entire institution can take. These frictions make perfect replication impossible and add layers of complexity that classical models tend to ignore. Despite this, complete market models remain commonly used due to a lack of efficient alternatives, with deep hedging being one of the solutions.

We have also discussed risk in detail in Section 4.5. In particular, we discussed market risk, which stems from unpredictable fluctuations in market prices; however, risk is not only related to market movements. There is another, often underestimated, kind - model risk. As Cont (2006) defines it, model risk comes from the choice of pricing model; it is present when models give significantly varying valuations for the same derivatives. This can lead to serious mispricing, which can, in turn, lead to wrong trading decisions and risk measurements and may sometimes result in substantial financial losses. Despite this being a significant problem, classical methods of handling model risk, such as worst-case scenario analysis or stress testing, do not distinguish between market risk and model risk. While Cont (2006) suggests several ways to quantify model risk, in particular, by adapting coherent and convex risk measures, these methods still require choosing a model in the first place. Deep hedging seems to be a natural solution as the framework is entirely model-free. Instead of relying on a fixed set of assumptions about the behavior of markets, it constantly updates its strategies based on real data.

Deep hedging is a compelling solution to these problems in traditional models. The framework does not assume idealised market conditions or require committing to a model that might later prove inaccurate. Instead, it learns directly from data and adapts dynamically, making it far

more robust in the chaotic world of real markets. Hence, deep hedging can be considered a step beyond the classical frameworks towards a more applicable, data-oriented approach for today's complex financial markets.

5.1.1 How Data-Driven Hedging Took Shape

In the second half of the 20th century, both mathematical finance and machine learning underwent a speedy evolution into the modern approaches we know today. As the deficiencies of traditional pricing models became evident, especially in light of market events and empirical inconsistencies, the idea of applying data-driven approaches to pricing developed as early as the 1990s.

Hutchinson et al. (1994) were among the first to propose using artificial neural networks (ANNs) to estimate option prices in their paper *A Nonparametric Approach to Pricing and Hedging Derivative Securities via Learning Networks*. Observing the issues presented by model misspecifications, market frictions, and lack of analytical solutions for specific scenarios, the authors proposed three machine learning solutions for pricing: radial basis functions (RBF), which use Gaussian-like functions to approximate smooth functions, project pursuit regression (PPR), which model the function as a sum of nonlinear functions of projections, and multilayer perceptrons (MLP), which are standard feed-forward neural networks with one hidden layer, as was defined in Chapter 3. Comparing the performance of these models in pricing and hedging against simulated Black-Scholes option prices, Hutchinson et al. (1994) found that data-based approaches price options with near-perfect accuracy and significantly outperform the Black-Scholes model while hedging in an incomplete real-world market. While the frameworks we have today are more advanced, the authors established that nonparametric machine learning approaches can price derivatives accurately and outperform traditional models in real markets, paving the way for modern deep hedging methods.

Since then, further developments have continued to push the boundaries of data-driven finance further. For instance, volatility estimates, neural network architectures consistent with financial constraints such as no-arbitrage conditions, and other improvements, which can be found in Ruf and Wang (2019), have been proposed to develop the framework further. Theoretical and practical improvements in machine learning and an increasing understanding of market incompleteness and model risk have led to a new generation of flexible data-based approaches. These ideas are the foundation of deep hedging, which integrates decades of research in finance and machine learning to develop a powerful and more realistic framework for setting prices in complex markets.

5.2 Market Setting and Hedging Problem

After establishing a motivation for working in a realistic market setting, we now describe the formal hedging problem following the framework introduced by Buehler et al. (2019). This approach captures the essential components of real-world trading, including transaction costs, market frictions and decision-making as a function of information, while remaining compatible with numerical methods and learning-based strategies. We aim to create a model that replicates the constraints of the trader and develop a hedging problem that can be addressed without the assumptions of a complete market or arbitrage-free pricing.

5.2.1 Market Framework: Assets, Information, and Timing

While in the real world, we are faced with endless outcomes and possibilities, here we fix a *finite* probability space $\Omega = \{\omega_1, \ldots, \omega_N\}$. Such a probability space can still be rich enough to model realistic scenarios while simplifying the analysis significantly. Indeed, if the space is finite, all random variables and stochastic processes we work with are effectively functions on a finite set.

Further, fix a probability measure for which all outcomes happen with a positive probability, that is, $\forall i = 1, ..., N$, $\mathbb{P}(\{\omega_i\}) > 0$.

In a real-world market, it is infeasible to record prices or adjust a trading strategy continuously. Traders adjust their portfolios at discrete time intervals, for example, every minute, hour or day, based on new information they receive. Continuous-time models rely heavily on stochastic calculus and martingale theory and, while they are elegant, such models are less intuitive and less suitable for numerical or machine learning methods. Therefore, to build the setting for deep hedging, consider a discrete-time financial market, which means that all trading and information updates happen only at n fixed time points. We call such time points trading times $0 = t_0 < t_1 < \ldots < t_n = T$.

To adjust a trading strategy based on new information a trader receives, we want to be able to represent this information in a manageable way. Suppose that at each time step t_k, an agent observes a vector of information pertaining to the market, $I_k \in \mathbb{R}^r$. This could be market costs, mid-prices [1] of liquid instruments, risk restrictions, etc. Then, the process $I = (I_k)_{k=0}^n$ represents how information accumulates over time. We then define the natural filtration $(\mathcal{F}_k)_{k=0}^n$, given by $\mathcal{F}_k = \sigma(I_0, \ldots, I_k)$, i.e. the σ-algebra by all relevant information generated in the market by time t_k. This is the information on which the trader will make their decision at that timestep. For example, if a timestep represents a day, $\mathcal{F}_1$ might be everything known after one trading day: the asset prices at t_0 and t_1, trading volume, news signals, and so on. Any decision made at time t_1 can depend on all of this, but not any information from t_2 or later.

Our market consists of d traded assets - hedging instruments, such as stocks. We denote this by a stochastic process $S = (S_k)_{k=0}^n$ that takes values in $\mathbb{R}^d$. The process is adapted to the natural filtration, that is, each S_k if $\mathcal{F}_k$-measurable, meaning that we can determine the price of S_k at time t_k based on the information available up to that point.

Aside from a finite probability space, we make several other simplifications. Firstly, we assume that the risk-free interest rates are zero, that is $r = 0$. Secondly, all cash flows, such as option payoffs, occur at time T. These assumptions allow us to focus on the framework guiding pricing and hedging without distractions, such as discounting or settlements at intermediate trading times.

One important modeling assumption that we make is that the market is not arbitrage-free. This might appear counterintuitive at first, especially if one is used to classical financial theory, where the absence of arbitrage is assumed, but this is quite understandable in the present setting. In reality, there are costs of trading, lack of liquidity, model uncertainty, and other constraints that may violate the no-arbitrage conditions. Furthermore, we do not aim to determine the theoretical value of the asset but to hedge a portfolio in a possibly imperfect market. Hence, rather than excluding arbitrage, we proceed with the necessary methods to tackle the risk.

5.2.2 Formulating the Hedging Problem

Now, let us introduce the hedging problem itself. We consider a portfolio of derivatives whose total liability, i.e. the amount the agent will owe, at time T is described by a random variable $Z \in \mathcal{F}_T$ [2]. We refer to Z as the contingent claim.

The agent can hedge the liability Z by trading in assets S using a hedging strategy $\delta = (\delta_k)_{k=0}^{n-1}$, where each δ_k is a vector in $\mathbb{R}^d$ corresponding to how much of each asset in $S \in \mathbb{R}^d$ the trader holds at time t_k. The trading strategy process is adapted to the natural filtration, that is, $\delta_k \in \mathcal{F}_k$ for each k, and so δ_k is a function of all the information available by time t_k. We denote the collection of all such strategies by $\mathcal{H}$.

In order to measure how successful a trader is at hedging, we need a way to measure the trader's

[1] The midpoint of a bid-ask spread.

[2] The notation $Z \in \mathcal{F}_T$ represents that the random variable Z is $\mathcal{F}_T$-measurable. This means that for each outcome $\omega \in \Omega$, the payout of $Z(\omega)$ is known, so the value of Z depends on the information revealed at time T, $\mathcal{F}_T$.

wealth. At each timestep, the value of assets changes from S_k to S_{k+1}. If a trader uses strategy δ_k at that time step, their portfolio value increases or decreases based on the amount of each asset S_k^i the agent holds, δ_k^i. Then, the total profit (or loss) at the terminal time T can be described by the sum of all such changes. We define the profit and loss as

$$(\delta \cdot S)_T = \sum_{k=0}^{n-1} \delta_k \cdot (S_{k+1} - S_k).$$

So, after an initial injection of cash of value p_0 in order to make the trading self-financed, we define the agent's wealth at time T as

$$-Z + p_0 + (\delta \cdot S)_T.$$

This represents the cumulative gain or loss from trading with $\delta \in \mathcal{H}$.

As discussed previously, trading is not always free in incomplete markets. After time t_k, the agent may want to adjust their trading strategy from δ_{k-1} to δ_k, which, in turn, incurs a cost of $c_k|\delta_k - \delta_{k-1}|$. Then, the total cost of trading a strategy δ in all timesteps is

$$C_T(\delta) = \sum_{k=0}^{n} c_k|\delta_k - \delta_{k-1}|.$$

We assume that the cost functions c_k are upper semicontinuous. Semicontinuity plays an important role in ensuring that the hedging problem is well-posed, and will be discussed throughout this chapter.

The agent's terminal portfolio value is then given by

$$PL_T(Z, p_0, \delta) = -Z + p_0 + (\delta \cdot S)_T - C_T(\delta). \tag{5.1}$$

Hence, the trader starts with initial capital p_0, trades according to a strategy $\delta \in \mathcal{H}$ and ends up with a terminal wealth $PL_T(Z, p_0, \delta)$. The main objective of the trader is to ensure that the terminal wealth offsets the liability Z and minimize the risk of losses.

5.3 Pricing Through Risk: Hedging Without Replication

If our setting were a complete market, we would have the guarantee of a perfect hedge. That is, there would exist a trading strategy δ that replicates the contingent claim $X \in \mathcal{X}$, where $\mathcal{X}$ is the space of all integrable random variables on Ω as in Subsection 4.5.1, meaning the agent's terminal wealth, as defined in (5.1), would be zero.

By the Second Fundamental Theorem of Asset Pricing 4.3.6, such a market would have exactly one risk-neutral probability measure (see Definition 4.3.3). In that case, the price of a contingent claim would be given by its expected value under the risk-neutral probability measure. This pricing rule is elegant and powerful; however, real-world markets are rarely complete and frictionless. Therefore, we cannot hedge perfectly, and thus, we need to find another way to quantify the success of a hedge.

To accomplish this, we revisit the notion of convex risk measures, given in definition 4.5.4. Recall that a convex risk measure can be viewed as a capital requirement - the smallest amount of money we can add to the position X to make the position acceptable. This interpretation implies a natural strategy in incomplete markets: instead of trying to replicate a contingent claim perfectly, we could search for a trading strategy that minimizes the capital requirement for the resulting position. Since we may not be able to fully cancel out the liability through trading alone, we accept that some residual risk remains, and we aim to control that risk rather than eliminate it, using the risk measure ρ. In this way, hedging becomes a problem of risk minimization, not replication.

We let $\rho : \mathcal{X} \to \mathbb{R}$ be a convex risk measure as in definition 4.5.4. Then, given a liability $X \in \mathcal{X}$, we define the optimization problem as in Buehler et al. (2019):

$$\pi(X) = \inf_{\delta \in \mathcal{H}} \rho(X + (\delta \cdot S)_T - C_T(\delta)). \tag{5.2}$$

This defines a pricing functional $\pi : \mathcal{X} \to \mathbb{R}$, which evaluates the smallest risk-adjusted capital required to hedge the liability X. The optimal strategy $\delta \in \mathcal{H}$ is any minimizer of (5.2). We also note the following result from Buehler et al. (2019).

Proposition 5.3.1 (Properties of the Hedging Functional) *The pricing functional π, as given by* (5.2)*, is monotone, decreasing, and cash-invariant.*

Further, suppose C_T and $\mathcal{H}$ are convex, that is, C_T satisfies the convexity axiom (4.11) *and $\forall \delta_1, \delta_2 \in \mathcal{H}$ and $\lambda \in [0, 1]$, $\lambda\delta_1 + (1-\lambda)\delta_2 \in \mathcal{H}$. Then, the functional π is a convex risk measure.*

Clearly, monotonicity and cash-invariance are inherited from the properties of ρ. Convexity of π follows because, if both the trading costs and the set of strategies are convex, then convex combinations of strategies leads to a valid strategy with no worse (and often better) overall risk, and the convexity of ρ ensures that the risk of the mix does not exceed the sum of individual risks.

The fact that π is a convex risk measure ensures that the hedging problem is stable and solvable, both mathematically and computationally, as well as that it reflects realistic decision-making under uncertainty.

5.3.1 Indifference Pricing

In classical theory, pricing is all about replication - if a contingent claim can be replicated, then its price is the cost of the replicating strategy. However, in the presence of market frictions, we look for risk-aware ways to price claims. We define the following as in Buehler et al. (2019).

Definition 5.3.2 (Indifference Price) *The indifference price is given by*

$$p(Z) = \pi(-Z) - \pi(0).$$

The indifference price is derived from the convex risk measure used to formulate the hedging problem (5.2). This represents the smallest amount of cash the agent would require in order to accept the liability $Z \in \mathcal{X}$, such that their risk as measured by π does not change. The trader is then *indifferent* between keeping their current portfolio with no additional liabilities and taking on the liability $-Z$ with the compensation of $p(Z)$.

This approach poses the optimization problem by describing how to minimize the agent's uncertainty about what will happen, given the available information and decision-making costs. It also describes the agent's individual attitude toward risk. The indifference price differs from the traditional replication price because it does not use the equivalent martingale measure and reflects the cost and difficulty of hedging in imperfect markets.

To build some intuition, we consider the scenario with no transaction costs and perfect replication. The following result is from Buehler et al. (2019).

Lemma 5.3.3 (Indifference Price in the Case of Perfect Replication) *Suppose that there are no transaction costs, that is, $C_T = 0$. If a claim Z is attainable, i.e. there exists $\delta^* \in \mathcal{H}$ and $p_0 \in \mathbb{R}$ such that*

$$Z = p_0 + (\delta^* \cdot S)_T,$$

then the indifference price is $p(Z) = p_0$.

Proof of Lemma 5.3.3. *Taking $C_T = 0$ and substituting the replicating strategy for Z in* (5.2)*, we have*

$$\begin{aligned}
\pi(-Z) &= \inf_{\delta\in\mathcal{H}} \rho(-Z + (\delta\cdot S)_T) \\
&= \inf_{\delta\in\mathcal{H}} \rho(-p_0 - (\delta^*\cdot S)_T + (\delta\cdot S)_T) \\
&= \inf_{\delta\in\mathcal{H}} \rho(-p_0 + ((\delta-\delta^*)\cdot S)_T) \\
&= \inf_{\delta\in\mathcal{H}} \rho((\delta-\delta^*)\cdot S)_T + p_0 && \text{by cash invariance (4.7)} \\
&= p_0 + \inf_{\delta\in\mathcal{H}} \rho((\delta'\cdot S)_T) && \text{substituting } \delta' := \delta-\delta^* \in \mathcal{H} \\
&= p_0 + \pi(0).
\end{aligned}$$

□

This tells us that in the idealized case of a complete market, in which replication is possible and trading does not incur costs, the indifference price reduces to the initial cash injection required to satisfy the self-financing condition - exactly as classical pricing theory would suggest. Lemma 5.3.3 suggests that the indifference price extends the replication price to scenarios in which replication is impossible.

Consider the risk measure in the form

$$\rho(X) = \inf_{w\in\mathbb{R}} \{w + \mathbb{E}[\ell(-X-w)]\} \tag{5.3}$$

for $X \in \mathcal{X}$, where $\ell : \mathbb{R} \to \mathbb{R}$ is a continuous, non-decreasing, and convex *loss function*. Note that in Definition 4.5.9 of the Optimized Certainty Equivalent, the risk measure is expressed as in (5.3) with the loss function $\ell(x) = -u(-x)$ for $x \in \mathbb{R}$ and a utility function $u : \mathbb{R} \to \mathbb{R}$. To understand the indifference price better, we also look at the result from Buehler et al. (2019) with such a risk measure representation.

Proposition 5.3.4 (Properties of the Indifference Price) *Suppose that S is a $\mathbb{P}$-martingale and that the risk measure $\rho(X)$, the pricing functional $\pi(X)$ and the indifference price $p(Z)$ are given as in* (5.3)*,* (5.2) *and Definition 5.3.2, respectively. Then, we have*

(i) $\pi(0) = \rho(0)$*,*

(ii) $p(Z) \geq \mathbb{E}[Z]$ *for all* $Z \in \mathcal{X}$*.*

Proof of Lemma 5.3.4.

(i) Note that as $\mathcal{H}$ is a vector space, $0 \in \mathcal{H}$. Additionally, it does not cost anything to not take on a liability, that is $C_T(0) = 0$. Plugging in the zero strategy gives $\pi(0) \leq \rho(0)$.

For the reverse inequality, note that since S is a martingale, we have $\mathbb{E}[(\delta\cdot S)_T] = 0$ for all $\delta \in \mathcal{H}$, and because $C_T(\delta) \geq 0$, then $\mathbb{E}[-(\delta\cdot S)_T + C_T(\delta)] \geq 0$. Then, by Jensen's inequality and convexity of ℓ, we have

$$\begin{aligned}
\pi(0) &= \inf_{\delta\in\mathcal{H}} \rho((\delta\cdot S)_T - C_T(\delta)) \\
&= \inf_{\delta\in\mathcal{H}} \inf_{w\in\mathbb{R}} \{w + \mathbb{E}[\ell(-(\delta\cdot S)_T + C_T(\delta) - w)]\} \\
&\geq \inf_{\delta\in\mathcal{H}} \inf_{w\in\mathbb{R}} \{w + \ell(\mathbb{E}[-(\delta\cdot S)_T + C_T(\delta)] - w)\} \\
&\geq \inf_{\delta\in\mathcal{H}} \inf_{w\in\mathbb{R}} \{w + \ell(-w)\} \geq \inf_{w\in\mathbb{R}} \{w + \mathbb{E}[\ell(-w)]\} = \rho(0)
\end{aligned}$$

(ii)

$$\begin{aligned}
\pi(-Z) &= \inf_{\delta\in\mathcal{H}} \rho(-Z + (\delta\cdot S)_T - C_T(\delta)) \\
&= \inf_{\delta\in\mathcal{H}} \inf_{w\in\mathbb{R}} \{w + \mathbb{E}[\ell(Z - (\delta\cdot S)_T + C_T(\delta) - w)]\} \\
&\geq \inf_{w\in\mathbb{R}} \{w + \ell(\mathbb{E}[Z] - w)\} && \textit{by Jensen's inequality \& convexity of } \ell \\
&= \rho(-\mathbb{E}[Z]) = \mathbb{E}[Z] + \rho(0) && \textit{by cash invariance } (4.7) \\
&= \mathbb{E}[Z] + \pi(0) && \textit{by (i)}
\end{aligned}$$

□

The first result simply tells us that, if there is no claim to hedge, $Z = 0$, then the functional gives $\rho(0)$, the trader's baseline level of risk. The second result provides more insights as it reflects the agent's risk aversion: the indifference price is always no less than the expectation of the claim. In a frictionless market, the trader would accept the expected value, however, in the presence of constraints and hedging costs, there is always uncertainty that remains, hence, the agent requires compensation for the risk.

Example 5.3.5 (Risk premium) *Consider something called a binary option. Such an option pays £1 if a certain event occurs at time T and nothing otherwise. Call the event A and suppose that $\mathbb{P}(A) = 0.5$, so the expected value of the claim is*

$$\mathbb{E}[Z] = 0.5.$$

In a complete and frictionless market, this would be the price of our binary option. However, the trader is risk-averse and cannot replicate the payoff. Suppose they use a risk measure to evaluate risk and solving the optimization problem, they find $\pi(-Z) - 0.7, \quad \pi(0) = 0.1$. Then, the indifference price is

$$p(Z) = \pi(-Z) - \pi(0) = 0.7 - 0.1 = 0.6.$$

This means that the agent would only sell the claim if they are offered at least £0.6, not just the expected value £0.5. The additional £0.1 is the risk premium - a value that reflects the risk the agent takes in an incomplete market, according to their risk preferences.

Indifference pricing gives us a flexible and risk-aware way of pricing contingent claims in an incomplete market. It keeps many of the desirable features of classical pricing, and many of them extend to more realistic market settings.

5.4 Neural Network Approximation of Hedging Strategies

We now reformulate the risk minimization problem in (5.2) as a problem of optimizing over neural networks as in Buehler et al. (2019). Revisiting Definition 3.2.2, we consider feed-forward neural networks $F : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$. Recall that in Theorem 3.3.1, we defined $\mathcal{NN}_{\infty,d_0,d_1}$ as the set of all such neural networks F.

Consider a sequence $\{\mathcal{NN}_{M,d_0,d_1}\}_{M\in\mathbb{N}}$, where M denotes the architectural complexity of rthe neural networks in this set - it is a bound for the number of layers, neurons, total parameters, and so on. Such a sequence is always a subset of the universal class, so we have $\mathcal{NN}_{M,d_0,d_1} \subset \mathcal{NN}_{\infty,d_0,d_1}$ for every complexity. For small M, the networks are simpler and can only approximate simple functions. However, as M gets bigger, the networks become more expressive. Each network in the class is defined by a vector of parameters (such as weights and biases) θ, the number of which is also bounded by M, that is, $\mathcal{NN}_{M,d_0,d_1} = \{F^\theta : \theta \in \Theta_{M,d_0,d_1}\}$, where $\Theta_{M,d_0,d_1} \in \mathbb{R}^q$ for some $q \in \mathbb{N}$. If we consider the sets of networks of all complexities together, we get the full universal approximator: $\bigcup_{M\in\mathbb{N}} \mathcal{NN}_{M,d_0,d_1} = \mathcal{NN}_{\infty,d_0,d_1}$.

When setting up the pricing functional given by (5.2), we defined a minimizer of π, $\delta \in \mathcal{H}$, as the optimal hedging strategy and, therefore, the solution to the optimization problem. As the optimal hedging strategy is the desired solution, we now redefine the trading position at time step k as the output of the neural network with parameters θ, $\delta_k^\theta := F_k^\theta(I_0, ..., I_k, \delta_{k-1})$. Hence, the position held by the agent at time t_k depends on all data observed up to t_k and the previous hedging strategy. We also note that the neural network which determines the hedging strategy is in the class of neural networks $F_k^\theta \in \mathcal{NN}_{M,r(k+1)+d,d}$. We have $I_j \in \mathbb{R}^r$ for $j = 0, ..., k$ and $\delta_{k-1} \in \mathbb{R}^d$, resulting in the input dimension $r(k+1)+d$. The output dimension d is that of the new hedging strategy.

We introduce a function $H : \mathbb{R}^d \to \mathbb{R}^d$, which adjusts the raw output of the strategy δ_k into a trading action which accounts for the present trading constraints, such as trading limits. The notation $H \circ \delta$ indicates that the function is applied pointwise.

In Buehler et al. (2019), the neural network that allows to solve the optimization problem is then given by the finite-dimensional subset of the trading strategies $\mathcal{H}$,

$$\begin{aligned}\mathcal{H}_M &= \{(\delta_k)_{k=0}^{n-1} \in \mathcal{H} : \delta_k = F_k(I_0, ..., I_k, \delta_{k-1}), F_k \in \mathcal{NN}_{M,r(k+1)+d,d}\} \\ &= \{(\delta_k^\theta)_{k=0}^{n-1} \in \mathcal{H} : \delta_k^\theta = F^{\theta_k}(I_0, ..., I_k, \delta_{k-1}), \theta_k \in \Theta_{M,r(k+1)+d,d}\},\end{aligned} \tag{5.4}$$

and the neural network approximation of the hedging price using strategies in $\mathcal{H}^M$,

$$\begin{aligned}\pi^M(X) &= \inf_{\delta \in \mathcal{H}_M} \rho(X + (H \circ \delta \cdot S)_T - C_T(H \circ \delta)) \\ &= \inf_{\theta \in \Theta_M} \rho(X + (H \circ \delta^\theta \cdot S)_T - C_T(H \circ \delta^\theta)),\end{aligned} \tag{5.5}$$

where $\Theta_M = \prod_{k=0}^{n-1} \Theta_{M,r(k+1)+d,d}$ is the total parameter space for all timesteps. Reformulating the problem as in (5.4) and (5.5) changes the infinite-dimensional optimization problem into a finite-dimensional one, facilitating training neural networks.

5.5 Why Deep Hedging Works: A Theoretical Perspective

The deep hedging framework has significantly impacted pricing and hedging due to practical success. Its effectiveness can be explained by considering semicontinuity and exploring the convergence of convex risk measures.

Let us revisit semicontinuity given in Definition 4.5.14. The following result from Föllmer and Schied (2004) establishes a condition ensuring that a limit cannot "escape to infinity".

Lemma 5.5.1 *A convex and lower semicontinuous function $h : \mathbb{R}^d \to \mathbb{R}$ with a finite value at zero, $h(0) < \infty$, attains its minimum if, for all $v \in \mathbb{R}^d$ with $v \neq 0$ we have*

$$\lim_{\alpha \to \infty} h(\alpha v) = \infty.$$

This result is important because under certain integrability conditions, e.g. when defined on L^p for $p \geq 1$, risk measures are lower semicontinuous. Recall that we defined the risk measures appearing in the optimization problem on $\mathcal{X}$, the space of all integrable functions on Ω. Hence, the risk measures used in deep hedging satisfy integrability conditions and are lower semicontinuous. Moreover, we have seen in Proposition 5.3.1 that the pricing functional π, given by (5.2), itself is a convex risk measure defined on $\mathcal{X}$, meaning that π is lower semicontinuous.

In the context of our optimization problem given by (5.4) and (5.5), if π is lower semicontinuous, that is, satisfies the conditions of Proposition 5.3.1, Lemma 5.5.1 ensures that the infimum in the optimization problem is achieved. This, in turn, means that an optimal hedging strategy indeed exists, so the hedging problem is well-posed and has solutions.

Buehler et al. (2019) establish several results verifying the soundness of this framework. We restate a proposition detailing how neural networks approximate the optimal price.

Proposition 5.5.2 (Convergence of the Pricing Functional) *Define $\mathcal{H}_M$ and π^M as in Equations 5.4 and 5.5, respectively. Then, for any $X \in \mathcal{X}$,*

$$\lim_{M\to\infty} \pi^M(X) = \pi(X).$$

Proposition 5.5.3 (Convergence of the pricing functional) *Define $\mathcal{H}_M$ and π^M as in Equations 5.4 and 5.5, respectively. Then, for any $X \in \mathcal{X}$,*

$$\lim_{M\to\infty} \pi^M(X) = \pi(X).$$

Proof of Proposition 5.5.2. *We already know that $\pi^M(X) \geq \pi(X)$ by construction as $\pi(X)$ is the infimum over all strategies, while $\pi^M(X)$ is the infimum over a restricted set of strategies. Hence, it is enough to show that*

$$\limsup_{M\to\infty} \pi^M(X) \leq \pi(X).$$

That is, we show that for any $\varepsilon > 0$, there exists a network that is big enough and gives us a price that is not worse than $\pi(X) + \varepsilon$.

Pick a strategy $\delta \in \mathcal{H}$ that is close to optimal, that is

$$\rho(X + (H \circ \delta \cdot S)_T - C_T(H \circ \delta)) \leq \pi(X) + \frac{\varepsilon}{2}.$$

Since δ_k depends on information $(I_0, ..., I_k)$, there exists a function f_k such that $\delta_k = f_k(I_0, ..., I_k, \delta_{k-1})$. Then, each component $f_k^i \in L^1(\mathbb{P})$, so we can approximate each f_k^i by a neural network. By the Universal Approximation Theorem 3.3.1, we can find neural networks $F_{k,n}^i$ that approximate $f_k^i \in L^1(\mathbb{P})$ as $n \to \infty$:

$$F_{k,n}^i(I_0, ..., I_k) \to f_k^i(I_0, ..., I_k) \quad \text{as } n \to \infty.$$

As Ω is assumed to be finite, convergence in L^1 implies pointwise convergence on Ω. We therefore have, as $n \to \infty$,

$$\delta_k^n(\omega) := F_{k,n}(I_0, ..., I_k)(\omega) \to \delta_k(\omega) \quad \text{for all } \omega \in \Omega.$$

Now, recall that ρ is convex and continuous because Ω is finite and that we assumed that C_T is upper semicontinuous. Then, we get

$$\begin{aligned}\liminf_{n\to\infty} \rho(X + (H \circ \delta^n \cdot S)_T - C_T(H \circ \delta^n)) &\leq \rho(X + (H \circ \delta \cdot S)_T - \limsup_{n\to\infty} C_T(H \circ \delta^n)) \\ &\leq \rho(X + (H \circ \delta \cdot S)_T - C_T(H \circ \delta^n)).\end{aligned}$$

Hence, for large enough n, we have

$$\rho(X + (H \circ \delta^n \cdot S)_T - C_T(H \circ \delta^n)) \leq \pi(X) + \varepsilon.$$

Since, if the complexity M is large enough, we have $\delta^n \in \mathcal{H}_M$, we conclude that

$$\pi^M(X) \leq \pi(X) + \varepsilon \quad \Longrightarrow \quad \limsup_{M\to\infty} \pi^M(X) \leq \pi(X).$$

□

Therefore, Proposition 5.5.2 concludes that as the complexity of the networks increases, the neural network approximation $\pi^M(X)$ converges to the true indifference price.

Another fundamental result shows that the risk measure converges as the complexity increases. We do not state this result in full detail (see Theorem 4.13 in Buehler et al. (2019)); however, the authors prove that under mild conditions on the finiteness and continuity of the penalty

function α in the robust representation 4.5.12, the risk measure $\rho(X)$ can be approximated by neural network functionals, that is

$$\rho(X) = \lim_{M \to \infty} \rho^M(X). \tag{5.6}$$

The optimization for each $\rho^M(X)$ occurs within neural networks. The optimization process now focuses on neural networks with bounded complexity, which enables computational tractability while the true risk measure is approximated by increasing network dimensions.

Proposition 5.5.2 and the result given by (5.6) demonstrate a strong theoretical foundation for the deep hedging framework. The convergence of the pricing functional approximated by the neural networks shows that we can approximate hedging strategies reliably, using neural networks of increasing complexity. Semicontinuity, in particular, the semicontinuity of the risk measure ρ and the trading costs C_T, plays an important role in ensuring this convergence.

The upper semicontinuity of C_T guarantees that small changes in the trading strategy do not cause sudden increases in the trading costs. This is the key factor allowing us to conclude the convergence of the pricing functional from the convergence of hedging strategies in the proof of Proposition 5.5.3.

Further, the Fatou and Lebesgue properties given by Lemma 4.5.17 and Lemma 4.5.18, respectively, of the risk measure ρ help in establishing convergence of the optimization problem. While the Fatou property indicates that the risk values will remain stable, the Lebesgue property establishes that the risk will actually converge exactly as the strategies converge. When our neural networks learn better strategies, in the sense that their outputs get closer pointwise to the true optimal strategy, the Lebesgue property ensures that the risk values we compute using these approximations also get closer to the true risk. This goes beyond the Fatou property, which only guarantees that the risk does not get worse in the limit. With the Lebesgue property, we get full convergence: the risk of the approximated strategies becomes arbitrarily close to the risk of the optimal one. Because we assume that the probability space Ω is finite, this property holds automatically, making the convergence results in Proposition 5.5.2 and (5.6) much stronger. It confirms that as we increase the size of the neural networks, we are truly learning strategies whose performance matches the theoretical optimum, not just in strategy space but also in terms of actual risk.

Together with the properties discussed in 4.5.4.1, the semicontinuity and convergence properties above demonstrate that neural network-based hedging strategies work well in practice and are theoretically well-founded. They guarantee that as we train our networks, the solutions remain stable and that the learned strategies converge to the true optimal ones. This gives confidence that we are not just fitting something that happens to work but approximating a well-defined solution to a carefully posed mathematical problem. These properties are what give deep hedging its theoretical foundation.

Chapter 6

Practical Deep Hedging

The deep hedging framework has had a significant impact on the field of quantitative finance by introducing a powerful way to use machine learning to create hedging strategies. This framework is exciting because it handles market frictions and allows us to learn directly from data using a model-free approach, and, as we have seen in Chapter 5, it is theoretically sound.

Many have found that deep hedging can outperform traditional hedging models, especially in scenarios where risk is difficult to manage. In this chapter, we explore how the framework performs in pactice, seeking to understand how the model works and grasp when it works best.

6.1 Numerical Setup and Model Construction

We now build the foundation of our deep hedging implementation, which we later use to explore the model's behavior under various market conditions and risk measures. The setup is built in Python with machine learning libraries such as TensorFlow, showing how deep hedging blends ideas from computational finance and data-driven modeling.

As introduced in Section 5.4, we model the hedging strategy as a sequence of functions $\delta^\theta = (\delta_0^\theta, \dots, \delta_{n-1}^\theta)$, where each δ_k^θ is represented by a neural network. Each of these networks learns to predict the hedge position at a given time step based on the current state of the market. Before we go into details of how the model is trained, we build this sequence of networks.

6.1.1 The List of Neural Networks

The first function builds a list of neural networks, where the aim of each network is to learn the hedge at time t_k.

Build a list of neural networks for each timestep

```
import tensorflow as tf

def create_networks(d, hidden_nodes, L, n):
    # Initialize an empty list where the networks will be stored
    Hedging_Networks = []

    for j in range(n):                          # For each time step
        inputs = tf.keras.Input(shape = (d,))   # Define the input layer
        x = inputs
```

```
        for i in range(L):                          # For each layer
            if i < L-1:                             # Define hidden layers
                layer = tf.keras.layers.Dense(
                    hidden_nodes,
                    activation = 'tanh',
                    trainable = True,
                    # Initial weights and biases
                    kernel_initializer =
                    ↪  tf.keras.initializers.RandomNormal(0,1),
                    bias_initializer = 'random_normal',
                    name = str(j)+'step'+str(i)+'layer'
                )
                x = layer(x)
            else :                                  # Define output layer
                layer = tf.keras.layers.Dense(
                    d,
                    activation = 'linear',
                    trainable = True,
                    kernel_initializer =
                    ↪  tf.keras.initializers.RandomNormal(0,0.1),
                    bias_initializer = 'random_normal',
                    name = str(j)+'step'+str(i)+'layer'
                )
                outputs = layer(x)
                network = tf.keras.Model(inputs=inputs, outputs=outputs)
                Hedging_Networks.append(network) # Add the NN to the list
    return Hedging_Networks
```

The function `create_networks` returns a list of n networks, each corresponding to one of the n trading times $t_0, \ldots, t_{n-1}$. Each network takes an input of dimension d and maps it to a hedge position δ_k^θ. Each network consists of L fully connected (dense) layers.

A fully conneceted, or dense, layer is a type of neural network layer where every input node is connected to every output node. This allows for all input features to interact with each other. In financial modeling, this is a natural choice because relationships between variables are often global. For example, a change in one asset price may affect the optimal hedge for multiple hedging instruments.

Each hidden layer has `hidden_nodes` number of neurons and a hyperbolic tangent activation function, $\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, illustrated in Figure 6.1. This activation function fits the inputs into the range $(-1, 1)$, so positive inputs take values near 1, while negative inputs take values near -1. Notice that $\tanh(0) = 0$. A zero-centered activation function eases weight updates, while its smoothness ensures training stability.

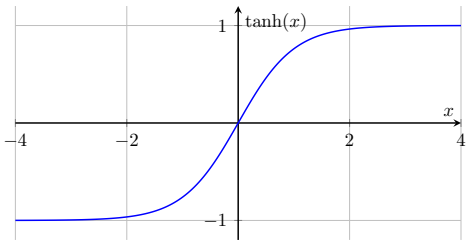


Figure 6.1: The hyperbolic tangent activation function $\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

The output at each step is $\delta_k^\theta \in \mathbb{R}^d$, each entry in the vector representing how many units of

the asset we hold at time t_k. The hedging strategy can be positive for a long position, negative for a short position, or zero for not hedging at all. Hence, it can be any real number without any constraints. This is why we use a linear activation function in the output layer. The transformation with weights and biases is applied to the raw output, keeping δ_k^θ unrestricted.

Lastly, we use a smaller variance when initializing the weights in the output layer compared to the hidden layers. This helps keep the hedge predictions small and stable at the very start of training, which is important for avoiding large, random losses early on. In contrast, the hidden layers use a higher weight variance so that their outputs are more diverse. This gives the network more room to explore different patterns in the data and learn useful behaviors as training progresses.

6.1.2 The Deep Hedging Model

The `DeepHedgingLayer` is a class that describes what a deep hedging layer should do, like taking price data, computing hedge positions with networks, and returning the final portfolio value. Our `DeepHedgingLayer` class defines a custom Keras layer that puts the full deep hedging model together. The idea is that at each time step, we use a neural network to decide how much of the risky asset to hold - this is our hedge. The hedge is based on the current asset price and is learned directly from data.

Custom deep hedging layer

```
class DeepHedgingLayer(Layer):
    # The __init__ is the function that runs each time we create a new
    ↪  instance of the class
    # self.variable defines a variable belonging to this class
    def __init__(self, Hedging_Networks, Premium_Network, n, **kwargs):
        super(DeepHedgingLayer, self).__init__(**kwargs)
        self.Hedging_Networks = Hedging_Networks
        self.Premium_Network = Premium_Network
        self.n = n

    # This is a method - an action this class can do.
    def call(self, price):
        """
        A forward pass through the deep hedging layer.
        """
        price_difference = price[:,1:,:] - price[:, :-1,:]   # S_k - S_{k-1}
        hedge = tf.zeros_like(price[:,0,:])                        # PL_0 = 0
        # Learn p_0
        premium = self.Premium_Network(tf.ones_like(price[:,0,:]))

        for j in range(self.n):                          # For each time step
            # delta_k = F_k(S_k)
            strategy = self.Hedging_Networks[j](price[:,j,:])
            # PL += delta_k * (S_k-S_{k-1})
            hedge += strategy * price_difference[:,j,:]

        outputs = premium + hedge                      # Final portfolio value
```

The `DeepHedgingLayer` layer is a direct implementation of the framework described in Chapter 5. It models the hedged portfolio value by learning the initial premium p_0 and applying the sequence of neural networks to the hedge positions δ_k^θ at each time step. The cumulative portfolio

value is then calculated by adding up the gains over all steps:

$$PL_T(p_0, \delta) = p_0 + (\delta^\theta \cdot S)_T = p_0 + \sum_{i=1}^{n} \delta_k^\theta (S_{k+1} - S_k).$$

The final deep hedging model, consisting of a network for learning the initial premium and n neural networks that learn the hedging strategy with input and output of dimension d, L layers in each network with `hidden_nodes` neurons in each hidden layer is built using the following function.

The final model

```
def build_dh_model(d, hidden_nodes, L, n):
    # List of networks for each time step
    Hedging_Networks = create_networks(
        d=d,
        hidden_nodes=hidden_nodes,
        L=L,
        n=n
    )
    # Initial premium layer
    Premium_Network = tf.keras.layers.Dense(d, use_bias=False)

    # Input: price paths of shape: (Nsample, n+1, d)
    price = tf.keras.Input(shape=(n+1,d))

    # Output : p_0 + PL
    outputs = DeepHedgingLayer(
        Hedging_Networks=Hedging_Networks,
        Premium_Network=Premium_Network,
        n=n
    )(price)

    model = tf.keras.Model(inputs=price, outputs=outputs)
    return model, Premium_Network, Hedging_Networks
```

6.1.3 Model Training and Risk Minimization

Now that we have the structure of our deep hedging model, we want to teach the model how to hedge. To train the model, we need to specify how it should improve over time by quantifying its performance using a loss function and an optimizer to update the model's weights to reduce that loss.

One of the most widely used loss functions in machine learning is the mean squared error (MSE). Since it was one of the first approaches to quantifying loss, it is a natural starting point for training neural networks. The MSE is simple and effective, however, it merely minimizes the average error with no regard for risk. Therefore, we also consider risk measures as loss functions following the discussion in Sections 4.5.3, 5.3 and 5.4.

The first risk measure chosen to act as a loss function is Value-at-Risk (VaR). Recall from Example 4.5.1 that VaR is given by

$$\mathrm{VaR}_\alpha(X) = \inf_{m \in \mathbb{R}} \{\mathbb{P}(-X \leq m) \geq \alpha\}.$$

VaR is very popular in finance, used widely in banks, regulation and risk management. We can think of VaR as the worst expected loss. For instance, a 95% VaR of £100 means that there is a 5% chance that an investor could lose £100. As a risk measure, it is very intuitive and interpretable, however, it ignores what happens in the worst 5%.

Given the popularity and drawbacks of Value-at-Risk, we are interested to explore how well it quantifies the quality of a hedge. Since VaR is expressed as a quantile of $-X$, it can have difficulties when computing gradients. In our implementation, we use an approximation of VaR commonly seen in machine learning.

Value-at-Risk loss function

```
def VaR(alpha=0.05):                        # Default is the 95th percentile
    def loss(y_true, y_pred):
        res = y_true - y_pred               # Compute the residual
        return tf.reduce_mean(tf.maximum(alpha * res, (alpha - 1) * res))
return loss
```

Since the first choice ignores extreme tail losses, we want to compare it to a risk measure designed to address this drawback. If the worst 5% of losses are bad, we look at the average of those bad outcomes using the Conditional Value-at-Risk (CVaR). Recall from Example 4.5.3 that CVaR, given by

$$\mathrm{CVaR}_\alpha(X) = \frac{1}{\alpha}\int_0^\alpha \mathrm{VaR}_s ds = \mathbb{E}[-X \mid -X \geq \mathrm{VaR}_\alpha(X)],$$

is both convex and coherent. We can thus test whether it handles tail losses well and whether its coherence can, indeed, discourage diversification.

Conditional Value-at-Risk loss function

```
import tensorflow_probability as tfp

def CVaR(alpha=0.05):      # Focus on the worst 5% of the outcomes by default
    def loss(y_true, y_pred):
        res = tf.abs(y_true - y_pred)            # Compute absolute residual

        # Find threshold loss such that only an alpha proportion of the worst
        ↪ losses exceed it
        var_alpha = tfp.stats.percentile(res, 100 * (1 - alpha), axis=0,
        ↪ interpolation='linear')
        # Create a binary mask that selects only the losses that are in the
        ↪ worst-case tail
        mask = tf.cast(res >= var_alpha, tf.float32)
        # Sum the tail losses and divide them by the number of values in the
        ↪ tail
        # Add a small constant in the denominator to avoid division by 0
        cvar = tf.reduce_sum(res * mask) / (tf.reduce_sum(mask) + 1e-6)
        return cvar
    return loss
```

Lastly, we quantify loss using entropic risk. This risk measure comes from exponential utility theory and is closely linked to information theory, as seen in example 4.5.13. Recall from example 4.5.5 that entropic risk, given by

$$\rho_\beta(X) = \frac{1}{\beta}\log \mathbb{E}[e^{-\beta X}],$$

is convex. Here, $\beta > 0$ is the risk aversion parameter. A small β means that the agent is less risk averse, and the function behaves more like the average: as $\beta \to 0$, $\rho_\beta(X) \to \mathbb{E}[-X]$. A large β means that the investor is risk-averse as the function emphasizes the worst-case losses. We want to avoid using values that are too large in training because this can lead to numerical instability.

Entropic risk loss function

```
def entropic(beta=3):                    # Moderate risk aversion by default
    def loss(y_true, y_pred):
        res = tf.abs(y_true - y_pred) # Compute absolute residual
         # Prevent overflow or underflow in the exponential calculation
        scaled = tf.clip_by_value(beta * res, -50.0, 50.0)
        entropic_risk = tfp.math.reduce_logmeanexp(scaled, axis=0) / beta
        return tf.reduce_mean(entropic_risk)
    return loss
```

The MSE loss provides us with a risk-neutral view, showing how the model performs on average and providing a reference point. Value-at-Risk is popular but limited. It focuses on a particular quantile, telling us what size loss we should not exceed. Conditional Value-at-Risk, in turn, explores the severity of tail risk and encourages the model to manage large loss events instead of avoiding them. Finally, entropic risk considers the agent's risk preferences, showing how sensitive the strategy is to large losses. All four loss functions give us the potential to understand a full picture of model performance.

To train the model, we use the Adam optimizer, introduced by Kingma and Ba (2017). Adam stands for *Adaptive Moment Estimation* and is one of the most widely used optimization algorithms in deep learning today. It works by tracking both the first moment (the mean) and the second moment (the uncentered variance) of the gradients. We do not give a full explanation of Adam but refer the interested reader to the original paper by Kingma and Ba (2017) for details.

We use Adam instead of standard stochastic gradient descent (SGD) because it is more robust and efficient. Unlike SGD, which uses a fixed learning rate for all parameters, Adam dynamically adjusts the learning rate for each parameter based on how the gradients evolve over time. Because of this, Adam is better for complex losses such as CVaR or entropic risk.

6.2 How Well Does Deep Hedging Learn the Black-Scholes Market?

We begin with the most straightforward setup: a market that closely follows the assumptions of the Black-Scholes model. As discussed in Section 4.4, the underlying follows a geometric Brownian motion (see Definition 4.4.3), the volatility is constant, and there are no frictions. While the trading is not continuous, the discretization is fine enough to approximate continuous-time trading reasonably. Despite the deep hedging framework being designed to handle more complex and realistic markets, we start here for two reasons. First, this allows us to compare our results to the analytical solutions from the Black-Scholes model and verify that deep hedging indeed works. Second, using a clean and controlled setting makes it much easier to interpret what the model is doing. Before diving into scenarios with market frictions or other complications, we want to understand how deep hedging works when things are simple.

To verify our results, we define a function that allows us to calculate the price of an option and the Greeks using the analytical solutions from the Black-Scholes model.

Analytic solution of the Black-Scholes PDE

```
def BlackScholesPrice(T, S, K, sigma, r=0, greeks=False, option_type='call'):

    d_1 = (np.log(S/K) + (r + sigma**2) * T)/(sigma * np.sqrt(T))
    d_2 = d_1 - sigma * np.sqrt(T)

    # Greeks common to both call and put
```

```
    gamma = norm.pdf(d_1)/(S * sigma * np.sqrt(T))
    vega = S * np.sqrt(T) * norm.pdf(d_1)

    # Price and Greeks for the European call option
    if option_type=='call':
        price = S * norm.cdf(d_1) - K * np.exp(-r * T)*norm.cdf(d_2)
        delta = norm.cdf(d_1)
        theta = - (S*norm.pdf(d_1)*sigma)/(2*np.sqrt(T)) -
        ↪  r*K*np.exp(-r*T)*norm.cdf(d_2)
        rho = K*T*np.exp(-r*T)*norm.cdf(d_2)

    # Price and Greeks for the European put option
    elif option_type=='put':
        price = K * np.exp(-r * T) * norm.cdf(-d_2) - S * norm.cdf(-d_1)
        delta = norm.cdf(d_1) - 1
        theta = - (S*norm.pdf(d_1)*sigma)/(2*np.sqrt(T)) +
        ↪  r*K*np.exp(-r*T)*norm.cdf(-d_2)
        rho = -K*T*np.exp(-r*T)*norm.cdf(d_2)

    else:
        raise ValueError(f"Invalid option type: '{option_type}'. Select 'put'
        ↪  or 'call'.")

    if greeks==True:
        return {'price' : price, 'delta' : delta, 'gamma' : gamma,
            'theta' : theta, 'vega'  : vega, 'rho'   : rho
        }
    else:
        return price
```

This function takes the initial asset price S and the volatility, σ, of the underlying. We also specify whether we are using a put or call European option using `option_type` and provide the time-to-maturity of the option T as well as its strike price K. Lastly, we specify whether we want the values of the Greeks by setting `greeks=True` or `greeks=False`.

6.2.1 Simulating GBM Data

In order to train our neural network to price and hedge, we need data for the network to learn from. Since we are investigating a near-perfect Black-Scholes market, we use simulated geometric Brownian motion data. Recall from Definition 4.4.3 that geometric Brownian motion follows

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

We, thus, define a function that simulates such data.

Simulating geometric Brownian motion data

```
import numpy as np

def generate_GBM_data(N, sigma, d, n, K, T, S):

    time_grid = np.linspace(0, T, n+1) # Discretized time-to-maturity
    dt = T/n                           # Time increment

    BM_path_nozero = np.cumsum(
```

```
        np.random.normal(loc=0, scale=np.sqrt(dt), size=(N,n,d)),
        axis = 1
    )
    # Add 0 to each path for B_0=0
    BM_path = np.concatenate([np.zeros((N,1,d)), BM_path_nozero], axis = 1)

    # Convert BM to a GBM price path: S_t = S exp(sigma B_t - 0.5sigma^2 t)
    price_path = S * np.exp(sigma * BM_path - 0.5 * sigma**2 *
    ↪  time_grid[None, :, None])

    # Calculate the European option payoff
    if option_type=='call':
        payoff_func = lambda x: 0.5* (np.abs(x-K)+x-K)
    else:
        payoff_func = lambda x: 0.5 * (np.abs(x - K) + K - x)
    payoff = payoff_func(price_path[:,-1,:])

    return price_path, payoff
```

The `generate_GBM_data` first creates a time grid by dividing the time-to-maturity T into n discrete time steps. Then, we simulate N Brownian motion paths for d assets. We then compute the GBM price paths using initial asset price S with volatility σ and the corresponding European option payoff using its strike price K. Note that we use $\mu = 0$ in this data generating function.

We generate the data using the following variables.

```
import numpy as np

price_paths, payoffs = generate_GBM_data(
    N = 10**5,     # Number of samples to generate
    sigma = 0.2,   # Volatility
    d = 1,         # Number of assets
    n = 100,       # Number of hedging steps
    K = 1.0,       # Strike of the option
    T = 1.0,       # Time-to-maturity
    S = 1.0        # Initial asset price
)
```

6.2.2 Training the Model

We now train the model using the simulated GBM data. The goal is to understand how different loss functions affect the network's performance. Since each loss function represents a different attitude to risk and has a different purpose, this allows us to see how the model behaves under various objectives.

We begin by splitting the simulated data into training and testing sets.

```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(
    price_paths, payoffs,
    test_size = 0.2,         # 20% of data is used for testing
    random_state = 42,
    shuffle = True)
```

Splitting data into training and testing sets is common in machine learning and is generally considered good practice. The primary interest is not knowing how well an algorithm performs on data it has already seen. In practice, we seek predictions for data for which we do not know the true values. Hence, we want to know how well the model generalizes to new and unseen data, so we split the data into two sets. The training set is used to fit the neural network and adjust its parameters, while the testing is set aside for now - we will use it to evaluate the model's performance. This helps us avoid overfitting, common in deep learning, when the model memorizes the patterns in the training data but fails to generalize.

Next, we define a learning rate schedule.

```
import tensorflow as tf

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate = 0.005,
    decay_steps = 1000,
    decay_rate = 0.96,
    staircase = True    # The learning rate is not reduced continuously
)
```

Recall from Section 3.4 that a learning rate controls how much the weights are adjusted in each learning step. Instead of a fixed learning rate, we use one that decays exponentially. We begin with an initial learning rate that is larger and then gradually reduce it over time, multiplying it by a factor of 0.96 every 1000 training steps. Then, at the beginning of training, the model can explore the loss to ensure convergence, but in the later stages, the smaller learning rate ensures precision.

The last step before training is defining an early stopping condition.

```
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor = 'val_loss',
    patience = 3,
    restore_best_weights = True
)
```

In addition to using a train-test split, we also use a validation set. During training, a fraction of the training data is split off and then used to evaluate the model's performance after each epoch. This set is not used to update weights. This, once again, helps improve the model's ability to generalize to unseen data.

An early stopping callback automatically stops the neural network training once the model stops improving based on its performance on the validation set. If the validation loss has not improved for three consecutive epochs, the training is stopped, and the model reverts to its best version so we do not keep the weights for a possibly overfitted network.

This is particularly informative for our goal of comparing the performance of different loss functions. Using an early stopping callback in training lets us evaluate both the speed and quality of each loss function as we can see which generalize the best and which converge the fastest.

We are finally ready to train our deep hedging model. To keep track of the results, we first initialize empty lists for what we want to record. Then, we train a separate model for each loss function.

```
# Initialize lists
losses = ['mse', VaR(alpha=0.05), CVaR(alpha=0.05), entropic(beta = 3)]
```

```
titles = ['MSE', 'VaR', 'CVaR', 'Entropic']
histories = []
models = []
premiums = []
hedges = []
times = []
final_hedges = []

# Training Loop
for i, lossfn in enumerate(losses):
    tf.keras.backend.clear_session()    # Avoid interactions between models
    print(f"\nTraining model with {titles[i]} loss.")

    model, Premium_Network, Hedging_Networks = build_dh_model(
        d = 1,                  # 1 asset
        hidden_nodes = 32,      # 32 neurons per layer
        L = 3,                  # 3 layers
        n = 100                 # 100 time steps
    )

    model.compile(
        optimizer = keras.optimizers.Adam(learning_rate=lr_schedule),
        loss = lossfn
    )

    start = tf.timestamp()          # Track training time
    history = model.fit(
        x=xtrain, y=ytrain,
        epochs = 50,
        batch_size = 1024,
        validation_split = 0.1,     # 10% of the training data used for
        ↪  validation
        callbacks = early_stop,
        verbose = False
    )
    time = tf.timestamp() - start   # Total training time

    # Predict hedged portfolio value on the test set
    hedge = model.predict(xtest, verbose = 0)
    # Extract the learned initial premium p_0 from Premium_Network
    premium = float(Premium_Network(tf.ones([1,1])).numpy().squeeze())
    # Compute the hedge at final time step, delta_{n-1}
    final_hedge = Hedging_Networks[-1](xtest[:, -2, 0:1]).numpy().flatten()

    # Store the results before training a new model
    models.append(model)
    premiums.append(premium)
    histories.append(history)
    hedges.append(hedge)
    times.append(float(time))
    final_hedges.append(final_hedge)
```

6.2.3 Analysis of Results

With the models trained, we now explore what they learned. In this subsection, we try to understand how our implementation of the framework works in a perfect setting and how different loss functions change the results. The full Python code for each of the experiments can be found in *Pricing and Hedging Using Neural Networks* (2025).

First, we want to compare the payoff predictions for the test set of the deep hedging strategy and the true payoff of the European call option with both strike and maturity 1. Figure 6.2 shows this for all four loss functions. Visually, all four models seem to match the true payoff reasonably well, which validates that the model is not just pricing correctly on average, that is, according to the loss function, but also replicating the correct payoff structure for different asset values.

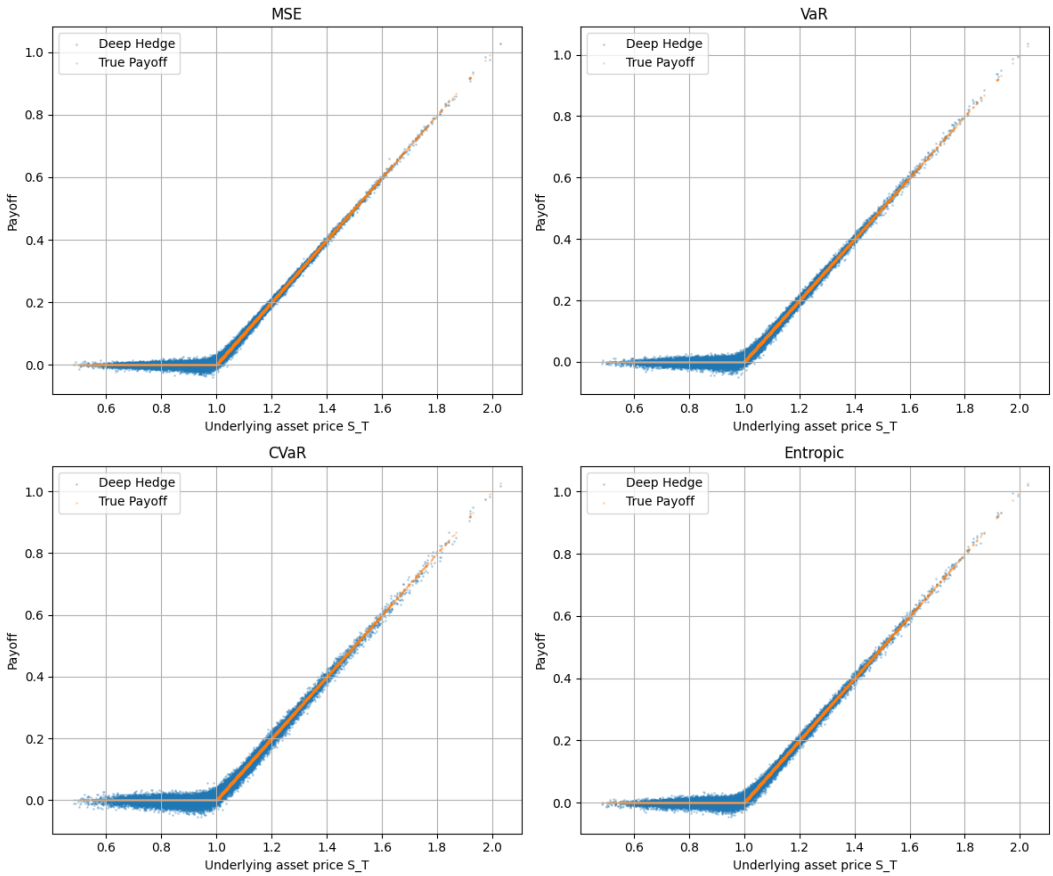


Figure 6.2: The predicted payoffs vs. true payoffs from a deep hedging model under different loss functions

MSE matches the European call option payoff $C(S_T, T) = (S_T - K)^+$ the most closely, with the least spread around the payoff line. Notice that MSE slightly underestimates the payoff for higher values of the terminal asset price S_T. This is because MSE tries to keep the average error small for all price paths, and high asset prices are less common, so the model focuses more on the whole picture.

Value-at-Risk also gives a very good fit, with slight overestimation near high S_T. This is because VaR focuses more on controlling losses at the 95% quantile rather than being accurate when the prices (and hence the payoffs) are high.

Both CVaR and entropic risk have slightly more variance. CVaR is mostly spread out below the strike, which matches the theory - CVaR focuses on minimizing large losses, so the payoff

for $S_T < K = 1$ is underestimated. Entropic risk displays similar behavior, most likely due to a relatively high risk aversion parameter $\beta = 3$.

We can verify the discussion above by looking at the learned premium. Figure 6.3 shows the learned premium p_0 for each loss function as well as the error $p_0 - C_0$, where C_0 is the analytical price calculated using the Black-Scholes analytical solution. Overall, we see that the model performs well since all errors are below 10^{-3}, showing that each loss function can price the option with good accuracy.

Loss function	Premium	Error
MSE loss	0.07964585	0.00038614
VaR loss	0.07945513	0.00019542
CVaR loss	0.07826568	-0.00099403
Entropic risk	0.07961762	0.00035791

Figure 6.3: Learned model premiums and pricing errors under different loss functions in the Black-Scholes market.

VaR achieves the smallest error of just 0.00019542. Both MSE and entropic risk have very similar predictions and, by penalizing overall error and extreme losses respectively, push the premium estimate higher. CVaR is the only loss function that underestimates the price, which aligns with the theory. Expected shortfall is designed to focus on the worst-case losses, so the model trained with CVaR sacrifices performance in the average case to do better when losses are large.

While the scatterplots and premium errors provide insights into the final performance, they do not give us a complete view. A model can predict the price or the payoff well, but that does not mean it learned efficiently and without overfitting. We thus turn to Figure 6.4 and Figure 6.5, presenting training times and validation curves, respectively. These results will help us understand how each loss function affects convergence, stability, and generalization throughout training.

Model	Training time (seconds)
MSE loss	167.45
VaR loss	106.71
CVaR loss	88.60
Entropic risk	86.35

Figure 6.4: Training times for each loss function.

Using the mean squared error as a loss function provides us with a smooth and stable convergence, however, it is relatively slow. Training this model took the longest, with 48 epochs and a total training time of 167 seconds. Indeed, MSE aims to minimize errors throughout the entire distribution, not just risky cases, making the optimization task broader.

The training and validation loss for Value-at-Risk converge quickly, with the most improvement until the fifth epoch. It is quicker than MSE loss both in terms of time and epochs as VaR focuses on a fixed quantile, so the model only needs to fit that part well. However, VaR is slower than the following two loss functions, likely due to a non-smooth gradient.

The validation curve of Conditional Value-at-Risk is more noisy and decreases more gradually than the other loss functions. CVaR considers the entire tail beyond the chosen quantile, not just a fixed point like VaR. Despite this, it trains relatively fast, just with less stability.

Both CVaR and entropic risk train faster than the other loss functions, entropic risk being the fastest with just 19 epochs and 86 seconds of training time. Note that both are convex risk measures and, thus, they encourage smooth strategies, which means they learn a more balanced hedge more quickly. Entropic risk is especially fast because it reacts strongly to big mistakes early on. It penalizes large losses exponentially, so the model quickly learns to avoid them.

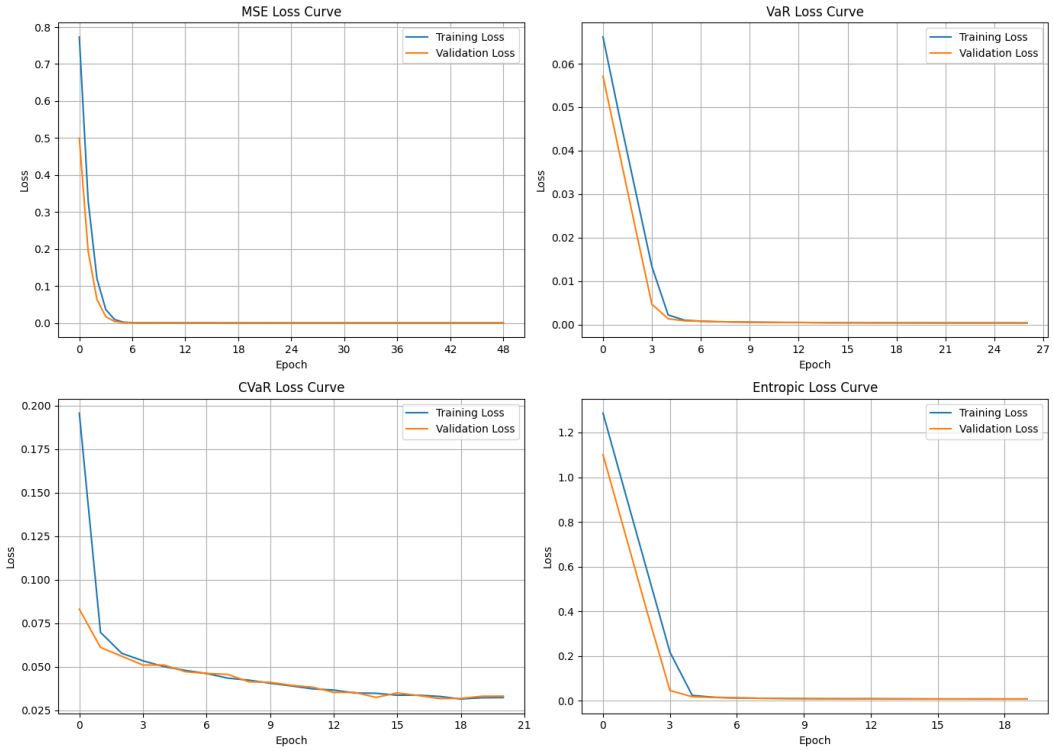


Figure 6.5: Training and validation loss curves for each loss function.

Another point of interest is residual statistics. Given the predicted payoffs $\hat{y}$ and the true payoffs y, we define the residual as $r = \hat{y} - y$. We are then interested in the mean residual

$$\mu_r = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i),$$

which measures bias - whether the model tends to systematically underpredict or overpredict. The standard deviation of the residuals,

$$\sigma_r = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i - \mu_r)^2},$$

measures spread, that is, how consistent the prediction errors are.

The residual statistics for each loss function are shown in Figure 6.6.

Loss	Mean residual	Standard deviation
MSE loss	-0.000032	0.008013
VaR loss	-0.000291	0.009880
CVaR loss	-0.001497	0.012988
Entropic risk	-0.000122	0.010160

Figure 6.6: Mean and standard deviation of residuals for each loss function.

As expected, the MSE model is the most balanced as its average error is nearly zero and makes consistent predictions. VaR is also reasonably accurate but shows more variation since it only focuses on the worst-case quantile and ignores the rest. CVaR, which tries to control tail losses, clearly "plays it safe": it underpredicts more and is less consistent. Entropic risk sits in between

- it is more cautious than MSE but not as conservative as CVaR. These statistics confirm the perspective each model has towards risk.

Lastly, an important goal of the deep hedging framework is to learn the hedging strategy. Recall from Section 4.4 that the Black-Scholes model allows us to eliminate risk entirely. Indeed, choosing $\Delta_t = \frac{\partial V}{\partial S}$, the Greek delta, replicates the option's price perfectly in a complete market. Thus, we compare the final hedging value, δ_{n-1}, with the theoretic ideal to see how well the models learn hedging strategies. This can be seen in Figure 6.7.

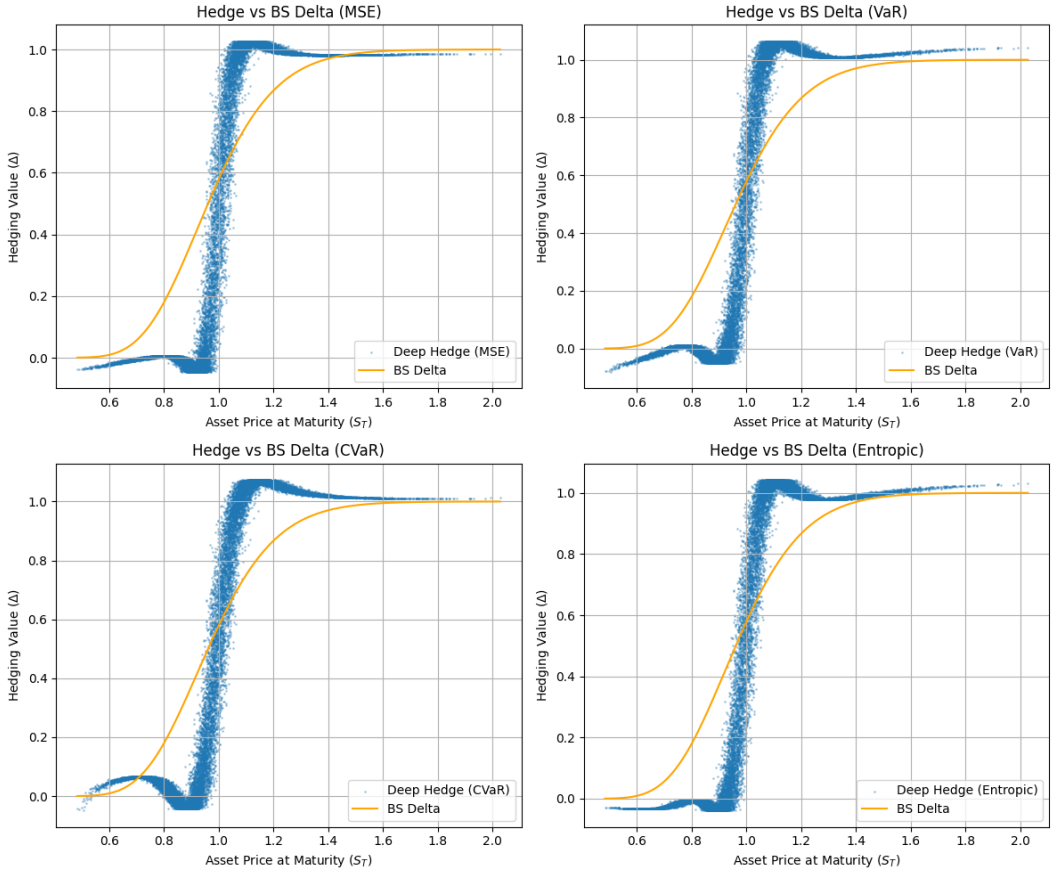


Figure 6.7: Comparison of deep hedging strategies with Black-Scholes delta across different loss functions.

Across all loss functions, we observe the characteristic S-shaped curve, which matches the shape of the Black-Scholes delta and suggests the model has learned to hedge reasonably well. However, each loss function has a specific behavior. The model trained with MSE is smooth and well-aligned but slightly off around $S = K = 1$ since it treats all errors equally. VaR also follows the theoretical hedge closely but is more scattered, representing its focus on quantile-level risk. The CVaR hedge has more noise, especially around the strike, because it targets the worst-case losses beyond the 95% threshold, sacrificing smoothness for caution. Finally, entropic risk leads to a steep hedge around the strike: its exponential penalty on large losses forces the model to react strongly to high-risk scenarios. Each loss function shows how different ways of thinking about risk influence the strategy the model learns.

6.3 Why Risk Measures Matter: A Heavy-Tailed View

So far, we have seen that deep hedging performs well in the clean and controlled Black-Scholes market, where returns are normally distributed and extreme events are rare. However, such markets are rare, if not impossible, in practice.

In Subsection 6.1.3, we selected a varied range of loss functions for testing and briefly mentioned that we are interested in Value-at-Risk not only due to its popularity but also a significant limitation - ignoring tail risk. Indeed, VaR only considers a single quantile and ignores everything that happens beyond it. Because of this, it can overlook both large losses in the left tail and large gains in the right tail.

As outlined in McNeil et al. (2016), VaR was created in the early 1990s since financial institutions needed systematic ways to report risk and control their risk exposure. However, during the 2007–2009 financial crisis, it received heavy criticism for failing to consider the magnitude of losses beyond the single quantile threshold. A reliable risk management framework must be sensitive to such extreme events.

To explore this problem, we contrast the idealized experiment in Section 6.2 with one that evaluates the deep hedging framework in a setting with heavy-tailed price and return distributions. This allows us to understand which model is the most robust under extreme market conditions.

6.3.1 Simulating Heavy-Tailed Data

A heavy-tailed distribution is a probability distribution where extreme (very large or very small) values are more likely than in a normal distribution. In finance, this implies that large losses or gains happen more often than a Gaussian distribution, used when simulating data for our first test, would predict. There are many ways to simulate heavy-tailed data. Here, we choose jump-diffusion.

6.3.1.1 Merton Jump-Diffusion Model

The Merton jump-diffusion model, introduced in Merton (1976) to account for discontinuities and extreme market movements, extends the Black-Scholes model by letting the asset price experience sudden and discontinuous jumps along with the movements from Brownian motion. These jumps make the model more realistic, especially by creating heavy tails, where very large gains or losses happen more often than a normal distribution would predict.

Under the Merton model, the stock price follows the stochastic differential equation

$$\frac{dS_t}{S_t} = (\mu - \lambda k)\, dt + \sigma dB_t + (e^Y - 1) dN_t,$$

where μ is the drift, Y is the normally distributed logarithm of a jump size with $Y \sim N(\mu_{\text{jump}}, \sigma^2_{\text{jump}})$ so that $k := \mathbb{E}[e^Y - 1]$ is the expected percentage change in the asset price due to a jump, B_t is a Brownian motion with volatility σ and N_t is a Poisson process with intensity λ.

For simulating asset paths, it is common to use the log-form expression

$$\ln S_t = \ln S_0 + \left(\mu - \frac{1}{2}\sigma^2\right) t + \sigma B_t + \sum_{i=1}^{N_t} Y_i, \tag{6.1}$$

where each Y_i represents the logarithmic jump size.

Compared to the Black-Scholes model, the Merton jump-diffusion model introduces randomness in the number and size of jumps, leading to a heavier-tailed distribution for returns. Since the Black-Scholes model was already explained in detail in Section 4.4, the reader is referred to Merton (1976) for a full discussion of the jump-diffusion extension.

We simulate the data for this experiment using (6.1):

Simulating Heavy-Tailed Jump-Diffusion Data

```
def generate_jump_diffusion_data(jump_int, mu_jump, sigma_jump, N, sigma, d,
↪  n, K, T, S, option_type='call'):
    time_grid = np.linspace(0, T, n+1)     # Discretized time-to-maturity
    dt = T/n                               # Time increment

    # Brownian part (same as with GBM)
    BM_path_nozero = np.cumsum(np.random.normal(loc=0, scale=np.sqrt(dt),
    ↪  size=(N,n,d)), axis=1)
    BM_path = np.concatenate([np.zeros((N,1,d)), BM_path_nozero], axis=1)

    # Poisson jumps
    number_jumps = np.random.poisson(jump_int*dt, size=(N,n,d))
    jump_sizes = np.random.normal(loc=mu_jump, scale=sigma_jump,
    ↪  size=(N,n,d))
    jumps = number_jumps * jump_sizes
    jumps = np.concatenate([np.zeros([N,1,d]), jumps], axis=1)

    # Combine BM and jumps, drift mu=0
    log_price = np.log(S) - 0.5*sigma**2*time_grid[None,:,None] +
    ↪  sigma*BM_path + jumps
    price_paths = np.exp(log_price)

    # Calculate the European option payoff
    if option_type == 'call':
        payoff_func = lambda x: 0.5*(np.abs(x-K)+x-K)
    else:
        payoff_func = lambda x: 0.5*(np.abs(x-K)+K-x)
    payoff = payoff_func(price_paths[:,-1,:])

    return price_paths, payoff
```

In Section 6.2, we compared the premium learned by the network to the analytic solution to the Black-Scholes PDE for pricing European options. Since we are now using a different data-generating model, we use the closed-form solution for the price of a European call option given in Merton (1976):

$$C(S,t) = \sum_{m=0}^{\infty} \frac{e^{-\lambda T}(\lambda T)^m}{m!} C_{\text{Black-Scholes}}(S,t;r_m,\sigma_m^2,K),$$

where $r_m = r - \lambda k + mT\mu_{\text{jump}}$ and $\sigma_m^2 = \sigma^2 + \frac{m\sigma_{\text{jump}}^2}{T}$. This represents the Black-Scholes call option price conditional on m jumps. We define the pricing function as follows.

Merton model pricing formula

```
from scipy.stats import poisson

def MertonJumpDiffusionPrice(S, K, T, r, sigma, jump_int, mu_jump,
↪  sigma_jump):
    # Expected jump size
    k = np.exp(mu_jump + 0.5*sigma_jump**2)-1
    lambda_T = jump_int*T
    price = 0.0
```

```
    # Sum to 99.9% of the probability mass of the Poisson distribution with
    ↪  mean lambda*T. This is used as a rule of thumb.
    N_terms = int(np.ceil(lambda_T + 4*np.sqrt(lambda_T)))

    # Sum over the number of jumps from m=0 to N_terms
    for n in range(N_terms):
        # Calculate sigma_m and r_m
        sigma_m = np.sqrt(sigma**2 + m*sigma_jump**2/T)
        r_m= r - jump_int*k + m*mu_jump/T
        weight = poisson.pmf(m, lambda_T)

        # Sum
        bs_price = BlackScholesPrice(T, S, K, sigma_m, r=r_m)
        price += weight*bs_price
    return price
```

6.3.1.2 Is the Data Heavy-Tailed?

Using this function with parameters identical to those in Subection 6.2.1, that is, $N = 10^5$, $\sigma = 0.2$, $d = 1$, $n = 100$, $K = S = T = 1$ for a call option, as well as setting $\mu_{\text{jump}} = 0.75$, $\sigma_{\text{jump}} = 0.001$ and jump intensity, representing expected number of jumps per unit time, $\lambda = 10$. This results in a terminal price S_T kernel density estimate[1]shown in Figure 6.8.

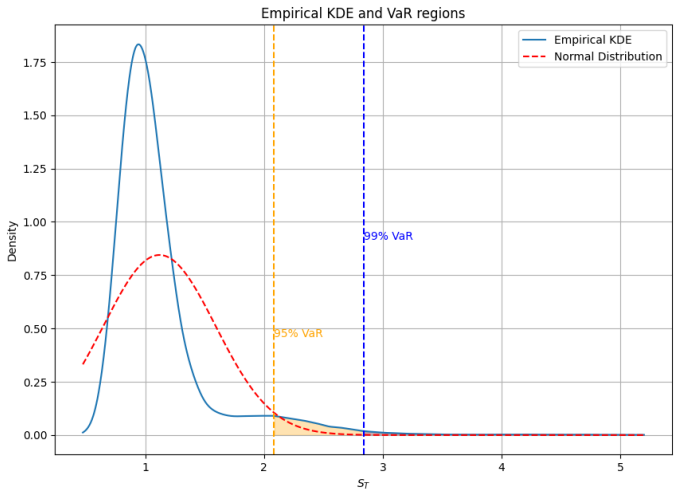


Figure 6.8: Empirical and normal density estimates of terminal prices S_T with 95% and 99% Value-at-Risk (VaR) thresholds.

Figure 6.8 confirms the presence of a heavy right tail visually. Observe that decay past the 95th quantile is much slower than that of the normal distribution so tail values have more probability mass. This suggests that extremely high terminal prices are more frequent than in the Gaussian price paths.

We can further confirm heavy tails using kurtosis and an estimate of the tail index. Kurtosis of a probability distribution is a statistic which tells us about the shape of a distribution, in particular, how heavy the tails are compared to those of a Gaussian distribution. For a random

[1]A kernel density estimate (KDE) is a non-parametric way to estimate the probability density function of a random variable.

variable X, it is given by

$$\text{kurtosis} = \frac{\mathbb{E}[(X-\mu)^4]}{\sigma^4}.$$

Normal distribution has a kurtosis of 3 so a value larger than 3 indicates that the distribution of X is leptokurtic, that is, there are more extreme events than usual so the tails are heavy.

Moreover, the Hill estimator gives us an estimate of the tail index α of the distribution, telling us how steeply the largest values diverge from the rest. Given a sorted sample $X_{(1)}, \ldots, X_{(N)}$, we find the Hill estimate for the right tail by selecting k largest values $X_{(N-k+1)}, \ldots, X_{(N)}$ and finding the tail index

$$\hat{\alpha} = \left(\frac{1}{k} \sum_{i=1}^{k} \log \frac{X_{(N-i+1)}}{X_{(N-k)}} \right)^{-1}.$$

A tail index smaller than 4 indicates heavy tails.

The distribution of the simulated terminal prices S_T has kurtosis 39.61 and the Hill estimator is $\hat{\alpha} = 3.88$. Both statistics suggest that the distribution is clearly non-Gaussian - it is highly skewed towards extreme values. Large upwards movements in the asset price occur with non-negligible probability. This is where VaR typically fails - by underestimaing the probability of high losses or gains.

6.3.2 Training and Results

To see how our deep hedging models respond to extreme events, we keep the training the same as in Subsection 6.2.2 except for two adjustments: architecture and callback settings.

Firstly, we reduce the number of neurons yet increase the number of layers. We do this because increasing the number of layers from $L = 3$ to $L = 5$ allows the network to learn more irregular, discontinuous, complex data. We also set `hidden_nodes` to 16 instead of 32 because testing shows instability with more neurons (see *Pricing and Hedging Using Neural Networks* (2025)).

Second, we change the early stopping patience from 3 to 5 epochs, meaning that the network will stop training early if the validation loss does not improve for five consecutive epochs rather than three. Jump-diffusion data is more volatile, so the validation loss can fluctuate even without overfitting. Giving the model more patience allows it to keep training even when small fluctuations occur, which is important to capture rare extreme events.

6.3.2.1 Payoff Comparison and Premium Pricing Errors

We revisit the same initial results as in Section 6.2. Figure 6.9 compares the true European call option payoff to the payoffs predicted by each model. Unlike GBM data, the jump-diffusion data makes the differences between the loss functions more clear.

Similarly to GBM data, MSE performs well overall with some underpricing at extreme values of S_T. This happens because MSE spreads its attention evenly across all scenarios, so it does not focus heavily on rare extreme jumps.

VaR gives the worst match, with significant underpricing for $S_T > 2$. This is expected because VaR only cares about losses up to a fixed threshold, ignoring anything worse. Once the market becomes extreme, VaR does not model those scenarios properly, leading to poor hedging.

CVaR fits the payoff function very closely, even for high tail prices with surprisingly little spread. Indeed, this is what expected shortfall does best - it manages extreme risk past the specified threshold.

Entropic risk also performs well but with slightly more spread than CVaR. While it still underprices in the right tail, it does so much less than VaR. This makes sense because, with moderate risk aversion, the entropic loss strongly penalizes large errors but still allows for a little bit more flexibility than CVaR.

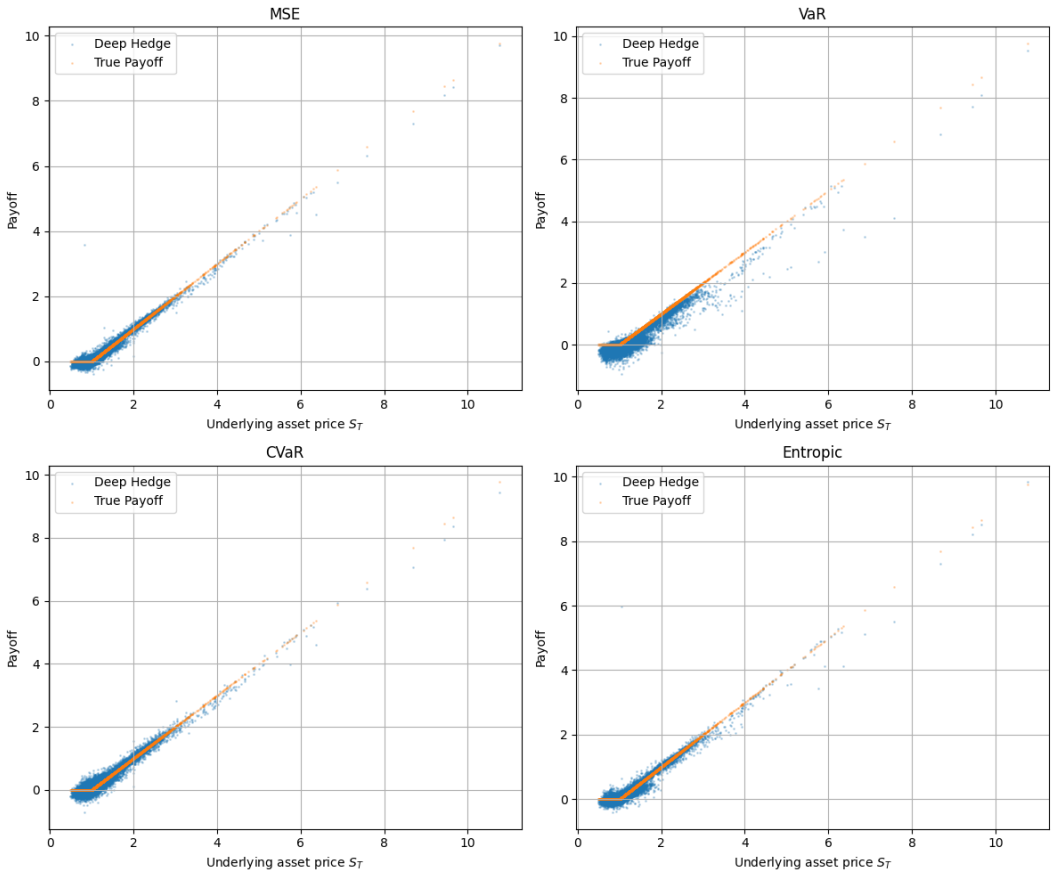


Figure 6.9: The predicted payoffs vs. true payoffs from a deep hedging model under different loss functions

Figure 6.10 shows the learned premium and its error, $p_0 - C_0$, where C_0 is calculated using Merton's pricing formula given above.

Despite Figure 6.9 showing slight underpricing at extreme values visually, MSE overprices the premium. This can, once again, be explained by the aim of MSE - minimizing average error. Most scenarios have small to moderate returns, so the model slightly increases the premium to compensate for small underestimations elsewhere.

VaR has the largest error by far, underestimating the premium significantly and even predicting a negative value. Indeed, VaR completely ignores the losses between the 95th quantile, leading to poor performance when the outcomes between this quantile carry a significant probability mass.

CVaR significantly overprices the premium. This is natural because CVaR focuses only on the worst outcomes and thus deliberately prices very conservatively, even if it leads to overestimating in most typical cases. This is actually desirable when the goal is safety in extreme markets.

Entropic risk gives the best premium estimate. It balances penalizing big mistakes (like CVaR) with keeping attention on moderate outcomes (like MSE). It captures risk preferences more smoothly than the other loss functions.

Training with GBM data shows almost perfect alignment between the deep hedge and the true payoff, while the models trained on jump-diffusion data show clear deviations in tails, especially for VaR. Heavy-tailed data reveals the weaknesses of loss functions: VaR underhedges, while CVaR gives an accurate match but overprices. Indeed, the GBM data shows a perfect view of the model since all losses converge to a similar hedge, hiding the differences in risk sensitivity,

Loss function	Premium	Error
MSE loss	0.10355181	0.06391845
VaR loss	-0.01336549	-0.05299885
CVaR loss	0.13742767	0.09779432
Entropic risk	0.08945036	0.04981700

Figure 6.10: Learned model premiums and pricing errors under different loss functions with jump-diffusion data.

suggesting that heavy tail models are essential for a meaningful comparison of models determined by different risk measures.

6.3.2.2 Validation Curves and Training Time Analysis

After the initial overview of the simplest of results, payoff prediction, and premium errors, we are now interested in the efficiency of each loss function. Figure 6.11 shows the training times for the models with each loss function, while Figure 6.12 displays the training and validation losses across epochs.

Training with MSE leads to stable and smooth convergence, but it is slow compared to the other loss functions. MSE tries to minimize the average error across all data points. Jump-diffusion data contains rare yet extreme events, which MSE treats the same as any other data point, hence, it takes more epochs to reach a low loss. However, because it fits the overall data well, training and validation curves match closely, and there is little overfitting.

The model trained with Value-at-Risk takes the longest to converge, with the most epochs and the longest training time. Since VaR only focuses on a fixed quantile and ignores what happens beyond it, the model struggles to learn from rare, extreme events in jump-diffusion data. This makes convergence slower and noisier. The training and validation losses show jumps because the model sometimes fits the quantile well but misses it again as extreme outcomes randomly appear.

CVaR loss also converges relatively slowly and starts at a very high initial loss. CVaR focuses on the worst outcomes, so the model initially struggles to identify and hedge the extreme scenarios. Both training and validation losses are noisy, but, unlike VaR, they track each other fairly closely. This means that while the learning process is volatile due to fitting the tails, the model does not heavily overfit, and the noise comes from the difficulty of modeling extreme events, not memorization.

The model trained with the entropic risk loss is the quickest to converge, with the least number of epochs and the shortest training time. Despite starting high, training and validation losses drop quickly and smoothly. Entropic risk heavily penalizes large losses but does so smoothly instead of focusing on a single threshold. This makes the model quickly avoid major mistakes without getting distracted by small noise. Moderate risk aversion means the model stays cautious about extreme outcomes while still fitting the overall data well. As a result, training is fast, stable, and shows almost no overfitting.

Model	Training time (seconds)
MSE loss	109.49
VaR loss	118.60
CVaR loss	109.72
Entropic risk	89.27

Figure 6.11: Training times for each loss function.

Training on jump-diffusion data leads to important changes compared to GBM. Overall, convergence becomes more difficult, causing longer training times for most models. However, not

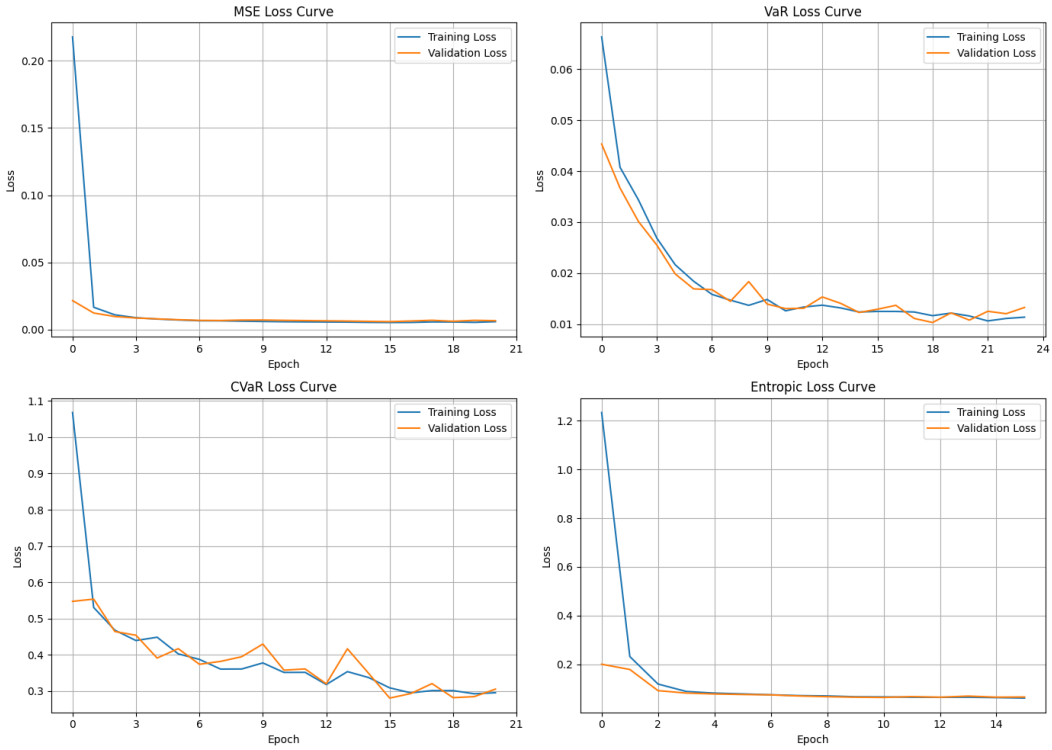


Figure 6.12: Training and validation loss curves for each loss function.

all models are affected equally. MSE, the slowest to train under GBM, now trains significantly faster because the sharp jumps in the data make the general payoff shape easier to learn. On the other hand, VaR and CVaR take longer to train. VaR especially struggles, taking the longest time, because it only focuses on a fixed quantile and ignores the size of extreme outcomes, which are now more common. Entropic risk remains the fastest to train, just like with GBM, thanks to its smooth exponential penalization of large losses. The ranking of training times changes: in GBM, MSE was the slowest, but under jump-diffusion, VaR is the slowest and MSE becomes much faster.

6.3.2.3 Right Tail Hedging Errors

Figure 6.13 compares the average absolute hedging error in the right tail when S_T exceeds the 95th percentile across the different loss functions and the frequency of errors larger than 0.5.

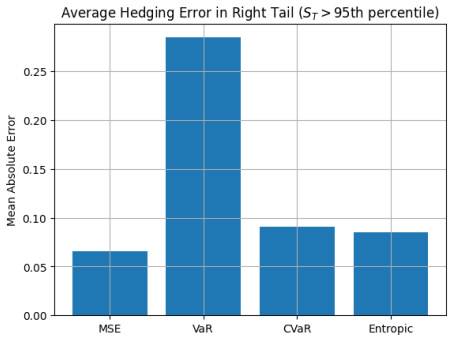
As expected, the model trained with VaR shows the highest average hedging error and the largest frequency of large mistakes. The main weakness of VaR explains this: since VaR only cares about controlling risk up to the 95th quantile, it completely ignores how bad the losses get beyond that point. With heavy-tailed distributions, this leads to poor hedging performance when extreme prices occur.

MSE, CVaR, and entropic risk all achieve significantly lower average errors. However, an interesting observation is that MSE performs the best among all models in terms of these figures despite not being a risk-sensitive loss. This can be explained by the structure of heavy-tailed data: although extreme events occur more often than in a normal distribution, they are still relatively rare compared to the majority of the data. MSE minimizes the overall average error across all scenarios. Therefore, the model learns a hedging strategy that works well for typical outcomes while still giving reasonable results in the tail. By fitting the general payoff structure smoothly, without focusing too much on either normal or extreme outcomes, MSE avoids severe

hedging mistakes even in rare events.

CVaR and entropic risk perform strongly but are slightly worse than MSE in this specific metric. CVaR explicitly targets the worst-case outcomes beyond the 95th quantile threshold, which makes it very cautious in extreme scenarios. However, this focus can make the model slightly less precise for moderate tail outcomes that are not as severe as the worst cases. As a result, while CVaR prevents extreme errors, it may tolerate somewhat larger errors on moderately large price movements, increasing the average error slightly.

Entropic risk, similarly, emphasizes penalizing large losses but does so in a smooth and exponential way rather than sharply focusing on a fixed threshold like CVaR. This allows the entropic model to balance caution across the entire distribution, not just the worst tail. However, because it spreads its attention more evenly than CVaR, it does not specialize fully in protecting against the largest values. Thus, while it avoids large mistakes efficiently, its average hedging error remains slightly higher than that of MSE.



Loss	Frequency
MSE	0.30%
VaR	14.50%
CVaR	0.60%
Entropic	1.30%

Figure 6.13: Average hedging error (left) and frequency (right) of large errors (> 0.5) in the right tail for different loss functions.

Figure 6.14 shows the distribution of the absolute hedging errors for the largest asset prices, namely those beyond the 95th quantile of the test data. The left plot shows the density directly for a more intuitive understanding, while the right plot uses a logarithmic scale to highlight small changes.

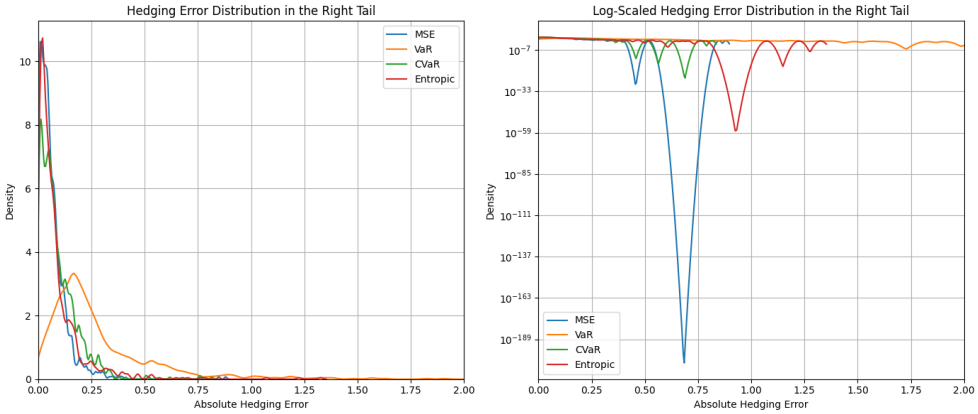


Figure 6.14: Kernel density estimates of absolute hedging errors in the right tail (left) and log-scaled densities (right) for different loss functions.

The KDE plots for MSE, CVaR and entropic risk have sharp peaks near zero, so most hedging errors are small even when the prices are extreme. This matches the theory: MSE minimizes average errors, while CVaR and entropic risk specifically focus on penalizing large losses, keeping

big errors rare.

VaR performs the worst: its curve is shifted toward larger errors, and it shows a much flatter decay. This happens because VaR does not penalize mistakes beyond the fixed quantile, so it allows large hedging errors in the extreme right tail.

The log-scaled plot shows that MSE, CVaR, and entropic densities drop off sharply with increasing error, while VaR remains relatively flat, confirming that VaR fails to control large tail losses.

The last tool we use to evaluate the performance of our deep hedging models is the Sharpe ratio. We state the definition as in Föllmer and Schied (2004).

Definition 6.3.1 (Sharpe Ratio) *Fix the probability space* $(\Omega, \mathcal{F}, \mathbb{P})$. *Suppose an investment has random payoff* $X \in L^2(\Omega, \mathcal{F}, \mathbb{P})$ *with expected value* $\mathbb{E}[X]$ *and standard deviation* $\sigma(X) > 0$. *The Sharpe ratio is defined as*

$$\textit{Sharpe Ratio} = \frac{\mathbb{E}[X]}{\sigma(X)}.$$

Here, X *typically represents the discounted final value of the portfolio.*

The Sharpe ratio measures how much excess return an investment earns for each unit of risk it takes. A higher Sharpe ratio means that the investment offers a better trade-off between return and risk: it earns more return per unit of risk. A lower Sharpe ratio indicates either low returns, high volatility, or both. Figure 6.15 shows the Sharpe ratios of the final profit-and-loss (PnL) from the deep hedging models trained under different loss functions.

Loss	Sharpe Ratio
MSE	0.4283
VaR	0.0591
CVaR	0.4561
Entropic	0.4182

Figure 6.15: Sharpe ratios of hedging PnL under different loss functions

CVaR and MSE achieve the highest Sharpe ratios, creating the most stable and rewarding hedging strategies relative to the risk taken. Entropic risk is not far behind, showing that it also manages risk well, just a little less efficiently. On the other hand, the VaR-trained model performs very poorly, with a Sharpe ratio close to zero. This means its profits are very unstable compared to their size, which makes sense because, as we saw earlier, VaR ignores what happens in extreme scenarios, which is a big problem when markets have heavy tails.

This shows that risk-aware loss functions like CVaR and entropic risk help build more reliable hedging strategies. Interestingly, MSE still does surprisingly well by aiming to keep the average error low across all outcomes, even without explicitly focusing on risk.

6.3.2.4 Right Tail Error Sensitivity to Tail Heaviness

In the above analysis, we reviewed how the four loss functions respond to tail risk. In order to understand how risk measures respond when the tail heaviness changes, we focus on the risk measure known for this limitation, VaR, and the risk measure designed to address the problem, CVaR.

We generate a hundred datasets using the Merton jump-diffusion model. Each dataset contains 10^5 price paths and is generated using different jump intensities and jump size distributions so that we have a range of heavy-tailed scenarios. For each dataset, we estimate the Hill index for the right tail and train two deep hedging models: one using VaR and one using CVaR as the loss function. We then find the mean hedging error in the right tail in the same way as in Figure 6.13 (left), focusing on the errors when the terminal underlying price exceeds the 95th percentile.

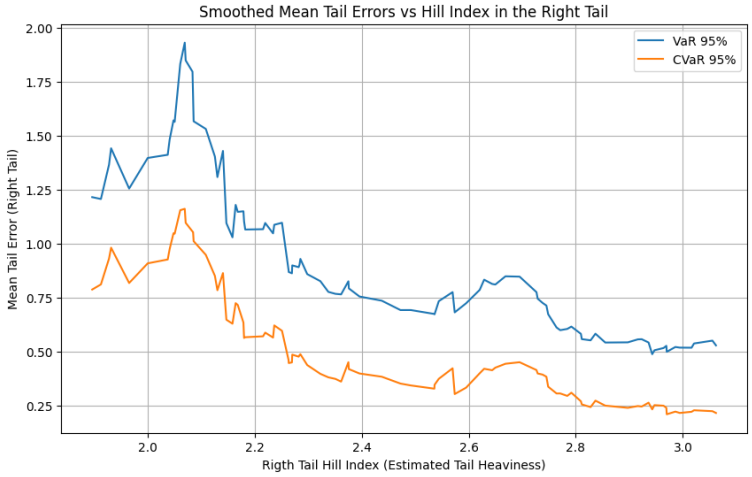


Figure 6.16: Mean right tail hedging errors (for S_T above the 95th percentile) plotted against the right tail Hill index across 100 datasets with varying right tail heaviness.

The results, shown in Figure 6.16, show that the limitation of Value-at-Risk of underestimating tail risk is consistent. Indeed, training with VaR results in higher tail errors than training with CVaR for all levels of tail heaviness. On average, VaR models have a right tail error of 0.9210, while CVaR models have a lower mean of 0.5135. The average difference between the right tail errors for VaR and CVaR models is approximately 0.4076.

As the right-tail Hill index decreases, indicating heavier tails, the right-tail mean error of models trained with VaR increases significantly, reinforcing that VaR does not account for the size of losses beyond the 95th quantile. Interestingly, the plots of the mean error produced by VaR and CVaR track each other in shape. Both models are affected by the same asset price data, so when the tail becomes heavier, both models struggle more.

This section shows that in markets with heavy tails, it is not enough to focus only on typical outcomes. Risk-aware loss functions like CVaR and entropic risk create more stable and effective hedging strategies by accounting for extreme events. In contrast, VaR struggles because it ignores what happens beyond a fixed quantile, leading to instability. Even though MSE does not directly target risk, it still performs well by smoothing out errors across all scenarios.

6.4 Conclusion

We have come a long way since the 1970s and the birth of the Black-Scholes model, which is often credited for shaping mathematical finance as we know it. Despite its elegance, the Black-Scholes relies on assumptions which rarely hold in practice. In the decades since, researchers have attempted to tackle the challenge of hedging in real-world markets, where replication often fails, leading to model risk and uncertainty.

At the same time, we have recently seen a surge of interest in machine learning and neural networks, both in academic circles and the financial industry. With this in mind, this dissertation explores how deep learning can provide a modern solution to classical problems such as pricing and hedging. We showed that not only to price derivatives but also to learn dynamic hedging strategies directly from data, without depending on the existence of an equivalent martingale measure. This is more than a computational shortcut. As shown in Subsection 4.5.3, the dual representation of convex risk measures lets deep hedging minimise risk over a family of probability measures, avoiding the need for a fixed equivalent martingale measure.

The experiments demonstrate several important insights. First, deep hedging models trained

under different loss functions can learn hedging strategies with varying accuracy and risk sensitivity. In stable and normal markets, even simple loss functions perform well, however, when extreme losses come into play, the benefits of risk-aware learning become evident. Entropic risk and Conditional Value-at-Risk, in particular, help the models pay more attention to tail risks, leading to more robust hedging strategies.

At the same time, the results highlight important limitations. While the deep hedging framework is model-free, it is not entirely free of modelling choices. In practice, the neural network architecture plays a vital role in shaping the hedging strategies learned by the models. The choice of loss function also significantly impacts the convergence speed, efficiency, and accuracy of the models.

We could build on this and further refine the models in several ways. For instance, we can apply deep hedging to more complex markets, like those with illiquidity or a large number of assets, developing guidelines for modelling choices that allow the models to learn a specific type of market the best. We could also experiment with designing activation functions tailored to financial data, potentially making the learning more stable and efficient.

While deep hedging is a powerful and flexible approach to pricing and hedging, there remain challenges. With machine learning growing rapidly, including improvements in model interpretability and efficiency, we can expect exciting new methods to address these challenges and improve the performance and reliability of deep hedging models.

Bibliography

Akyıldırım, E. and Soner, H. M. (2014). A brief history of mathematics in finance, *Borsa Istanbul Review* **14**(1).

Artzner, P., Delbaen, F., Eber, J.-M. and Heath, D. (1999). Coherent measures of risk, *Mathematical Finance* **9**(3).

Ben-Tal, A. and Teboulle, M. (2007). An old-new concept of convex risk measures: The optimized certainty equivalent, *Mathematical Finance* **17**(3).

Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities, *Journal of Political Economy* **81**(3).

Buehler, H., Gonon, L., Teichmann, J. and Wood, B. (2019). Deep hedging, *Quantitative Finance* **19**(8).

Castillo, R. E. and Rafeiro, H. (2016). *An Introductory Course in Lebesgue Spaces*, Springer Publishing.

Cesa, M. (2017). A brief history of quantitative finance, *Probability, Uncertainty and Quantitative Risk* **2**(1).

Cont, R. (2006). Model uncertainty and its impact on the pricing of derivative instruments, *Mathematical Finance* **16**(3).

del Rio, R., Franco, A. and Lara, J. (2017). A direct proof of f. riesz representation theorem.

Durett, R. (2019). *Probability: Theory and Examples*, Cambridge University Press.

Friedman, A. (1982). *Foundations of Modern Analysis*, Dover Publications.

Föllmer, H. and Schied, A. (2002). Convex measures of risk and trading constraints, *Finance and Stochastics* **6**(4).

Föllmer, H. and Schied, A. (2004). *Stochastic Finance. An Introduction in Discrete Time*, De Gruyter.

Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*, MIT Press.

Higham, C. F. and Higham, D. J. (2019). Deep learning: An introduction for applied mathematicians, *SIAM Review* **61**(4).

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks, *Neural Networks* **4**(2).

Hull, J. C. (2021). *Options, Futures, and Other Derivatives*, Pearson Higher Ed.

Hutchinson, J. M., Lo, A. W. and Poggio, T. (1994). A nonparametric approach to pricing and hedging derivative securities via learning networks, *The Journal of Finance* **49**(3).

Itô, K. (1951). *On Stochastic Differential Equations*, American Mathematical Society.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

McNeil, A. J., Frey, R. and Embrechts, P. (2016). *Quantitative Risk Management: Concepts, Techniques and Tools*, Princeton University Press.

Merton, R. C. (1976). Option pricing when underlying stock returns are discontinuous, *Journal of Financial Economics* **3**(1).

Pricing and Hedging Using Neural Networks (2025). `https://github.com/ru-ulians/Pricing-and-Hedging-Using-Neural-Networks`.

Ruf, J. and Wang, W. (2019). Neural networks for option pricing and hedging: A literature review, *Journal of Computational Finance* .

Wilmott, P., Howison, S. and Dewynne, J. (1995). *The Mathematics of Financial Derivatives: A Student Introduction*, Cambridge University Press.