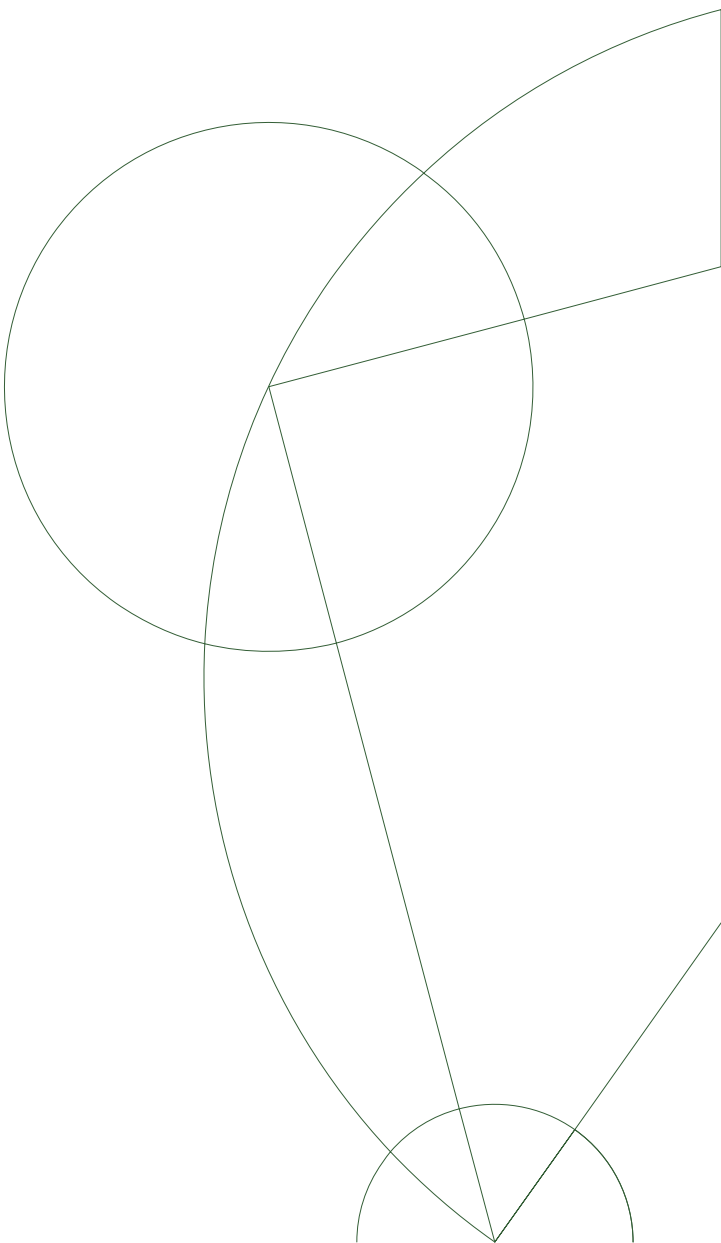
UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

A Deep Study of Deep Hedging

Tobias Cramer Pedersen



Master's Thesis in Mathematics-Economics
Department of Mathematical Sciences
Advisor: Rolf Poulsen

31. May 2021

Abstract

We investigate recent advances in the use of statistical learning methods for risk optimal hedging of derivative books. The portfolio strategies are approximated by parametric families such as neural networks and subsequently trained on sample paths of the underlying state process. As such, the methods are applicable for general derivative books in market models with frictions such as transaction costs and time discretisation.

Particular attention is paid towards constructing rigorous tests of performance in multivariate environments. It is verified that the algorithms work well for univariate settings producing sensible solutions within a short time frame. We further find that, if training times are kept short, the performance deteriorates as the dimension increases. A novel approach, utilising sample paths to compute sensitivities instead, is proposed to remedy the issue.

We test the applicability of the algorithms for hedging path dependent derivatives. In particular, the idea of utilising memory in neural networks is of interest. It allows us to create an algorithm for hedging general path dependent derivatives without having to explicitly specify what information is relevant throughout the trajectory of the sample path. We find that the networks are adept at learning to memorise the necessary information as long as the architecture is chosen appropriately.

Contents

Chapter 1

Introduction

Statistical learning lies at the core of at least three milestones within computational finance. The first milestone is [1] in which a neural network is trained to approximate the price function of the SABR model from [2]. The approximation is orders of magnitude faster and more accurate than the original asymptotic formula. A notable extension of the paper is [3] in which a neural network is trained to replicate the entire volatility surface of the Rough Bergomi model [4] allowing for calibration of the otherwise computationally expensive model.

The second milestone is [5], in which a framework is established allowing for training statistical learning models in the context of hedging financial derivatives. The models are trained on simulations of the sample paths of the underlying instruments and the data sets can therefore be synthetically generated on demand. This allows for approximating risk optimal portfolio strategies in hitherto unsolvable models as well as incorporate market inefficiencies such as transaction costs and time discretisation.

The third milestone is the development of *differential regularisation* in [6] which allows for fast approximation of prices *and* sensitivities for arbitrary derivative books of arbitrary models. The paper utilises neural networks for approximation trained only on a few thousand sample paths.

The thesis investigates the latter two milestones in the context of hedging financial derivative *books* - that is, derivative portfolios consisting of multiple underlying instruments. With the introduction of a multivariate input dimension, the problems explode in complexity. Therefore, it it important to construct as rigorous a testing framework as possible, so as not to erroneously conclude superior performance. The thesis attempts to do just that - and along the way, develop new methods inspired by the original papers.

The structure of the thesis is as follows. In chapters 2 and 3, the necessary theoretical framework for hedging liabilities and training statistical learning models is presented. In chapter 4, the tests conducted in the paper [5] is extended in several ways, most notably with the addition of multivariate derivative books and an arguably more rigorous utilisation of benchmarks. In chapter 5, the framework from [5] is combined with [6] to develop an equally general hedging algorithm which is faster to train, performs better in multivariate environments, and produces interpretable outputs. In chapter 6, inspired by findings from [3], the hedging framework is tested in the context of hedging jump risk. And finally in chapter 7, an interesting idea from [5] is investigated concerning the concept of *memory* in neural networks to formulate a general hedging algorithm for path dependent derivatives.

The code base written for the thesis spans over seven thousand lines. The code is located at

`https://github.com/tcpedersen/deephedging`

Chapter 2

The General Theory of Hedging Liabilities

A liability is a net short position in a combination of financial instruments. The liability naturally imposes a certain degree of risk upon the holder from which it is often beneficial to protect oneself. Exactly what constitutes adequate protection and how one achieves it is the primary focus of this chapter.

2.1 Attainability of Derivatives in Frictionless Markets

The theory of hedging liabilities in continuous time is well known. This chapter is a short reminder of the main theory and mostly serves the purpose of setting up notation for later chapters. More general results than the ones presented here exist, but these encompass far beyond the models treated in this thesis. The overall framework follows that of [7].

Consider a market with a finite time horizon T defined on the probability space $(\Omega, \mathcal{F}, P)$ along with a filtration $\mathbf{F} = \{\mathcal{F}_t | 0 \leqslant t \leqslant T\}$. On the probability space the following four $\mathbf{F}$-adapted processes are defined:

1. a k-dimensional Brownian Motion W_t^P,
2. a d-dimensional stochastic process α_t,
3. a $d \times k$-dimensional stochastic process $\mathbf{\Sigma}_t$,
4. and a real valued stochastic process r_t.

The market consists of the tradable instruments process $\mathbf{S}_t \in \mathbb{R}^d$ and another real-valued numeraire process B_t. The numeraire process is often referred to as the *bank account*. The instruments and numeraire are solutions (assuming they exist) to the following stochastic differential equations[1]

$$\begin{aligned} d\mathbf{S}_t &= D\left[\mathbf{S}_t\right] \alpha_t dt + D\left[\mathbf{S}_t\right] \mathbf{\Sigma}_t dW_t^P, \\ dB_t &= r_t B_t dt. \end{aligned}$$

[1] The diagonal operator: $D\left[\mathbf{x}\right]_{ij} = \begin{cases} \mathbf{x}_i & i = j \\ 0 & i \neq j \end{cases}$.

Note that the presence of $D[\mathbf{S}_t]$ has to influence on the generality of the results as α_t and $\mathbf{\Sigma}_t$ are allowed to depend on $\mathbf{S}_t$. If α_t, $\mathbf{\Sigma}_t$, and r_t are functions of time, the instruments, and the numeraire, then exact conditions are known for solutions to exist [8, ch. 5.2]. See appendix C for additional details.

It is known that such an economy admits no arbitrage[2] if and only if there exists an equivalent probability measure Q such that the normalised processes $\mathbf{S}_t/B_t$ is a (local) martingale under Q [7, ch. 10]. For some fixed martingale measure, the only price process H_t of an $\mathcal{F}_T$-measurable contingent claim Z consistent with maintaining no arbitrage is

$$\frac{H_t}{B_t} \triangleq E^Q\left[\frac{Z}{B_T}\middle|\mathcal{F}_t\right], \tag{2.1.1}$$

as such a definition ensures that the measure Q is also an equivalent martingale measure for the extended market $(\mathbf{S}_t, B_t, H_t)$. As the probability measure Q is not necessarily unique, there might exist multiple potential price processes.

Let $\mathbf{a}_t \in \mathbb{R}^{1\times d}$ and $b_t \in \mathbb{R}$ respectively denote the $\mathbf{F}$-adapted processes representing positions in a portfolio consisting of $\mathbf{S}_t$ and B_t. The associated *value process* of the portfolio is naturally just the weighted sum of the holdings,

$$V_t \triangleq \mathbf{a}_t\mathbf{S}_t + b_tB_t. \tag{2.1.2}$$

The portfolio strategy $(\mathbf{a}_t, b_t)$ is self-financing if the associated value process is a solution to the stochastic differential equation [7, ch. 10.3]

$$d\left(\frac{V_t}{B_t}\right) = \mathbf{a}_t d\left(\frac{\mathbf{S}_t}{B_t}\right). \tag{2.1.3}$$

Let V_0 be some initial cash-injection into the portfolio. From equation (2.1.3) it follows (by definition) that if the portfolio strategy is self-financing, then the terminal value is

$$\frac{V_T}{B_T} = \frac{V_0}{B_0} + \int_0^T \mathbf{a}_t d\left(\frac{\mathbf{S}_t}{B_t}\right), \tag{2.1.4}$$

Thus, if the self-financing condition is superimposed on the portfolio strategy, then the exact specification of b_t is irrelevant to the terminal value process.

An $\mathcal{F}_T$-measurable contingent claim Z is said to be attainable if there exists a self-financing portfolio strategy such that the terminal value of the value process and claim are identical P-a.s. Naturally, if Z is interpreted as a liability, it can be perfectly protected by following the attaining portfolio strategy. The exact *cost* of protecting the liability is the initial cash-injection V_0 which must coincide with the arbitrage-free price of the claim H_0, as we otherwise will have found an arbitrage opportunity. In fact, the value process of the portfolio and the arbitrage-free value of the liability must be equal P-a.s.

An economy is said to be complete if all contingent claims are attainable. It can be shown that if there exists an equivalent martingale measure, then the market is complete if and only if the measure is unique [7, ch. 10].

The conditions for the model being arbitrage-free and complete can be simplified if we let the filtration be generated solely by the Brownian motion. In this case, the model is arbitrage-free if the volatility matrix $\mathbf{\Sigma}_t$ has rank d. If the volatility matrix is invertible, then the economy is complete [7, ch. 14]. The drift term of the instruments under the martingale measure is the short-rate r_t. The diffusion term is unchanged. Note that these results rely on the simplistic form of the numeraire.

[2] Technically, *no free lunch with vanishing risk*. Also, the processes must be locally bounded.

The setting does not exclude the presence of other non-tradable processes as long as they are solely driven by the Brownian motions. The notion of other processes is usually abstracted away by claiming the existence of some *state process* $\mathbf{X}_t$ which at least contains the tradable instruments and numeraire. However, the process can also encompass a stochastic volatility or other observable market quantities. The dynamics of these are dependent on the martingale measure [7, ch. 14].

2.2 Measuring the Risk of a Financial Position

All claims are attainable in a complete market [7, ch. 8]. Under such an assumption, any financial liability can be protected perfectly and the notion of risk is therefore irrelevant. Once various inefficiencies are introduced, the quantity of risk associated with a particular protective (portfolio) strategy must be captured. To achieve this purpose, the concept of a risk measure is introduced.

Throughout this chapter the existence of a probability space $(\Omega, \mathcal{F}, P)$ is assumed. Let the set of all real valued random variables with Ω as domain be denoted as $\mathbb{X}$. An element of $\mathbb{X}$ is interpreted as a financial position.

2.2.1 Convex Risk Measures

We have certain intuitive presuppositions about what properties risk measures should adhere to. The measures must however also be theoretically tractable as they shall be used for modelling purposes. In this thesis it is of particular importance that we at least restrict ourselves to the class of convex risk measures. The measures are based on the philosophy that the risk of a position is to be interpreted as the minimal amount of capital one needs to add to a financial position in order to find the position *acceptable*.

Definition 2.2.1. *A convex risk measure is a functional $\rho : \mathbb{X} \longrightarrow \mathbb{R}$ which for all $X, X_1, X_2 \in \mathbb{X}$ satisfies the following three properties*

1. *If $X_1 \geqslant X_2$ P-a.s. then $\rho(X_1) \leqslant \rho(X_2)$* *(monotonically decreasing)*
2. *For all $\lambda \in (0,1)$, $\rho(\lambda X_1 + (1-\lambda)X_2) \leqslant \lambda\rho(X_1) + (1-\lambda)\rho(X_2)$* *(convex)*
3. *For all $c \in \mathbb{R}$, $\rho(X + c) = \rho(X) - c$* *(cash-invariant)*

The interpretation of the three conditions is straightforward: monotonicity implies that a more favourable position carries less risk, convexity implies that diversification reduces risk, and cash-invariance implies that adding cash to a position reduces the risk by an equal amount.

If $\rho(0) = 0$ then the risk measure is said to be *normalised*. For a normalised risk-measure, it is natural to consider the *acceptance set* as the set of financial positions with a negative risk,

$$\mathbb{A}_\rho \triangleq \{X \in \mathbb{X} : \rho(X) \leqslant 0\}.$$

This definition naturally ties into the aforementioned idea of risk measures as acceptable financial positions. In fact, the risk measure can be recovered from the acceptance set by the following relation [9]

$$\rho(X) = \inf\{c \in \mathbb{R} : X + c \in \mathbb{A}_\rho\}.$$

The relation above allows one to interpret the risk measure as exactly the minimal amount of capital one needs to add to a position to make it acceptable.

2.2.2 Optimised Certainty Equivalence

The class of convex risk measures is still too broad to be theoretically tractable. This chapter will introduce the particularly useful subclass of convex risk measures known as the class of *optimised certainty equivalence (OCE) measures*. An instance of the class is defined through a continuous, non-decreasing, and convex *loss function* $\ell : \mathbb{R} \longrightarrow \mathbb{R}$,

$$\rho(X) \triangleq \inf_{w \in \mathbb{R}} E\left[w + \ell\left(-X - w\right)\right].$$

In this thesis the family of OCE risk measures will be the primary focus, as they seamlessly integrate into the overall numerical schemes. The class is surprisingly broad and spans the following popular risk measures.

The expected shortfall is typically interpreted as the expected loss of a position conditional on the loss being within the α quantile of worst possible losses. At level α the measure can be recovered from the loss function

$$\ell(x) = \frac{1}{1-\alpha} \max(x, 0).$$

Even though the function is technically only piece-wise continuous, it has no influence as the loss function only appears inside an expectation. If differentiation is allowed under the expectation, then the first order condition with respect to w is

$$P\left(-X \geqslant w\right) = 1 - \alpha.$$

Thus, w represents the value-at-risk for the loss $-X$. The measure is normalised.

Alternatively, the more traditional mean-variance risk measure can be utilised. However, this is *not* an OCE risk measure. The mean-variance risk measure under constant relative risk aversion γ can be recovered by setting

$$\ell(x) = x + \frac{\gamma}{2} x^2.$$

The loss function is not non-decreasing, so the associated risk measure is not monotone. The optimisation problem can be solved explicitly yielding $w = -E[X]$, which translates into the well-known mean-variance risk measure

$$\rho(X) = \frac{\gamma}{2} V\left[X\right] - E\left[X\right].$$

The measure is normalised. While it does not satisfy the property of monotonicity, it is convex and cash-invariant.

2.3 Risk Optimal Hedging in Markets with Frictions

The idea that liabilities can be perfectly hedged is of course highly stylised. Markets are subject to a myriad of inefficiencies which prevents the agent from perfectly protecting him- or herself against a liability. The possession of a liability is therefore subject to risk and the central question is if this risk is acceptable from a risk measure perspective.

This chapter presents the problem of hedging a contingent claim under a convex risk measure. While the presentation follows [5], the entire theory is reformulated in continuous time. The original paper also does not include the non-trivial extension to generic numeraire assets as it implicitly assumes a constant interest of zero.

2.3.1 A Brief Presentation of Frictions

All trades incur *transaction costs*. The costs usually scale in proportion to the magnitude of the trade, but can also be a fixed fee from entering a trade. If trading options, the cost might depend on the sensitivity of the option to market risk such as volatility. In this thesis only proportional transaction costs are considered, but the theory easily extends to much more complicated fee structures. For now, the cumulative transaction costs incurred until time t are modelled by an $\mathbf{F}$-adapted stochastic process $C_t \in \mathbb{R}$. It is assumed that the costs do not depend on b_t nor V_0. The cumulative costs are of course zero at time zero.

While transaction costs can be considered as indirect constraints on how trading can be conducted, trades can also be subject to direct constraints such as no short-selling or liquidity shortage. Therefore we restrict the set of portfolio strategies at time t to the set $\mathcal{A}_t$ of allowed portfolio strategies.

Despite what the theory in chapter 2.1 claims, it is impossible to trade in continuous time. An even more unpleasant realisation is that the state process $\mathbf{X}_t$ *does* in fact evolve in continuous time. As the portfolio strategy cannot continuously adapt to the changing market, it is in general not possible, even without any other frictions, to perfectly replicate a claim.

Finally, the claim might not be attainable at all. Attainability of a claim is only guaranteed in complete markets, but many important market models not covered in chapter 2.1 includes various market risks which cannot be hedged. The textbook example is subjecting the instrument process to *jump risk*, and it is the exact topic treated in chapter 6.

2.3.2 The Liability Owner's Problem

Imagine an agent holding a real valued $\mathcal{F}_T$-measurable contingent claim X measured in terms of the numeraire B_T. The objective of the agent is to construct a portfolio strategy $\mathbf{a}_t \in \mathcal{A}_t$ for all $t \in [0, T]$ such that the (discounted) terminal wealth at time T is subject to as little risk as possible. The risk is measured through an a priori chosen risk measure ρ, which measures the risk *in terms of the numeraire*. Abstractly, the agent solves the minimisation problem

$$\inf_{\mathbf{a}_t \in \mathcal{A}_t} \rho\left(\frac{V_T - C_T}{B_T} + X\right). \tag{2.3.1}$$

While the notation implies otherwise, the minimisation is meant as finding the portfolio strategy at all points in time from time 0 to T. As the risk measure is cash-invariant, the problem can be reformulated as,

$$\inf_{\mathbf{a}_t \in \mathcal{A}_t} \rho\left(\int_0^T \mathbf{a}_t d\left(\frac{\mathbf{S}_t}{B_t}\right) - \frac{C_T}{B_T} + X\right) - \frac{V_0}{B_0}.$$

An important consequence is that the optimal solution is independent of the initial cash-injection as the last term is constant. In fact, as we shall see in chapter 2.3.3, a fair initial cash injection can be *inferred* from the optimal solution. These considerations motivates defining *the liability holder's problem* as

$$\pi(X) \triangleq \inf_{\mathbf{a}_t \in \mathcal{A}_t} \rho\left(\int_0^T \mathbf{a}_t d\left(\frac{\mathbf{S}_t}{B_t}\right) - \frac{C_T}{B_T} + X\right). \tag{2.3.2}$$

Assuming that the transaction costs and trading restraints are convex in $\mathbf{a}_t$, it is possible to show that π is also a convex risk measure [5]. This is an important property, which is assumed to hold throughout the thesis.

2.3.3 Indifference Pricing

Recall the discussion in chapter 2.2 where we showed that ρ can be interpreted as the minimal amount of capital one must add to a position in order to make it acceptable. This interpretation allows for a definition of the price of a contingent claim in terms of the risk measure. It is assumed throughout this chapter that there are no trading restraints on $\mathcal{A}_t$ for all t.

Assume the agent sells a contingent claim Z at time zero at a price (in terms of the numeraire) of V_0/B_0. The agent is then able to use the income as a cash injection and hedge according to equation (2.3.2). It is clear from equation (2.3.1) that the risk of such a trade is

$$\pi\left(-\frac{Z}{B_T}\right) - \frac{V_0}{B_0}.$$

In other words, the trade is within the acceptance set of the agent if the above quantity is negative. However the agent might also consider not accepting the trade and trading freely in the market without any liabilities. The latter approach incurs a risk of $\pi(0)$ and so the *indifference price* is therefore defined as the minimal price with which the agent is indifferent between selling Z and trading liability free,

$$\frac{p\left(Z\right)}{B_0} \triangleq \pi\left(-\frac{Z}{B_T}\right) - \pi(0).$$

Three results will now be presented which justify the definition and establish important insights that will be used later. The results are inspired by proofs in [5], but are more precise due to the continuous time reformulations of all results in this chapter.

Recall that a contingent claim Z is attainable if there exists a real number V_0 and portfolio strategy $\mathbf{a}_t^*$ such that P-a.s.

$$\frac{Z}{B_T} = \frac{V_0}{B_0} + \int_0^T \mathbf{a}_t^* d\left(\frac{\mathbf{S}_t}{B_t}\right).$$

A justification for the definition of indifference pricing lies in the following result. In particular, the result shows that if the market is complete and arbitrage-free, then the indifference price under no transaction costs corresponds to the unique arbitrage-free price of the claim.

Proposition 2.3.3. *Assume that the claim Z is attainable using V_0 and $\mathbf{a}_t^*$. If there are no transaction costs, then $p(Z) = V_0$.*

Proof. By definition of π and using the attainability of Z,

$$\pi\left(-\frac{Z}{B_T}\right) = \inf_{\mathbf{a_t}} \rho\left(\int_0^T \mathbf{a}_t d\left(\frac{\mathbf{S}_t}{B_t}\right) - \frac{Z}{B_T}\right) = \inf_{\mathbf{a}_t} \rho\left(\int_0^T \left(\mathbf{a}_t - \mathbf{a}_t^*\right) d\left(\frac{\mathbf{S}_t}{B_t}\right)\right) + \frac{V_0}{B_0}.$$

As $\mathbf{a}_t - \mathbf{a}_t^*$ is simply a portfolio strategy, the above formulation implies $\pi(-Z/B_T) = \pi(0) + V_0/B_0$. □

The next proposition shows that if risk is measured relative to the risk-neutral dynamics, then the arbitrage-free price systematically underestimates the indifference price in the presence of transaction costs.

Proposition 2.3.4. *Let Z be a contingent claim. If the market is arbitrage-free and ρ is an OCE risk measure, where the expectation is taken with respect to the equivalent martingale measure, then $\pi(0) = \rho(0)$ and $p(Z) \geqslant H_0$.*

Proof. As $\mathbf{S}_t/B_t$ is a martingale under Q, it holds that the expected value of any portfolio strategy is zero [7, ch. 4.3],

$$E^Q\left[\int_0^T \mathbf{a}_t d\left(\frac{\mathbf{S}_t}{B_t}\right)\right] = E^Q\left[\int_0^T \mathbf{a}_t D\left[\frac{\mathbf{S}_t}{B_t}\right]\mathbf{\Sigma}_t dW_t^Q\right] = 0.$$

It always holds that $\pi(0) \leqslant \rho(0)$ as $\mathbf{a}_t = \mathbf{0}$ is a viable portfolio strategy. As ℓ is convex, non-decreasing and transaction costs are positive, it follows from Jensen's inequality that

$$\begin{aligned}
\pi\left(-\frac{Z}{B_T}\right) &= \inf_{w\in\mathbb{R}}\inf_{\mathbf{a}_t}\left\{w + E^Q\left[\ell\left(-\int_0^T \mathbf{a}_t d\left(\frac{\mathbf{S}_t}{B_t}\right) + \frac{C_T}{B_T} + \frac{Z}{B_T} - w\right)\right]\right\} \\
&\geqslant \inf_{w\in\mathbb{R}}\inf_{\mathbf{a}_t}\left\{w + \ell\left(E^Q\left[-\int_0^T \mathbf{a}_t d\left(\frac{\mathbf{S}_t}{B_t}\right) + \frac{C_T}{B_T} + \frac{Z}{B_T} - w\right]\right)\right\} \\
&\geqslant \inf_{w\in\mathbb{R}}\left\{w + \ell\left(E^Q\left[\frac{Z}{B_T} - w\right]\right)\right\} \\
&= \rho\left(-E^Q\left[\frac{Z}{B_T}\right]\right) \\
&= \rho(0) + E^Q\left[\frac{Z}{B_T}\right].
\end{aligned}$$

Letting $Z = 0$ yields $\pi(0) \geqslant \rho(0)$ which proves the first claim. Rearranging the terms yields the second claim,

$$\pi\left(-\frac{Z}{B_T}\right) - \pi(0) \geqslant E^Q\left[\frac{Z}{B_T}\right],$$

as we just proved that $\pi(0) = \rho(0)$. □

The final proposition shows that if the market is sufficiently efficient and risk is measured in risk-neutral terms, then perfectly replicating the contingent claim is also efficient from a risk measure perspective. The result is important as it allows for a way to test whether or not numerical hedging schemes are sufficiently capable of finding optimal strategies.

Proposition 2.3.5. *Assume that the claim Z is attainable using V_0 and $\mathbf{a}_t^*$. If the market is arbitrage-free, there are no transaction costs, and ρ is an OCE risk measure, where the expectation is taken with respect to the equivalent martingale measure, then the liability owner's problem is solved by $\mathbf{a}_t^*$.*

Proof. Using $\mathbf{a}_t^*$ as portfolio strategy yields a risk of

$$\rho\left(\int_0^T \mathbf{a}_t^* d\left(\frac{\mathbf{S}_t}{B_t}\right) - \frac{Z}{B_T}\right) = \rho(0) + \frac{V_0}{B_0}.$$

In the proof of proposition 2.3.3 it was shown that $\pi\left(-Z/B_T\right) = \pi(0) + V_0/B_0$. From proposition 2.3.4 it follows that $\pi(0) = \rho(0)$. Thus $\mathbf{a}_t^*$ attains the minimum possible risk, which is equivalent to solving the liability owner's problem. □

Chapter 3

Statistical Learning Theory

The theory of statistical learning is the workhorse behind all numerical results in this thesis. The field is broad, fast moving and development is largely driven by real world problems. The goal of this chapter is to present the reader with the important models, algorithms and enhancements necessary for training the models developed in later chapters.

3.1 Formal Description

The field of statistical learning is a framework concerning the identification of a predictive function h from a hypothesis space $\mathcal{H}$ usually by an application of gradient descent. Let $\mathbf{X}$ and $\mathbf{Y}$ be two respectively d and p dimensional real valued random variables following some unknown joint distribution $p(\mathbf{x}, \mathbf{y})$. The loss associated with a predictor $h : \mathbb{R}^d \longrightarrow \mathbb{R}^p$ is captured by an a priori chosen *loss function* L. Assuming it is well defined, the expected loss can then be defined as

$$\mathcal{I}[h] \triangleq \int\int L(h(\mathbf{x}), \mathbf{y})\, p(\mathbf{x}, \mathbf{y})\, d\mathbf{x} d\mathbf{y}. \tag{3.1.1}$$

The objective is to find the element $h \in \mathcal{H}$ which achieves the smallest possible expected loss. As searching in the space of all functions is infeasible, the hypothesis set is restricted to some *parametric family of functions* h_θ indexed by a parameter vector $\theta \in \Theta$. As analytical evaluation of the expected value is also impossible, a training sample $(\mathbf{x}_i, \mathbf{y}_i)$ for $i = 1, \ldots, m$ drawn from $p(\mathbf{x}, \mathbf{y})$ is collected. An approximate solution to the problem can then be found by minimising the empirical approximation of the expected loss

$$\min_{\theta \in \Theta} J(\theta) \triangleq \min_{\theta \in \Theta} \frac{1}{m} \sum_{i=1}^{m} L(h_\theta(\mathbf{x}_i), \mathbf{y}_i). \tag{3.1.2}$$

Depending on the problem, the above minimisation problem can be analytically solved or numerically approximated. This thesis contains issues relating to both types of problems.

3.1.1 Multivariate Regression

A special case of the statistical learning problem is the multivariate regression setting, which is characterised by the utilisation of the (squared) Euclidean norm between $h(\mathbf{x})$ and $\mathbf{y}$ as loss function. If the hypothesis space is sufficiently broad, the subsequent minimising predictor can be represented as the conditional expected value of $\mathbf{Y}$ given $\mathbf{X}$. To see this, let $(\mathbf{x}_i, \mathbf{y}_i)$ be a collection of independent and

identically distributed samples from $p(\mathbf{x}, \mathbf{y})$. The conditions for the law of large numbers is satisfied, so the loss can be lower bounded (in the limit) as

$$\frac{1}{m}\sum_{i=1}^{m}\left\|h_\theta(\mathbf{x}_i)-\mathbf{y}_i\right\|^2 \xrightarrow{\text{a.s.}} \sum_{j=1}^{p} E\left[h_\theta(\mathbf{X})^{(j)}-\mathbf{Y}^{(j)}\right]^2 \geqslant \sum_{j=1}^{p} E\left[E\left[\mathbf{Y}^{(j)}\middle|\mathbf{X}\right]-\mathbf{Y}^{(j)}\right]^2.$$

Thus for a sufficiently large sample size and assuming the parametric family can represent the conditional expectation, the global minimum corresponds to learning the conditional expectation of $\mathbf{Y}$ given $\mathbf{X}$. This fact is crucial for the theory developed in chapter 5.

3.2 Linear Feature Maps

The most basic parametric family of predictors is the family in which it is assumed that the predictor is linear in the features. Specifically, for a feature map $\xi : \mathbb{R}^d \longrightarrow \mathbb{R}^q$ for some $q \geqslant 1$, the parametric family of linear feature maps, traditionally indexed by $\beta \in \mathbb{R}^{q\times p}$, is defined as

$$h_\beta(\mathbf{x}) = \beta^\top \xi(\mathbf{x}).$$

The feature map ξ is often chosen as some polynomial transformation of the features, but the exact specification does not alter the fact that the predictor depends on the transformed features linearly.

The Stone-Weierstrass theorem famously states that any continuous function defined on a closed interval can be universally approximated by some polynomial. Thus if a polynomial of sufficiently high degree is used as feature transformation, then it is guaranteed that the parametric function can represent the optimal solution to arbitrarily high precision. On paper this is a sound approach and it can work decently in a univariate setting (as we shall see later). The fundamental issue with polynomials is that the number of coefficients is equal to $\left(p^d+1\right)q$ where p denotes the order of the polynomial. This explosion in parameters is due to the fact that *cross terms* must be included to account for *interaction effects* between the inputs.

One possibility is to assume that the ith output of h only depends linearly on the ith element of $\mathbf{x}$, which naturally only works if the input and output dimensions are of equal size. We can then restrict the parameter matrix β to achieve the predictor[1],

$$h_{\theta_0,\theta_1}(\mathbf{x}) = \begin{pmatrix} \beta_{10} & \beta_{11} & 0 & \cdots & 0 \\ \beta_{20} & 0 & \beta_{22} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta_{d0} & 0 & 0 & \cdots & \beta_{dd} \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_d \end{pmatrix} \triangleq \theta_0 + \theta_1 \odot \mathbf{x},$$

for two parameter vectors $\theta_0, \theta_1 \in \mathbb{R}^d$. As opposed to polynomials with cross terms, the number of parameters only grow linearly in the feature dimension. This simple parameterisation turns out to be surprisingly useful for numerical approximation of portfolio strategies.

3.3 Deep Learning

When a statistical learning problem is solved by utilising deep function architectures such as deep- or recurrent neural networks the process is referred to as *deep learning*. While the term deep architecture is rather vague, it often entails some parametric family with a specific recursive structure allowing for *universal approximation* properties and fast computations of gradients.

[1] The operator $\odot$ denotes the Hadamard product which is element-wise multiplication.

The special case of a *deep feedforward network* is much clearer. The network consists of $M \in \mathbb{N}$ layers with the first $M-1$ layers referred to as the hidden layers and the last layer as the output layer. For some a priori chosen number of hidden units $n_1, \ldots, n_l$, the lth hidden layer consists of a weight matrix $\mathbf{W}_l \in \mathbb{R}^{n_{l-1} \times n_l}$, a bias vector $\mathbf{b}_l \in \mathbb{R}^{n_l}$, and a non-linear activation function $g_l : \mathbb{R}^{n_l} \longrightarrow \mathbb{R}^{n_l}$. For an input $\mathbf{a}^{[0]} \triangleq \mathbf{x}$, the output of the network $\mathbf{a}^{[L]} \triangleq h_\theta(\mathbf{x})$ is then recursively defined as

$$\mathbf{a}_l = g_l(\mathbf{z}_l) = g_l\left(\mathbf{W}_l^\top \mathbf{a}_{l-1} + \mathbf{b}_l\right) \in \mathbb{R}^{n_l}, \quad l \in \{1, \ldots, M\}.$$

The parameter vector θ refers to the learnable parameters, which in the case of a deep feedforward network are weight matrices and bias vectors. For a general overview of different activation functions and their heuristics, see [10].

The gradient of $L(h_\theta(\mathbf{x}), \mathbf{y})$ for a deep feedforward neural network is obtained through the backpropagation algorithm. The algorithm is essentially just an application of the chain rule, but with a very important insight which is also used later in chapter 5.4 in a seemingly unrelated application. First, it is clear that the gradient with respect to the weight matrix and bias vector at the lth layer is

$$\nabla_{\mathbf{W}_l} L = \mathbf{a}_{l-1} \nabla_{\mathbf{z}_l} L^\top \quad \text{and} \quad \nabla_{\mathbf{b}_l} L = \nabla_{\mathbf{z}_l} L.$$

It is therefore, in fact, the gradient of the loss function with respect to $\mathbf{z}_l$ which is the true object of interest. Applying the chain rule multiple times yields the backwards equation

$$\begin{aligned}
\nabla_{\mathbf{z}_l} L &= \nabla_{\mathbf{a}_T} L \nabla_{\mathbf{z}_T} \mathbf{a}_T \nabla_{\mathbf{a}_{T-1}} \mathbf{z}_T \nabla_{\mathbf{z}_{T-2}} \mathbf{a}_{T-1} \cdots \nabla_{\mathbf{z_{l+1}}} \mathbf{a}_{l+1} \nabla_{\mathbf{a}_l} \mathbf{z}_{l+1} \nabla_{\mathbf{z}_l} \mathbf{a}_l \\
&= \nabla_{\mathbf{z}_{l+1}} L \nabla_{\mathbf{a}_l} \mathbf{z}_{l+1} \nabla_{\mathbf{z}_l} \mathbf{a}_l \\
&= \nabla_{\mathbf{z}_{l+1}} L \mathbf{W}_{l+1}^\top \odot g_l'(\mathbf{z}_l).
\end{aligned}$$

It is therefore possible to iteratively compute the gradient from a starting value of $\nabla_{\mathbf{z}_L} L$. Importantly, since the recursion is computed backwards, it solely consists of efficient vector-by-matrix products. Had we instead computed it forwards it would have consisted of matrix-by-matrix products. Furthermore, the gradients for *all* the weights and biases are computed within a single *backwards pass*. For a more detailed derivation of the backpropagation algorithm including how to generalise it to arbitrary sample size m, see [11].

A particularly important property of the deep feedforward networks is their universal approximation capability. Due to the network's high capacity for complexity it turns out that they are particularly adept at approximating continuous functions. Many *universal approximation theorems* exist, but we will only need to highlight two classical results.

The first result from [12] states that if the activation function is continuous, non-polynomial, and locally integrable, the set of all one-layer (shallow) networks is dense in the set of all d-dimensional continuous functions.

Another, more complicated result from [13], states that under certain regularity conditions on the activation function including it being $C^n\left(\mathbb{R}^d\right)$, the set of shallow neural networks is dense in $C^n\left(\mathbb{R}^d\right)$ with respect to a function norm which includes distances between derivatives. In other words, the networks can approximate the function *and* the first n derivatives arbitrarily well.

The universal approximation theorems primarily concern the approximate capabilities of shallow neural networks. Intuitively, the representative power of the networks stem from their capacity for complexity. A shallow network solely consists of multiplication and addition, while deeper network architectures includes *composition* of functions. In [14], a strong heuristic argument revolving around this exact phenomena is constructed to show that deeper networks allow for more representative power.

Another very strong case for depth lies in the overwhelming amount of empirical observations stating that deep vastly outperforms shallow architectures [15, ch. 6.4.1]. Shallow networks were likewise included in the early stages of experimentation in this thesis, but were quickly abandoned in favour of deeper architectures.

3.4 Stochastic Gradient Descent

The approximate problem in equation (3.1.2) is fundamentally just a minimisation of a real valued function and the solution is therefore characterised by the gradient being equal to zero. Assuming the gradient of J is available, the solution can be numerically approximated by *stochastic gradient descent*. For a learning rate α and initial guess θ_0, the parameter vector θ is iteratively approximated by the recursive sequence

$$\theta_t \longleftarrow \theta_{t-1} - \alpha \nabla_\theta J.$$

As the empirical gradient is a consistent estimator of the gradient of the expected loss the algorithm will converge to a local minimum for sufficient regulatory conditions on the learning rate [16, ch. 8.3].

One desires m to be large as this might promote good generalisation to samples outside of the training set. However, large m might increase the computational memory load above what is feasible. The fundamental insight in stochastic gradient descent is that if the gradient is computed with respect to a smaller mini-batch of size $k < m$, then the gradient is still a consistent estimator, and thus convergence is still guaranteed. An alternative to naive gradient descent is to split the training set into disjoint samples of size k and iteratively update θ_t for each mini-batch. An entire iteration over the training splits is referred to as an *epoch*.

The Adaptive Moments (Adam) algorithm from [17] is a more sophisticated extension of traditional stochastic gradient descent. For each mini-batch in an epoch, the parameter vector is updated as

$$\theta_t \longleftarrow \theta_{t-1} - \alpha \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}.$$

Here α is the learning rate and $\widehat{m}_t$ and $\widehat{v}_t$ are (bias corrected) exponential moving averages of the first- and second moment of the gradient. Dividing by the second moment ensures that when the variance of the estimate is high, the learning rate is decreased in order to dampen potential oscillations. The decay of the exponential weightings of the moments are controlled by two hyperparameters.

An error was discovered in the convergence proof of Adam, which helped explain why the algorithm had poor performance in some learning tasks with high output dimension [18]. A correction is proposed in the paper.

As for the computation of the gradient of J, if not analytically available, it is computed using *Automatic Adjoint Differentiation (AAD)* [16, ch. 6.5.6]. AAD is an exact numerical scheme for computations of gradients of general *tensor* valued differentiable functions highly influenced by the backpropagation algorithm from chapter 3.3. The Python library `Tensorflow` [19] is used for all numerical computations in this thesis as it provides an efficient AAD scheme.

3.5 Further Improvements of Deep Learning

While all necessary theoretical aspects of deep learning are presented in the previous chapters, blindly applying the methods will surely result in failure. The popularity of deep learning is arguably due to the myriad of optimisations specifically developed to aid in the training of deep architectures. This chapter

will provide an overview of the most important optimisations developed and encompass all utilised in this thesis.

To dampen the amount of information in this chapter the less interesting optimisations: *early stopping* and *cyclical learning rates* are omitted despite them being used in the experiments. For an explanation of early stopping see [16, ch. 7.8] and for cyclical learning rates see the original paper [20].

3.5.1 Parameter Initialisation

The objective function J in equation (3.1.2) is in general not convex when neural networks are utilised as parametric family. This coupled with the large amount of weights is worrisome as the prospect of local minima and saddle points worsens. Some research suggests that as network architectures deepens the number of saddle points far outweighs the number of local minima and that the values at these critical points are nearly identical [21]. This suggests that local minima are of no particular concern when training and could help to explain why deeper networks are often considered more stable in optimisation.

The above suggests that the initial starting point of the gradient descent algorithm θ_0 from chapter 3.4 has no particular influence on training. However, as the heuristic for utilising neural networks is that each unit will represent a different trait of the optimal solution, it seems counterproductive to initialise any of the weights to be identical. A random initialisation of the parameters using a high entropy real valued distribution is therefore the obvious initialisation strategy. If the variance of the distribution is too high, the output will explode in magnitude and if chosen too low the output will vanish [16, ch. 8.4].

The most common parameter initialisation strategies aims at keeping the variance of $\mathbf{z}_l$ constant for all $l \in \{1, \dots, M\}$. Importantly, these strategies are designed *assuming the input follows a standard normal distribution* [22, 23, 24], and it is therefore highly beneficial to aim at constructing a feature set which resembles that.

3.5.2 Preprocessing

The rate of convergence of neural networks is sensitive to the mean and variance of the data. If two measurements in the feature set are of a different variance, errors relating to the larger of the two will dominate in the weight update simply due to the magnitude. The errors will therefore not necessarily be scaled by the features' actual relevance. This coupled with the inherent assumptions in the weight initialisation scheme from chapter 3.5.1 strongly suggests normalising the input and output to zero mean and unit variance. This is sometimes referred to as feature scaling.

A more sophisticated preprocessing step also involves decorrelating the inputs. This is called *input whitening*. In [25], it is argued that whitening inputs aid in keeping the optimal weight configurations less dependent on each other. Below we shall review a slighty more complicated preprocessing step which also includes dimensionality reduction.

If we instead of correlated inputs consider the more extreme case of *linearly dependent* inputs, decorrelation is impossible as no eigenvalue decomposition can be performed. If the inputs in general exhibit colinear behaviour it is intuitively clear that much computational effort will be wasted in training as (close to) linearly dependent columns yields little extra predictive capacity. This leads to the idea of *dimensionality reduction* which is typically achieved through *Principal Component Analysis (PCA)*.

Dimensionality reduction is extremely important in finance as the size of the inputs can grow very large. An obvious example is hedging a basket option of equities. The inputs will consist of a variety of highly correlated assets and there is little doubt that the majority of the variance can be summarised in a space of much smaller dimension.

Let $\mathbf{x}$ be a (mean zero) d-dimensional input with variance matrix Σ. The aim of PCA is to localise a linear combination $z_1 \triangleq \alpha_1^\top \mathbf{x}$ such that the variance $\alpha_1^\top \Sigma \alpha_1$ is maximised subject to the constraint

$\alpha_1^\top \alpha_1 = 1$. This is clearly a Lagrange problem with a first order condition of

$$\Sigma\alpha_1 = \lambda\alpha_1,$$

where λ is the Lagrange multiplier. From the above condition we can by definition recognise λ as the eigenvalue of Σ with α_1 as the corresponding eigenvector. The question is therefore simply which eigenvalue should be chosen. Multiplying each side of the equation by $\alpha_1^\top$ and imposing the constraint $\alpha_1^\top \alpha_1 = 1$ reveals that the subsequent variance will in fact be the eigenvalue λ. The eigenvalue should therefore be chosen as the *largest* of the eigenvalues.

The next step localises another linear combination $z_2 \triangleq \alpha_2^\top \mathbf{x}$ which again maximises variance under the additional constraint that it should be uncorrelated with z_1. This can once again be shown to yield an identical first order condition [26, ch. 1] and thus the eigenvector corresponding to the second largest eigenvalue is chosen. This continues such that the kth principal component z_k is chosen to maximise variance subject to being uncorrelated with the prior $k-1$ principal components which happens to be achieved if the eigenvector corresponding to the kth eigenvalue is chosen.

As the variance explained by the principal components clearly decreases as the procedure is carried out, the dimension reduction can be achieved by only keeping the first $k \leqslant d$ principal components as input. In fact, it can be shown that if we consider a transformation $\mathbf{z} = \mathbf{A}\mathbf{x}$ for some $k \times d$ matrix, then the trace of the corresponding covariance matrix of $\mathbf{z}$ is maximised by choosing the columns of $\mathbf{A}$ as the first k eigenvectors of Σ corresponding to the first k eigenvalues [26, ch. 2].

The index k is usually chosen by considering some cutoff point $c \in (0, 1)$ usually chosen around 70% to 90%. Letting $\lambda_1, \ldots, \lambda_d$ represent the (sorted) eigenvalues of Σ with λ_1 being the largest, the proportion of *explained variance* relative to the total variance of $\mathbf{x}$ corresponding to choosing the first k eigenvalues is

$$\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{d} \lambda_j}.$$

If one desired to at least keep a proportion c of the total variance, one simply chooses the smallest k such that the above expression is larger than c. The numerator is equivalent to the trace of the variance matrix of $\mathbf{z}$, so the method is closely tied to the aforementioned result.

3.5.3 Batch Normalisation

The stochastic gradient descent algorithm relies on a first order Taylor expansion while ignoring higher order effects. At each iteration, the weights are updated under the assumption that the distribution of the input to the layer remains unchanged (prior layers' weights remain unchanged). As this is not the case, the internal layers constantly has to adapt to a changing input distribution. The network is said to experience *internal covariate shift*, when the input distributions change during training. In the presence of internal covariate shift, the learning rate has to be smaller, since even small changes in the first layers can have dramatic effects in the final layers.

A proposed solution named *batch normalisation* involves normalising the first and second moment of the input *at each layer*, thus ensuring a lesser degree of internal covariate shift [27]. More specifically, let $\mathbf{A}_l \in \mathbb{R}^{m \times n_l}$ denote the output of the lth hidden layer for some mini-batch of size m. In the notation of chapter 3.3 the computation of the $(l+1)$th layer is

$$\mathbf{A}_{l+1} = g_l\left(\gamma_l \frac{\mathbf{W}_{l+1}\mathbf{A}_l - \mu_l}{\sigma_l} + \beta_l\right),$$

where $\mu_l, \sigma_l \in \mathbb{R}^{n_{l+1}}$ are the empirical element-wise mean and standard deviation of $\mathbf{W}_{l+1}\mathbf{A}_l$. The parameters $\gamma_l, \beta_l \in \mathbb{R}^{n_{l+1}}$ are *learnable parameters*, which ensures that the network maintains its predictive

power after normalisation. The backpropagation is performed *including* the batch normalisation. Note that the bias vector $\mathbf{b}_l$ is redundant in the presence of β_l and is therefore omitted.

While batch normalisation can lead to quite substantial improvements in training it is important to understand that the optimisation is far from *free* from a computational point of view. Adding the extra batch normalisation layers can drastically alter the speed of the backpropagation algorithm as it essentially increases the depth of the network.

Chapter 4

Approximating Risk Optimal Portfolio Strategies

In this chapter the theories presented in the prior two sections are conjoined to formulate the liability owner's problem as a statistical learning problem. This will naturally lead to the algorithm developed in [5]. To begin with, only Markovian derivatives are considered as this allows for the simplest introduction of the algorithm. More complicated extensions are treated in subsequent chapters.

4.1 The Misdeeds of Δ Hedging

Before introducing various statistical learning solutions we must first understand why Δ hedging is not a sufficient tool for hedging in the first place. Assume that the economy is arbitrage-free and complete as described in chapter 2.1 and that the filtration is purely generated by a Brownian Motion. As the setup allows for random processes which are not necessarily tradable assets it will be necessary to refer to the state of the economy as a whole. If $\mathbf{U}_t$ denotes all the remaining non-tradable assets, then the entire state of the economy is summarised as $\mathbf{X}_t \triangleq (\mathbf{S}_t, B_t, \mathbf{U}_t)^\top$.

Assume finally that the state process $(t, \mathbf{X}_t)$ adheres to the *Markov property* and that the value of the derivative only depends on the terminal value of the state process through a measurable function. This type of derivative is often labelled as being Markovian. The immediate consequence of such a definition is that the value process can be expressed as [28, ch. 4.3]

$$\frac{H_t}{B_t} = E^Q\left[\frac{Z}{B_T}\middle|\mathcal{F}_t\right] = E^Q\left[\frac{Z}{B_T}\middle|\mathbf{X}_t\right] = \phi\left(t, \mathbf{X}_t\right) \tag{4.1.1}$$

for some real valued measurable function ϕ. If we further assume that ϕ is smooth[1], then the *continuous time* Δ of the derivative can be computed as

$$\Delta\left(t, \mathbf{x}\right) \triangleq \nabla_{\mathbf{x}}\phi\left(t, \mathbf{x}\right).$$

As the economy is purely driven by a Brownian motion, it is possible to derive a formula for the replicating portfolio strategy $\mathbf{a}_t$. To do this, we must of course first specify the risk neutral dynamics of the economy.

[1] A function ϕ is smooth if: $\phi \in C^1(\mathbb{R})$ with respect to time and $\phi \in C^2\left(\mathbb{R}^d\right)$ with respect to the underlying state process.

Denote the dynamics of $\mathbf{S}_t$ and $\mathbf{U}_t$ as respectively,

$$d\mathbf{S}_t = \cdots dt + D\left[\mathbf{S}_t\right]\boldsymbol{\Sigma}_s(t)dW_t^Q,$$
$$d\mathbf{U}_t = \cdots dt + \boldsymbol{\Sigma}_u(t)dW_t^Q.$$

Applying the multivariate Itô formula, remembering that the numeraire has no diffusion terms, and that H_t/B_t is a martingale under Q such that all dt terms vanish yields,

$$d\left(\frac{H_t}{B_t}\right) = \left(B_t\nabla_{\mathbf{s}}\phi D\left[\frac{\mathbf{S}_t}{B_t}\right]\boldsymbol{\Sigma}_s(t) + \nabla_{\mathbf{u}}\phi\boldsymbol{\Sigma}_u(t)\right)dW_t^Q$$

By comparing the expression with equation (2.1.4) it is clear that the replicating portfolio strategy $\mathbf{a}_t$ only depends on the derivative through the continuous time Δ. In fact, due to the completeness of the model, $\Sigma_s(t)$ is invertible, so the exact form of the replicating portfolio is

$$\mathbf{a}_t = B_t\Delta\left(t, \mathbf{X}_t\right) + \nabla_u\phi\left(t, \mathbf{X}_t\right)\Sigma_u(t)\Sigma_s^{-1}(t)D\left[\frac{B_t}{\mathbf{S}_t}\right].$$

We know from proposition 2.3.5 that this strategy solves the liability owner's problem if the market is sufficiently efficient. Note that while it is technically only shown that the portfolio strategy replicates the claim under Q, the result follows from the fact that the measures are equivalent and thus a Q-a.s. event is also a P-a.s. event.

4.1.1 The Δ Bleed

To understand the issues relating to Δ hedging, consider for simplicity a market only consisting of a single tradable asset and zero interest rate. The liability owner is short a Markovian derivative and protects the position by Δ hedging. Assuming ϕ is smooth, it follows by the Itô formula that the instantaneous change in the profit-and-loss is

$$\Delta_t dS_t - dH_t = -\left(\theta_t dt + \frac{1}{2}\Gamma_t\left(dS_t\right)^2\right), \tag{4.1.2}$$

where θ_t denotes the time derivative and Γ_t denotes the second derivative of ϕ. We do not have to explicitly assume that the economy is driven by a Brownian Motion as the above can simply be considered as a second order Taylor expansion instead.

If the portfolio is readjusted continuously, the right hand side is famously known to be zero [29], but if not, the expression describes the *discretisation bias* as discussed in chapter 2.3.1. For convex payoffs such as the European call option, the θ is negative and the Γ is strictly positive. The term relating to the θ is therefore beneficial for the liability owner and the term relating to the Γ is problematic.

When readjustments are conducted discretely, the value of the underlying asset moves between two readjustment dates. The subsequent shift in the Δ leads to a loss as the liability owner's holdings will not match the Δ. The degree of the loss is proportional to how much the Δ moves which is measured by the Γ. As implied by the squared dS term, the direction of the shift in the underlying asset has no influence, only the magnitude. The movement in the Δ is referred to as the Δ *bleed*.

The Γ will of course also change in value as a result of the shift in the underlying asset. This is naturally called the Γ *bleed*, but it is not captured by our second order Taylor expansion, as this is a third order effect. Neither is the fourth, fifth and higher order effects.

4.1.2 Hedging with Transaction Costs

The Γ is even more important with the inclusion of (proportional) transaction costs. By their very nature, the cost from a readjustment is scaled by the magnitude of the change in portfolio holdings. This shift is scaled by the Γ as well since the portfolio holdings *is* the Δ. This intuition is confirmed by more rigorous analysis in [30] which unsurprisingly also verify that high expected rates of return of the instrument negatively affect the profit-and-loss.

It seems the classical literature on hedging under transaction costs can be divided into two factions. Most of the results only hold in the Black and Scholes model and we shall therefore limit the discussion to this case. As this chapter is only meant to provide intuition the simple model will suffice.

The *time-based* strategies were popularised in [31] who proposed hedging (short) call option positions at *exogenous* discrete time points by increasing the volatility in proportion to the costs and first (absolute) moment of the returns of the instrument. The holdings in the bank account are modified likewise, which has the unfortunate consequence that the strategy is no longer self-financing. Increasing the volatility flattens the Δ when viewed as a function of the log-moneyness $\ln(F/K)$ and so the strategy essentially entails buying more when the asset is relatively cheap and buying less when the asset is relatively expensive. A high expected (absolute) rate of change entails more flattening as this accentuates the need for protection against transaction costs.

In the original paper, it is claimed that this strategy results in a perfect hedge for $dt \to 0$, however, this was quickly proven to be false [32]. Intuitively, as the time interval decreases to zero the altered volatility diverges towards infinity, which in turn yields a Δ equal to one. The error in the proof relates to an erroneous application of uniform convergence [33].

The alternative *move-based* strategies seem to have evolved from [34]. In the paper, it is shown that under the entropic risk measure (see [5]), the optimal portfolio strategy involves formulating *no-trade regions* as functions of time and the instrument. As long as the holdings in the underlying asset remains within the intervals, no trading is to be performed. Once the holdings move outside the intervals, the liability owner should trade such as to remain on the boundary.

While theoretically well justified, the no-trade intervals are computationally slow to find and therefore not useful in practice. An important addition to the literature was the development of [35] in which asymptotic approximations of the boundaries where derived for small transaction costs. As noted in [30] transactions costs from the liability owners point of view are small so these approximations are quite reasonable. Alternatively, many improvements in numerical approximations have since been developed [36].

The aforementioned methods rely on various model specifications, risk measures, asymptotic approximations etc., and none of them can truly be said to apply to general models. In the next chapter, a framework for computing move-based approximations to the optimal portfolio strategies is presented which is applicable to any model.

4.2 A Portfolio Strategy Approximation Algorithm

This chapter will explore the work of [5]. The paper presents a framework for approximating solutions for the liability owner's problem from chapter 2.3.2 for a large class of derivatives. For now, the applicability of the algorithm for Markovian derivatives is explored, but it is extended to jump models and path dependent derivatives in later chapters.

In order to formulate the algorithm, the liability owner's problem is approximated in three different ways. The first approximation is a necessary consequence of moving from continuous to discrete time.

The second and third approximation stem from the realisation that the liability owner's problem is essentially a special case of the statistical learning problem from chapter 3.

In order to work with the setting from chapter 2.1 computationally, the problem needs to be approximated by a discrete time model. Time is discretisised by approximating the continuous time interval from 0 to T into n parts

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

No hedging strategy is required for the last period, so it is only necessary to specify $\mathbf{a}_{t_k}$ for $k = 0, \ldots, n-1$. As it has already been established that explicitly specifying the holdings in the numeraire process is superfluous, it is only necessary to Euler discretise the terms inside the risk measure in π. In accordance with prior chapters, we can define the discrete time approximations of the value- and cost processes. By fixing the integrand values at the beginning of each period, these approximations are

$$\frac{V_T}{B_T} = \sum_{k=0}^{n-1} \mathbf{a}_{t_k} \left(\frac{\mathbf{S}_{t_{k+1}}}{B_{t_{k+1}}} - \frac{\mathbf{S}_{t_k}}{B_{t_k}} \right), \qquad \frac{C_T}{B_T} = \sum_{k=0}^{n-1} \left| \mathbf{a}_{t_{k+1}} - \mathbf{a}_{t_k} \right| \left(\frac{\mathbf{c}_{t_k}}{B_{t_k}} \right).$$

In the above it is implicitly assumed that $\mathbf{a}_{t_n} = \mathbf{a}_{t_{n-1}}$ such that no transaction costs are incurred in the final period. The instruments and numeraire processes are likewise approximated to discrete time[2], however the dynamics depends on which risk measure is chosen[3]. Finally, it is implicitly assumed that the transaction costs are proportional using some process $\mathbf{c}_{t_k} \in \mathbb{R}^d$. Throughout the thesis, this is chosen as $\gamma \mathbf{S}_{t_k}$ for some $\gamma \in (0, 1)$.

Under the assumption that the risk measure is an OCE risk measure, the discrete time approximation of the hedging problem from chapter 2.3 is naturally

$$\pi(X) = \inf_{w, \mathbf{a}_{t_k}} E\left[w + \ell\left(-\left[\frac{V_T}{B_T} - \frac{C_T}{B_T} + X \right] - w \right) \right]. \tag{4.2.1}$$

This is a special case of the statistical learning problem from chapter 3 albeit without any labels $\mathbf{Y}$. As with every other statistical learning problem, we approximate the optimal solution by some parametric family and replace the expected value by an empirical estimate. These are the final two approximations.

The portfolio strategy $\mathbf{a}_{t_k}$ at time t_k is approximated by a parametric family of functions $h_k : \mathcal{X} \times \Theta \longrightarrow \mathbb{R}^d$ for some feature domain $\mathcal{X}$ and parameter set Θ. A feature $\mathbf{x}_k \in \mathcal{X}$ represents the information available to the liability owner at time t_k. The exact specification of the feature set depends on the specific derivative and parametric family, but will in general contain the previous period's holdings and current period's instruments. See chapter 4.3 for details.

It is not feasible to compute the expected value in equation (4.2.1) analytically. The expected value is therefore numerically estimated across m independent simulations of the sample paths by utilising the empirical expected value instead. For a batch size of m, the samples can be stacked as $\mathbf{S}_{t_k} \in \mathbb{R}^{m \times d}$ and $\mathbf{B}_{t_k} \in \mathbb{R}^m$. The remaining quantities $\mathbf{a}_{t_k}$, $\mathbf{c}_{t_k}$ and $\mathbf{X}$ are likewise extended with a batch dimension. Assume further that the portfolio strategy approximation is $h_k : \mathcal{X} \times \Theta \longrightarrow \mathbb{R}^{m \times d}$ where the elements of $\mathcal{X}$ are also extended. Note importantly that neither the parameter set Θ nor w is extended with a batch dimension.

With the above formulation it is possible to iteratively compute the terminal wealth for sample paths of the instruments and numeraire. For an initial cash-injection $\mathbf{V}_{t_0} \in \mathbb{R}^m$ and cost $\mathbf{C}_{t_0} \in \mathbb{R}^m$ initialised

[2] Possibly by an Euler discretisation, but more advanced discretisation schemes are also available.

[3] If for example the expected value in the OCE risk measure from chapter 2.2.2 is chosen with respect to some martingale measure, then one would discretise under the risk-neutral dynamics.

at zero iteratively set for $k = 0, \dots, n-1$

$$\begin{aligned}
\mathbf{a}_{t_k} &= h_k\left(\mathbf{X}_k; \theta_k\right), \\
\frac{\mathbf{V}_{t_{k+1}}}{\mathbf{B}_{t_{k+1}}} &= \frac{\mathbf{V}_{t_k}}{\mathbf{B}_{t_k}} + \left[\mathbf{a}_{t_k} \odot \left(\frac{\mathbf{S}_{t_{k+1}}}{\mathbf{B}_{t_{k+1}}} - \frac{\mathbf{S}_{t_k}}{\mathbf{B}_{t_k}}\right)\right]\mathbf{1}, \\
\frac{\mathbf{C}_{t_{k+1}}}{\mathbf{B}_{t_{k+1}}} &= \frac{\mathbf{C}_{t_k}}{\mathbf{B}_{t_k}} + \left[\frac{\mathbf{c}_{t_k}}{\mathbf{B}_{t_k}} \odot \left|\mathbf{a}_{t_{k+1}} - \mathbf{a}_{t_k}\right|\right]\mathbf{1}.
\end{aligned}$$

Note that $\odot$ denotes the element-wise multiplication (Hadamard product) and $\mathbf{1}$ is a vector of ones. An approximation of the problem in equation (4.2.1) is then

$$\inf_{w,\theta_k} J\left(\theta_0, \dots, \theta_{n-1}, w\right) \triangleq \inf_{w,\theta_k} \left\{ \frac{1}{m} \sum_{i=1}^{m} w + \ell\left(- \left[\frac{\mathbf{V}_T^{(i)}}{\mathbf{B}_T^{(i)}} - \frac{\mathbf{C}_T^{(i)}}{\mathbf{B}_T^{(i)}} + \mathbf{X}^{(i)}\right] - w\right)\right\}. \tag{4.2.2}$$

The objective function J is essentially a function taking in the n parameter vectors of the portfolio strategy and w, utilising the portfolio strategy in a hedge simulation of m sample paths, and then empirically estimating the risk associated with the portfolio strategy resulting in a single real valued output. By utilising AAD to differentiate J the optimal parameter vectors (and therefore portfolio strategies) and w can be found by gradient descent or any other extension as described in chapter 3.4. The actual optimisation procedure will therefore require multiple hedge simulations to be conducted as the parameter vectors converge.

A major advantage is that the simulation of $\mathbf{S}_{t_k}$, $\mathbf{B}_{t_k}$ and $\mathbf{X}$ for all $k = 0, \dots, n-1$ and subsequent minimisation of equation (4.2.2) are completely separate computations. It is therefore possible to utilise various multiprocessing and vectorisation techniques to efficiently sample a large batch of instruments and numeraires before beginning the minimisation. In fact, the entire evaluation of J can be vectorised extensively which significantly decreases the computational time in `Tensorflow`.

4.3 Adapting the Algorithm to Markovian Derivatives

The description of the algorithm in chapter 4.2 is incomplete without an exact specification of the parametric family and the feature set. In this chapter, two different specifications are presented each specifically adapted to the Markovian setting with- and without transaction costs.

4.3.1 Deep Hedging

It is intuitively clear that if all h_k are *universal approximators* then the problem in equation (4.2.1) and (4.2.2) are identical. In [5], the author even goes through the trouble of proving it in the context of using neural networks as predictors. Unsurprisingly, the authors in the paper as well as [3] therefore propose utilising deep feedforward neural networks as predictors of the optimal portfolio strategy.

Formally, at each point in time, the portfolio strategy $\mathbf{a}_{t_k}$ is parameterised as a feed forward neural network $h_k\left(\mathbf{x}_k, \theta_k\right)$. The weights θ_k are distinct for each network so the number of parameters in the model easily surpasses several thousands depending on the complexity of each network. As is characteristic of deep learning, the subsequent optimal solutions are also impossible to interpret - so-called *black boxes*.

A network at time t_k receives a feature vector $\mathbf{x}_k$ representing the information available to the agent. From a measure theoretic point of view, it is clear the the σ-algebra generated by the information set

$\mathbf{x}_k \in \mathcal{X}$ must contain $\mathcal{F}_{t_k}$. Otherwise, insufficient information is provided to properly represent the optimal solution. This is an abstract formulation and the exact specification of the information set is important.

When the derivative is Markovian and no transaction costs are present, the information set only has to contain the current state of the economy $\mathbf{X}_{t_k}$. As explained in chapter 3.5.2 it is highly beneficial to preprocess the features prior to training. If the transaction costs are nonzero we shall follow [5] and include the holdings of the prior period in the feature set[4],

$$\mathbf{x}_k = \mathbf{X}_{t_k} \otimes \mathbf{a}_{t_{k-1}}. \tag{4.3.1}$$

Importantly, the feature set is now generated dynamically at each iteration during the minimisation procedure as the portfolio strategy of the prior period's network will converge. A consequence is that preprocessing of $\mathbf{a}_{t_{k-1}}$ is not feasible. Fortunately, if batch normalisation is used, this is not a major concern.

4.3.2 The Linearised Δ Strategy

Akin to how the normal distribution has a high entropy, using neural networks can be viewed as incorporating very little prior belief about the optimal solutions into the approximation. Truly, the only real assumption is that the optimal solution is continuous and consists of compositions, multiplications and additions of many smaller functions. This is arguably why neural networks appear in such a wide variety of applications and is often the first choice as parametric family when solving statistical learning problems.

The issue with neural networks is that we *do* in fact have strong, well-founded, prior beliefs about the optimal solutions of the liability owner's problem. As we saw in chapter 4.1.2, optimal solutions of simpler problems involve alterations of the Δ. It was even proven in proposition 2.3.5 that under sufficient assumptions, the theoretical Δ is the optimal strategy. Once transaction costs and time discretisation is introduced this is no longer the case, yet it seems unpragmatic to discard the theory altogether.

In order to test the hypothesis that incorporating prior beliefs is beneficial, consider letting the feature set consist of the continuous time Δ and simply utilising the linear approximator from chapter 3.2 such that

$$h_k(\mathbf{x}_k, \theta_k) = \theta_k^{(0)} + \theta_k^{(1)} \odot \Delta(t_k, \mathbf{x}_k) + \theta_k^{(2)} \odot \mathbf{a}_{t_{k-1}}. \tag{4.3.2}$$

Importantly, the computation of the Δ can be conducted prior to optimising the parameters. This method is therefore computationally very efficient. The disadvantage lies in the fact that the Δ of the derivative book must be known beforehand, but as we shall see in chapter 5, this is not nearly as big an issue as it might seem. Utilising this parametric family or any other of similar simplicity is not a part of the original paper [5] or seemingly any other literature for that matter.

Utilising this parametric family or the more general linear feature maps from chapter 3.2 is particularly well-behaved if there are no transaction costs, as the function is then linear in the parameters.

Proposition 4.3.3. *If the parametric family h_θ is linear in the parameters, then the objective function J from equation* (4.2.2) *is convex in θ and w.*

Proof. As linear combinations of linear functions are linear, it follows that V_T/B_T is linear. As a composition of a convex function (the absolute value) and a linear function is still convex, it follows that C_T/B_T is a weighted sum of convex functions, and therefore itself convex. Then $V_T/B_T - C_T/B_T$ is concave, since the linear term has no influence on the Hessian matrix. The entire expression inside ℓ is

[4]The concatenation operator: $(x_1, \ldots, x_n)^\top \otimes (x_{n+1}, \ldots, x_{n+k})^\top = (x_1, \ldots, x_{n+k})^\top$.

then convex which implies that the entire expression inside the expected value is convex, since ℓ is itself convex and non-decreasing. As the minimisation with respect to w is over the convex set $\mathbb{R}$, the proof is complete. □

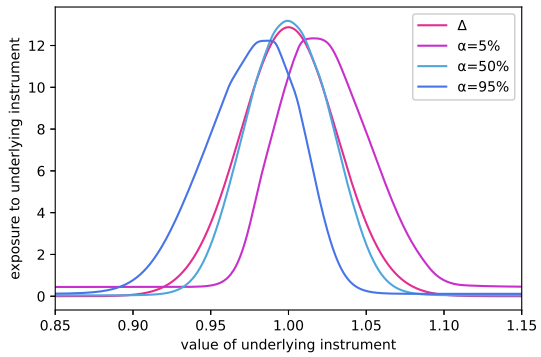
4.4 Effect of Risk Aversion in Univariate Environments

Before performing numerical tests of the algorithm from [5] we shall first verify that the algorithm produces sensible solutions in a simple univariate setting. Consider the Bachelier model,

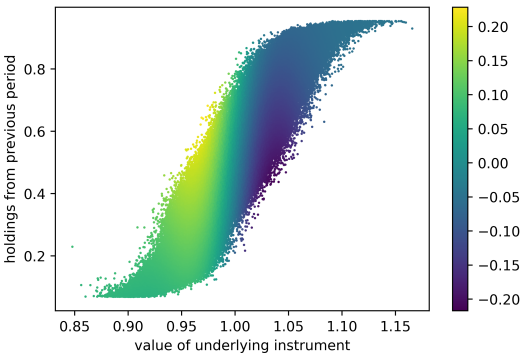
$$dS_t = \sigma dW_t, \quad S_0 = 1,$$

where we assume σ is 20% and that the interest rate is zero. The liability owner is assumed short an at-the-money binary option with a two week maturity and she wishes to protect the position by trading in the underlying instrument daily. The risk measure is computed under the risk neutral measure, so proposition 2.3.5 implies that Δ hedging is optimal for $dt \to 0$. As time discretisation is present, the optimal strategy should compensate for the higher order effects described in chapter 4.1.1.

A collection of feedforward neural networks are trained according to the descriptions in chapter 4.2 and 4.3.1. The details regarding the training procedures are reserved for the next chapter. The (trained) portfolio strategies are visualised in figure 4.1a at day eight where α indicates the risk level in the expected shortfall.



(a)



(b)

Figure 4.1: Result of experiments described in chapter 4.4.

To understand the behaviour of the algorithm, remember that the Γ of the binary option has the shape of a mirrored S rotated 90 degrees. In other words, it is positive when the spot is out-of-the-money and negative otherwise. Also the Γ peaks and troughs at approximately 0.96 and 1.04, and converges towards zero for the spot tending in either direction.

Returning to equation (4.1.2) we know that there is approximately two (opposing) sources of error: Δ bleed and θ decay. Both are zero at-the-money, but due to *time discretisation*, a large downward movement leads to a loss as the Γ will increase during the movement and vice versa for a large upward movement. For small movements, the θ decay is the dominant force yielding gains from downward moves and losses for upward moves. Regular Δ hedging does not take this into account.

The expected shortfall at level α is proportional to the expected shortfall at level $1 - \alpha$ as the loss distribution is symmetrical around zero. We would therefore expect the strategy to be identical at-the-money which is exactly the case. From the risk averse agent's point of view she will protect herself against a large negative move by lowering the holdings in the underlying asset.

As the third order moment *speed* increases as the spot moves further out-of-the-money the potential losses from a large positive move will at some point dominate the losses from a potential large downwards move, so a risk averse agent will sometime before the peak of the Γ begin overemphasising the exposure to the underlying asset relative to Δ hedging. The speed also increases as the spot moves further in-the-money so the same logic can be applied for the risk willing agent in this case. From figure 4.1a it is clear that this is the exact strategy adopted by the algorithm. We can there conclude that the algorithm *does* produce sensible solutions in this simple setting.

Similarly, we can investigate the behaviour when transaction costs are included. To make the results comparable to the theoretical results from [31] we shall train the networks to approximate optimal portfolio strategies for an at-the-money call option in the Black and Scholes model with no rates, 20% volatility and 1% proportional transaction costs. The behaviour of the algorithm at time t_k can be illustrated by a scatter plot $\left(S_{t_k}, a_{t_{k-1}}\right)$, where each dot is coloured according to

$$h_k\left(x_k \otimes a_{t_{k-1}}; \theta_k\right) - \Delta\left(t_k, X_{t_k}\right).$$

Such an illustration can be found in figure 4.1b at day eight for 256k test sample paths. The model seems to overemphasise holdings of the instrument for out-of-the-money spot values and vice versa for in-the-money spots.

The behaviour is strikingly similar to the transaction cost adjusted volatility strategy from [31] described in chapter 4.1.2 which exactly involved a flattening of the Δ around the strike. A major improvement for the deep network is that it can incorporate the prior periods holdings to produce an even better strategy.

4.5 Testing the Algorithm in Multivariate Environments

This chapter contains an exploration of the numerical performance of approximating hedging strategies in the setup from [5] discussed in chapter 4.2. The tests performed deviate from the paper in many important aspects and is in general more rigorous.

The most substantial deviation relates to the dimension of the instrument process. In general the performance of the algorithm does not seem to have been investigated anywhere before for high dimensional state processes despite this setting arguably being the most relevant one. In the paper [5] there is a small example estimating high dimensional performance, but the authors themselves suggest that a proper systematic study is required. In other words, the algorithm must hedge derivative *books* consisting of multiple derivatives written on multivariate instrument processes as the performance for single derivatives can be very misleading.

As no test is complete without some form of benchmark with which to gauge the relative performance with, we shall model the underlying state process as the correlated Black and Scholes model (see appendix A for details). Utilising this model allows for analytical calculations of the continuous time Δ.

Much care must be taken to provide a comparable test environment across dimensions. For dimension d, the derivative book consists of d butterfly spreads with a fourteen day maturity each written on one of the d underlying instruments. A butterfly spread is chosen as the sign of the Γ is spot dependent thus making it more difficult to hedge than a convex payoff. The book is hedged daily. The marginal distribution of each instrument is kept identical with a drift of 5% and volatility of 20%. The spot is likewise normalised to 100 and the interest rate is kept at 2%. This allows us to keep the overall arbitrage-free price of the derivative book constant across dimensions by setting the exposure to each

spread to $1/d$. Formally, for a spread of c and strike $K = S_0$, the payoff of the derivative book is

$$\sum_{i=1}^{d} \left[\frac{1}{d} \left(S_T^{(i)} - \left(K - \frac{c}{2} \right) \right)^+ - \frac{2}{d} \left(S_T^{(i)} - K \right)^+ + \frac{1}{d} \left(S_T^{(i)} - \left(K + \frac{c}{2} \right) \right)^+ \right].$$

As risk is measured in absolute terms, the anchoring of the price helps us to compare performance across dimensions. As the risk measures are convex, this will not be fully achieved, since introducing more instruments allows for a greater degree of *diversification*. Even the simple Δ hedge will seemingly inexplicably decrease in risk as the dimension increases despite no alteration to the strategy. Thus, we will showcase the algorithm's performance relative to Δ hedging.

While the marginal distribution is identical for all derivatives, we shall test the robustness of the algorithm by randomly generating the correlation matrix of the returns for each repetition. Further, for transaction costs, one of the advantages of using statistical learning is to allow the predictors to utilise correlation to create cheaper hedging strategies; a consideration which is not present in regular Δ hedging.

The motivations for different parametric families and feature sets are already described in chapter 4.3. For all experiments, the following three types of parametric families are tested,

1. *deep network* refers to a 15 unit feedforward neural network with four layers. The feature set is chosen as the log of the discounted instrument process and the prior period's holdings is appended if transactions costs are nonzero. The feature set is normalised as preprocessing. The network uses batch normalisation between each layer. The activation function is softplus.

2. *deep network w. PCA* refers to a network with an identical architecture as the *deep network*. The feature set is chosen similarly, but PCA with a threshold of 90% as described in chapter 3.5.2 is used as preprocessing instead. The network also uses batch normalisation between each layer.

3. *feature map w. true* Δ refers to the linearised Δ strategy described in equation (4.3.2) with the analytically computed Δ as feature set.

For all the experiments, the 95% expected shortfall is chosen as risk measure. The networks are trained for 2^{18} (256k) training samples and subsequently tested on 2^{18} test samples. These levels were chosen to ensure proper generalisation from training to test set. The mini-batch size is 2^{10} (1k) and each network is trained wtih the optimisations described in chapter 3.5.

4.5.1 Without Transaction Costs

With no transaction costs the only risk is the Δ bleed and other higher order losses as the time discretisation is the only source of inefficiency. It should therefore be much easier to locate the optimal portfolio strategy in this setting.

From proposition 2.3.5 it follows that if the volatility matrix of the underlying instruments is invertible, the instruments are simulated under the (unique) equivalent martingale measure, and no transaction costs are present, then the optimal solution under continuous portfolio readjustments is the continuous time Δ.

The above considerations leads us to the conclusion that if we train the models in the aforementioned setting then the performance of the Δ should act as rather tight lower bound on the optimal solution. If the parametric families do outperform the Δ we can expect the approximation to be *very* close to the optimal solution.

As the description of the algorithm in chapter 4.2 is quite intricate, the training procedure for the deep network is reiterated here. The time interval is discretised into $n = 14$ parts such that $t_0 = 0$ and

$t_n = T$ where $T = 14/250$. Then m samples of the d dimensional instrument process (and numeraire) are simulated at each point in time. From this the feature sets $\mathbf{X}_0, \dots, \mathbf{X}_{n-1} \in \mathbb{R}^{m \times d}$ are constructed by taking the logarithm of the discounted instruments and subsequently normalising said values. The payoff $\mathbf{Z} \in \mathbb{R}^m$ of the derivative book is also computed for each sample path.

The deep networks $h_1, \dots, h_{n-1}$ are then initialised such that training can commence as follows. For each iteration of the training procedure the $n-1$ portfolio strategies are computed from the relation,

$$\mathbf{a}_{t_k} = h_k\left(\mathbf{X}_k, \theta_k\right) \in \mathbb{R}^{m \times d}, \quad k \in \{0, \dots, n-1\}.$$

The portfolio strategies are then utilised to compute the empirical estimate of the risk of the portfolio strategy in accordance with equation (4.2.2). The derivatives of this expression with respect to w and θ_k for $k = 0, \dots, n-1$ are then computed using AAD. The gradients are used to perform a single step of the gradient descent algorithm. The procedure is continued until the change in risk between two iterations is below some threshold. The training of the feature map proceeds in the exact same manner except that the approximators are instead given by equation (4.3.2) where the Δs are computed as a part of the feature construction phase. Each experiment is repeated sixteen times and an average is computed over the measurements.

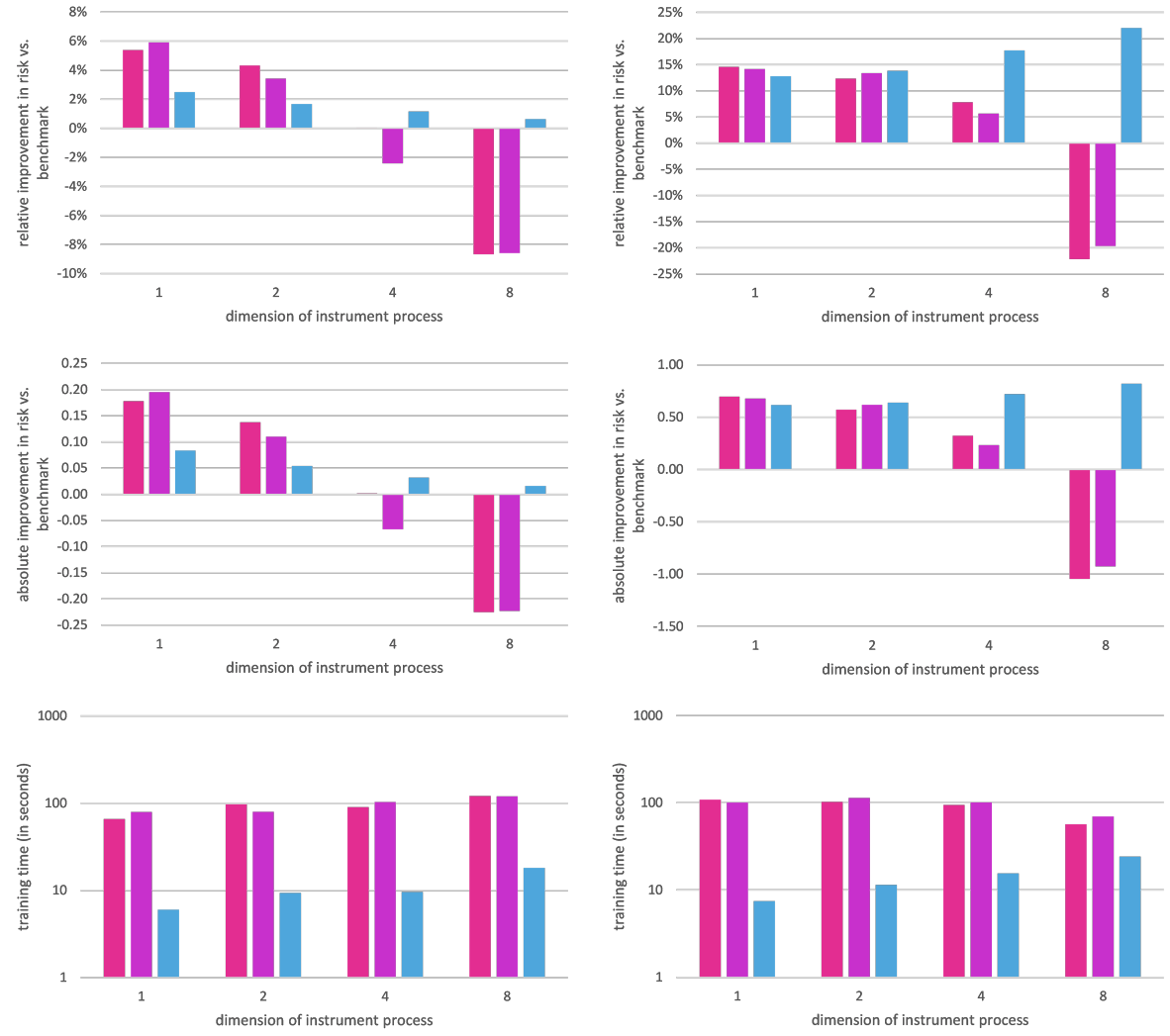
The results are illustrated in the left column of figure 4.2. In accordance with the findings from [5] we find that the deep network approximates a superior risk allocation in the univariate setting. The deep network manages to improve upon Δ hedging by 5% despite our earlier arguments that Δ hedging should be very close to optimal. We further extend upon the findings of [5] by verifying that the deep network also approximates a superior risk allocation for instrument processes up to four dimensions with a relatively unaffected training time. The invariance in training time relative to input dimension is arguably one of the most important traits of neural networks.

However, we also discover upon increasing the dimension to eight, that the training procedure for the deep network seems to converge to an inferior approximation relative to the Δ. The fourteen deep networks already contains more than thirteen thousand parameters in total and the training time already surpasses 100 seconds, so increasing the size of the networks is questionable. It is also important to note that there is no difference between the training- and test risk so it is not a generalisation issue due to lack of data. These results clearly illustrate why deep learning solutions must always be tested in multivariate settings.

The conclusion does not change when dimensionality reduction is utilised. In fact, the performance slightly deteriorates compared to the deep network. The dimension is generally not reduced much when performing PCA since there is very little linear dependence between the sample paths of the instrument processes. Nonetheless, it was important to verify that the lacking performance of the deep network in high dimensions is not due to sloppy preprocessing.

The simpler feature map parameterisation manages to consistently stay above the benchmark in terms of risk allocation. This is of course expected as the feature map should always perform at least as good as Δ hedging, so the results for this predictor without transaction costs are not very interesting. The model only contains around five hundred parameters, so the training time is very fast.

Overall, the predictors manage to utilise the discretisation bias in order to achieve a slightly more favourable wealth distribution than the benchmark for daily hedging in lower dimensions. The question remains if this is still possible if we decrease the bias further. As the time discretisation is the only possible source of error, increasing the frequency of hedging should lower the gap. To test this hypothesis another experiment is conducted where the hedge frequency is raised to five times daily ($n = 70$). A single derivative book is generated for $d = 1$ and $d = 10$. The test only runs for a single repetition as the



butterfly spread 14 days
repetitions 16
simulations under Q measure
arbitrage-free price of book: 1.93

deep network 4 layers 15 units
deep network 4 layers 15 units and PCA
feature map w. true Δ
benchmark is true Δ

butterfly spread 14 days
repetitions 16
transaction cost 1%
simulations under P measure

Figure 4.2: Results of the experiment described in chapter 4.5.

training time is very slow. The deep network impressively still managed to improve upon the benchmark by 2% in the univariate setting, but massively underperformed by -21.4% in the multivariate setting. The feature map improved upon the benchmark by respectively 1.3% and 0.1%.

4.5.2 With Transaction Costs

As it has been verified that the algorithm can locate the optimal hedging strategy in a frictionless setting (at least for low dimensions), it seems reasonable to assume that the same conclusion can be drawn in the presence of transaction costs. For neural networks of sufficient complexity, the universal approximation theorem at least guarantees that the predictor is able to represent the optimal function. However, it is not entirely clear that this is the case for the linear predictors as they are simply affine transformations of the continuous time Δ. In fact, in the previous experiment, the affine transformation simply converges towards the identity map $\theta_0 \longrightarrow \mathbf{0}$, $\theta_1 \longrightarrow \mathbf{1}$ for high dimensions and so the previous experiment is not a justification for the use of linear predictors.

To properly test the algorithm in the presence of frictions, the next experiment involves proportional transaction costs of 1%. The hedging is reduced to the original daily frequency ($n = 14$), but now the instruments are sampled under the statistical measure P and as such, the drift of the instrument is relevant. The correlation structure was relatively unimportant before, but now the algorithm should be able to find efficient ways of exploiting the correlation structure in order to limit the volume of trading.

The training procedure is very similar to the setting without transaction costs. The only difference is that the deep networks are now maps from $\mathbb{R}^{m\times 2d}$ to $\mathbb{R}^{m\times d}$ as the previous period's allocation is appended to the feature set as in equation (4.3.1).

The results are depicted in the right column of figure 4.2. For low dimensions the deep network massively outperforms the benchmark with 15% in improvement for $d = 1$. In absolute terms, the 95% expected shortfall is lowered by 0.6 on average compared to using Δ hedging. As the overall arbitrage-free price of the book is 1.93, this is quite a substantial improvement. The performance systematically worsens as the dimension is increased and at $d = 8$ the network is producing a portfolio strategy with a 20% *higher* risk than the benchmark. Once again utilising PCA has little to no effect on performance. Had the experiments only been run in a univariate setting these findings would have remained hidden.

To test if the architecture or training procedure has an influence on performance, the same experiment at dimension eight is performed for different changes to the deep network architecture: batch size reduced to 256 (lower batch size), number of layers increased to five (more layers), number of units increased to twenty (more units), and different activation functions (sigmoid, relu). The results are illustrated as a boxplot in figure 4.3. Lowering the batch dimension from the original 1024 to 256 does massively increase the performance albeit at the cost of a quadrupling the training time[5]. Increasing the number of layers or units improves upon performance slightly, but the variance also increases massively. Utilising the relu activation function improves upon the result, but in the no transaction cost case from before,

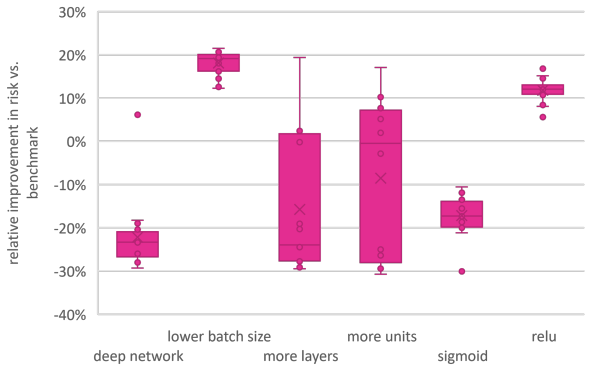


Figure 4.3: Result of experiment described in chapter 4.5.2.

[5]As the batch dimension is lowered to one fourth, the number of iterations in one epoch is quadrupled.

the relu activation function underperforms by -30% (not shown). We conclude that the deep networks *can* approximate the optimal solution in high dimensions, but at a significant cost to computational time.

Returning to figure 4.2, we see that as opposed to the risk neutral setting from chapter 4.5.1 the feature map produces portfolio strategies of similar quality to the deep network in the low dimensional setting. And as opposed to the deep network, the performance *improves* as the dimension increases. By comparing the performance at eight dimensions to figure 4.3 it is seen that the deep network trained with a batch size of 256 or with the relu activation function *still* does not reach the level of the feature map. As the analysis unfortunately does not include a benchmark for optimal hedging under transaction costs we have no way of knowing how close this strategy is to optimal, but it is still remarkable how vast of an improvement is possible by a simple linear transformation.

Utilising the feature map was not intended to showcase anything remarkable in particular about *linear* transformation of Δs, but rather to underline the enormous utility in incorporating well founded prior beliefs into the parametric family. The results clearly suggests that the focus of machine learning applications in finance should be shifted from *black box* solutions and over to precise model-free approximations of Δs. We shall investigate this idea further in chapter 5.

Chapter 5

The Joys of Adjoints

As evident by the findings in the previous chapter, the neural networks can only learn the high dimensional portfolio strategies from pure gradient descent if allowed to train within unreasonable time frames. Also, in the univariate setting, the models produce portfolio strategies which are not directly interpretable due to the black box nature of neural networks.

Interestingly, another finding from the previous chapter is that if the Δ of the derivative book is known, it is relatively simple to transform it into an optimal portfolio strategy under frictions using something as simple as a linear transformation. This suggests that the true challenge lies in computing Δs, *not* adapting the strategy to transaction costs and time discretisation.

The obvious issue lies in the actual computation of the Δ. As analytical pricing formulas are only available for particularly simple models and derivatives, it is not obvious how to apply the strategy in practice. In this chapter a framework for computing Δs for arbitrary models and portfolio strategies is presented. The only requirement is that the model can be simulated under the risk neutral measure and it is therefore *almost* as general as the algorithm from 4.2.

The ultimate goal, which is achieved in chapter 5.7, is to formulate a novel method for approximating risk optimal portfolio strategies which is stable in high dimensional settings, interpretable for humans, and faster to train than utilising neural networks in the algorithm from chapter 4.2.

5.1 The Least Squares Method

Let us first recall the classical regression problem of finding a function $h : \mathbb{R}^d \longrightarrow \mathbb{R}^p$ such that $h(\mathbf{X})$ for some feature vector $\mathbf{X} \in \mathbb{R}^d$ is the best possible predictor of the label vector $\mathbf{Y} \in \mathbb{R}^p$. The problem was briefly touched upon in chapter 3.1.1, but a more detailed exposition is presented now. The error of the prediction is naturally defined as

$$e(\mathbf{X}, \mathbf{Y}) \triangleq h(\mathbf{X}) - \mathbf{Y},$$

which is a vector in $\mathbb{R}^p$. The *best predictor* is defined as the minimiser of the expected squared Euclidean distance of e, which can be written as

$$\min_{h \in \mathcal{H}} E\left[e(\mathbf{X}, \mathbf{Y})^\top e(\mathbf{X}, \mathbf{Y})\right].$$

As argued in chapter 3.1.1, the minimum is achieved if h is equal to the expected value of $\mathbf{Y}$ conditional on $\mathbf{X}$. As it is often neither possible to search in the space of all functions nor to compute the analytical value of the above expectation, the function is approximated by a parametric predictor h_θ and the

expectation is approximated by the empirical estimate,

$$\min_{\theta\in\Theta} \frac{1}{m} \operatorname{tr}\left[e\left(\mathbf{X},\mathbf{Y}\right)^{\top} e\left(\mathbf{X},\mathbf{Y}\right)\right],$$

where the first dimension of $\mathbf{X}$ and $\mathbf{Y}$ is extended with a batch dimension of size m of independent empirical samples. If the predictor can represent the true solution and the sample size is large enough, the predictor should converge towards the correct solution.

The relationship to derivative pricing becomes apparent once one realises that derivative prices are also conditional expectations. The price of a derivative is the expected value of the discounted payoff conditional on the current state process,

$$\phi\left(t,\mathbf{x}\right) = E^{Q}\left[\frac{Z}{B_T}\middle|\mathbf{X}_t = \mathbf{x}\right].$$

Thus, if a predictor is trained on samples of the discounted payoff with the underlying state process as feature set, the predictor will converge towards the price of the derivative. This idea is often used in the context of pricing American options and dates back to at least [37]. In the paper, polynomials are used as parametric family, but it is equally viable to use more precise parametric families such as neural networks instead.

The pricing function is not of primary interest as the Δ is the relevant quantity to estimate for hedging purposes (see chapter 4.1). The method is therefore ultimately indirect in the sense that the predictor must be differentiated after training in order to compute the sensitivities required to replicate the claim. A more direct approach is available.

5.2 The Adjoint Method

Consider now the specific regression problem where the labels $\mathbf{Y} \in \mathbb{R}^p$ are generated through simulations involving $\mathbf{X} \in \mathbb{R}^d$. The labels can then be considered a transformation $\mathbf{Y} = F\left(\mathbf{X}\right)$ for some function $F : \mathbb{R}^d \longrightarrow \mathbb{R}^p$. It is then possible to compute the *pathwise differentials* or *adjoints* by differentiating F,

$$\overline{\mathbf{X}} \triangleq \nabla_{\mathbf{x}} F\left(\mathbf{X}\right) \in \mathbb{R}^{p\times d}.$$

In the context of derivative pricing this is of particular interest as the adjoints are (under regulatory conditions) unbiased estimates of the Δ of the derivative. To see this, let F be the transformation from $\mathbf{X}_t$ to the discounted payoff of the derivative, then

$$\Delta\left(t,\mathbf{x}\right) = \nabla_{\mathbf{x}} E^{Q}\left[\frac{Z}{B_T}\middle|\mathbf{X}_t = \mathbf{x}\right] = E^{Q}\left[\nabla_{\mathbf{x}} F\left(\mathbf{x}\right)\right]. \tag{5.2.1}$$

The necessary requirements for the interchange of derivative operator and expectation are presented in [38, ch. 7.2.2]. The primary concern is whether or not the payoff is continuous in the underlying instrument. This importantly rules out derivatives with discontinuous payoffs such as the binary option, but that can be handled by *fuzzy logic* [39]. The idea of using *adjoints* as unbiased estimates of Δs seems to stem from [40].

The same principle as with the least squares method can naturally be applied to train a predictor $\overline{h} : \mathbb{R}^d \longrightarrow \mathbb{R}^{p\times d}$ to approximate the Δ of the derivative. The only difference is that the labels $\mathbf{Y}$ are

replaced by the adjoints $\overline{\mathbf{X}}$. More formally, if we define the *adjoint error* as[1]

$$\overline{e}\left(\mathbf{X}, \overline{\mathbf{X}}\right) \triangleq \operatorname{vec}\left(\overline{h}\left(\mathbf{X}\right) - \overline{\mathbf{X}}\right) \in \mathbb{R}^{pd},$$

we can proceed exactly as in the beginning by approximating $\overline{h}$ with a parametric predictor $\overline{h}_\theta$ and minimising the empirical estimate

$$\min_{\theta \in \Theta} \frac{1}{m} \operatorname{tr}\left[\overline{e}\left(\mathbf{X}, \overline{\mathbf{X}}\right)^\top \overline{e}\left(\mathbf{X}, \overline{\mathbf{X}}\right)\right].$$

The idea of an adjoint is quite abstract, so to concretise ideas a quick example is now presented. Consider the Black and Scholes model with no rates and a call option with strike K. The feature is then the initial spot price S_0 and the label is the payoff $(S_T - K)^+$. Applying the chain rule yields

$$\overline{S}_0 = \frac{d\left(S_T - K\right)^+}{dS_T} \frac{dS_T}{dS_0} = 1_{(S_T > K)} \frac{S_T}{S_0}.$$

Thus computing the adjoint in this simple example is straightforward and requires no extra computational cost other than a simulation of S_T. This is not necessarily the case for more complicated derivatives and models. However, it is possible to leverage the theory of *Automatic Algorithmic Differentiation (AAD)* in order to compute the adjoints numerically within the same precision as the Euler scheme used to generate the state process. This is the topic of chapter 5.4.

5.3 Adjoints as Unbiased Regularisation

Instead of choosing between regressing on discounted payoffs or directly on adjoints, the authors of [6] suggest minimising a linear combination of the two instead,

$$\min_{h \in \mathcal{H}} E\left[\lambda e\left(\mathbf{X}, \mathbf{Y}\right)^\top e\left(\mathbf{X}, \mathbf{Y}\right) + (1-\lambda)\, \overline{e}\left(\mathbf{X}, \overline{\mathbf{X}}\right)^\top \overline{e}\left(\mathbf{X}, \overline{\mathbf{X}}\right)\right],$$

with the important detail that $\overline{h}$ is now replaced by $\nabla h_{\mathbf{x}}$ which is the actual gradient of the predictor h and not simply a separate predictor. The above problem maintains the same minimum irrespective of the choice of $\lambda \in [0, 1]$ as long as the adjoints are unbiased estimates of the gradient.

Using a parametric approximator h_θ and extending the variables with a batch dimension of size m yields the approximate problem

$$\min_{\theta \in \Theta} \left\{ \frac{\lambda}{m} \operatorname{tr}\left[e\left(\mathbf{X}, \mathbf{Y}\right)^\top e\left(\mathbf{X}, \mathbf{Y}\right)\right] + \frac{1-\lambda}{m} \operatorname{tr}\left[\overline{e}\left(\mathbf{X}, \overline{\mathbf{X}}\right)^\top \overline{e}\left(\mathbf{X}, \overline{\mathbf{X}}\right)\right] \right\}. \tag{5.3.1}$$

The challenge in the problem lies in the computation of the adjoint error $\overline{e}$ as it involves the gradient $\nabla_{\mathbf{x}} h_\theta\left(\mathbf{X}\right)$. This is accomplished by first computing a forward pass to compute $h_\theta\left(\mathbf{X}\right)$ and subsequently backpropagating the operations using AAD to compute the gradient. The values are utilised to compute the empirical error in equation (5.3.1), and then *the entire procedure is backpropagated once again* in accordance with chapter 3.4 to compute the gradient with respect to θ.

The method draws many similarities to classical regularisation, where a second term is added to the cost function controlled by some hyperparameter. The second term usually punishes large weights such as in Tikhonov regularisation, which helps prevent overfitting of noisy data [41]. This punishment effectively alters the minimum of the problem, which introduces a bias in the optimal solution. The present setup is therefore superior in the sense that the regularisation does not introduce a bias as long

[1] The vectorisation operator: $\operatorname{vec}\left(\mathbf{A}\right) \triangleq \mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_m$ for some $m \times n$-matrix where $\mathbf{A}_i$ is the ith row.

as h_θ can approximate the optimal solution.

Inspired by [42], a sensible choice of the hyperparameter λ is arguably to take as neutral a stance as possible. Such a stance could be, that we wish for the two terms to weight equally. This is trivially achieved by setting the two terms equal and solving for λ,

$$\frac{1-\lambda}{\lambda} = \frac{E\left[e\left(\mathbf{X},\mathbf{Y}\right)^\top e\left(\mathbf{X},\mathbf{Y}\right)\right]}{E\left[\overline{e}\left(\mathbf{X},\overline{\mathbf{X}}\right)^\top \overline{e}\left(\mathbf{X},\overline{\mathbf{X}}\right)\right]}.$$

The term on the right hand side can be numerically estimated. The estimate will change during training as the distribution of the output of h_θ will change. One possibility is therefore to continuously update the parameter during training. However, as the primary motivation for the choice of λ is to ensure that the training comes off to a good start, estimating the right hand side before training and keeping λ fixed throughout should suffice.

As with all other statistical learning problems involving neural networks, much benefit can be drawn from proper preprocessing of the data. The addition of adjoints superimposes a complication in this regard. Suppose the state process $\mathbf{X}$ and payoff $\mathbf{Y}$ each are preprocessed through some linear transformation (e.g. a normalisation),

$$\widehat{\mathbf{X}} \triangleq \mathbf{A}\mathbf{X} + \mathbf{b}, \qquad \widehat{\mathbf{Y}} \triangleq \mathbf{C}\mathbf{Y} + \mathbf{d},$$

where $\mathbf{A} \in \mathbb{R}^{d\times d}$ and $\mathbf{C} \in \mathbb{R}^{p\times p}$ are invertible matrices. After proprocessing the objective of learning is to find the function $\widehat{\mathbf{x}} \mapsto \widehat{h}\left(\widehat{\mathbf{x}}\right)$ which is the conditional expectation of $\widehat{\mathbf{Y}}$ given $\widehat{\mathbf{X}}$,

$$\widehat{h}\left(\mathbf{A}\mathbf{x}+\mathbf{b}\right) = E\left[\mathbf{C}\mathbf{Y}+\mathbf{d}\middle|\widehat{\mathbf{X}} = \mathbf{A}\mathbf{x}+\mathbf{b}\right].$$

Assuming that $\overline{\mathbf{X}}$ is an unbiased estimator of the gradient as in equation (5.2.1), it follows from the multivariate chain rule that

$$\nabla_{\widehat{\mathbf{x}}}\widehat{h}\left(\widehat{\mathbf{x}}\right) = E\left[\mathbf{C}\overline{\mathbf{X}}\mathbf{A}^{-1}\middle|\widehat{\mathbf{X}} = \widehat{\mathbf{x}}\right].$$

The proprocessing of the adjoints must therefore necessarily be a multiplication of $\mathbf{C}$ from the left and the inverse of $\mathbf{A}$ from the right. For the remainder of the chapter it is always assumed the the inputs to the networks are appropriately normalised.

5.3.1 Analytical Solution in Linear Regression

In [6], it is noted that the solution of the problem in equation (5.3.1) remains analytical for univariate linear regression. We shall employ this solution in chapter 5.7.1 and so it is derived here. Let us therefore once again transform the features through a feature map $\xi\left(\mathbf{X}\right) \in \mathbb{R}^{m\times q}$ for some sample size m and $q \geqslant 1$ depending on the specific transformation and denote the differentiated feature map with respect to the kth feature as $\overline{\xi}_k\left(\mathbf{X}\right) \in \mathbb{R}^{m\times q}$. Denote likewise the adjoint of $\mathbf{y}$ with respect to the kth feature as $\overline{\mathbf{X}}_k$. The problem from equation (5.3.1) can be written as

$$\min_{\beta\in\mathbb{R}^q}\left\{\lambda\left(\xi\left(\mathbf{X}\right)\beta-\mathbf{Y}\right)^\top\left(\xi\left(\mathbf{X}\right)\beta-\mathbf{Y}\right)+(1-\lambda)\sum_{k=1}^{d}\left(\overline{\xi}_k\left(\mathbf{X}\right)\beta-\overline{\mathbf{X}}_k\right)^\top\left(\overline{\xi}_k\left(\mathbf{X}\right)\beta-\overline{\mathbf{X}}_k\right)\right\}. \tag{5.3.2}$$

The above expression is once again easily differentiated with respect to β, which yields the first order conditions,

$$\lambda\xi\left(\mathbf{X}\right)^\top\left[\xi\left(\mathbf{X}\right)\beta-\mathbf{y}\right]+(1-\lambda)\sum_{k=1}^{d}\overline{\xi}_k\left(\mathbf{X}\right)^\top\left[\overline{\xi}_k\left(\mathbf{X}\right)\beta-\overline{\mathbf{X}}_k\right]=\mathbf{0}.$$

The equation can be written in a similar form as the with regular linear regression. Fundamentally, we are looking for a parameter vector $\beta \in \mathbb{R}^q$ that solves the system of linear equations,

$$\left(\lambda \xi\left(\mathbf{X}\right)^{\top} \xi\left(\mathbf{X}\right) + (1-\lambda) \sum_{k=1}^{d} \overline{\xi}_k\left(\mathbf{X}\right)^{\top} \overline{\xi}_k\left(\mathbf{X}\right)\right) \beta = \lambda \xi\left(\mathbf{X}\right) \mathbf{y} + (1-\lambda) \sum_{k=1}^{d} \overline{\xi}_k\left(\mathbf{X}\right) \overline{\mathbf{X}}_k. \tag{5.3.3}$$

No condition for the existence of a solution is provided in the original paper, so we shall derive a sufficient condition now. Note that the proposition guarantees that if ξ is chosen as monomials, then a unique solution exists.

Proposition 5.3.4. *If just a single one of the matrices $\xi\left(\mathbf{X}\right)$ or $\xi_k\left(\mathbf{X}\right)$ for $k = 1, \ldots, d$ contains linearly independent columns, then a unique solution of equation* (5.3.2) *exists.*

Proof. A matrix of the form $\mathbf{A}^{\top}\mathbf{A}$ is guaranteed to be positive semi-definite. It is further positive definite if and only if $\mathbf{A}$ has linearly independent columns [43, ch. A.4]. A sum of a positive definite matrix and a positive semi-definite matrix is positive definite. Thus the entire matrix on the left hand side of equation (5.3.3) is positive definite and the proof is complete. □

The family of linear predictors h_β almost certainly cannot represent the conditional expectation of $\mathbf{Y}$ given $\mathbf{X}$, so using adjoints as regularisation does introduce a bias into the solution. Of course, if h_β could represent the conditional expectation there would be no reason to use regularisation due to the problem being convex.

5.4 Algorithmic Computation of Adjoints

In the influential paper [44], the theory of AAD is utilised in the context of simulations of derivative payoffs. The theory is presented here due to its importance in generalising the method to arbitrary models. Restrict the setting from chapter 2.1 to Markovian processes of the form

$$d\mathbf{X}_t = \mu\left(t, \mathbf{X}_t\right) dt + \Sigma\left(t, \mathbf{X}_t\right) d\mathbf{W}_t,$$

where μ and Σ are functions, not processes. The state process is d-dimensional and the Brownian motion is p-dimensional. The state process $\mathbf{X}_t$ encompasses the set of tradeable assets $\mathbf{S}_t$ and B_t, but can also contain non-tradable processes such as volatility. In Monte Carlo simulations the typical approach to simulating such a process is through the Euler discretisation

$$\mathbf{X}_{t_{k+1}} = \mathbf{X}_{t_k} + \mu\left(t_k, \mathbf{X}_{t_k}\right) \Delta t_{k+1} + \Sigma\left(t_k, \mathbf{X}_{t_k}\right) \sqrt{\Delta t_{k+1}} Z_{k+1}, \quad k \in \{0, \ldots, n-1\} \tag{5.4.1}$$

where the recursion is instantiated at some deterministic value $\mathbf{X}_{t_0}$. The simulation is merely an approximation of a sample path of the state process, so the notation $\widehat{\mathbf{X}}_{t_k}$ would be more correct. However, this is omitted.

The object of interest is the derivative of some contingent claim Z with respect to the underlying state process $\mathbf{X}_{t_k}$. The method works for arbitrary path dependent derivatives, so let the discounted payoff of the derivative Z be a deterministic transformation of the state process path,

$$\frac{Z}{B_T} = g\left(\mathbf{X}_{t_0}, \ldots, \mathbf{X}_{t_n}\right),$$

where it is important to remember that all variables after time t_k depends on the state at time t_k. The adjoint is once again defined as the derivative of Z with respect to the value of the state process at time t_k,

$$\overline{\mathbf{X}}_{t_k} \triangleq \nabla_{\mathbf{x}_k}\left(\frac{Z}{B_T}\right) \in \mathbb{R}^{1\times d}.$$

The notation $\mathbf{x}_k$ refers to the derivative with respect to $\mathbf{X}_{t_k}$ where Z/B_T is only considered a function of $\mathbf{X}_{t_k}$. The important idea behind algorithmic adjoint differentiation is that the adjoints can be written as a *backwards* recursive equation by utilising the multivariate chain rule [16, ch. 6.5.2] multiple times,

$$\overline{\mathbf{X}}_{t_k} = \sum_{i=k}^{n} \underbrace{\nabla_{x_i} g}_{1\times d} \underbrace{\nabla_{\mathbf{x_k}} \mathbf{X}_{t_i}}_{d\times d} = \sum_{i=k}^{n} \nabla_{x_i} g \nabla_{\mathbf{x}_{i-1}} \mathbf{X}_{t_i} \cdots \nabla_{\mathbf{x}_k} \mathbf{X}_{t_{k+1}} = \overline{\mathbf{X}}_{t_{k+1}} \nabla_{\mathbf{x}_k} \mathbf{X}_{t_{k+1}}, \tag{5.4.2}$$

The derivation of this backwards recursive equation is identical to the derivation of the backpropagation algorithm from chapter 3.3. The above implies that if the derivative of $\mathbf{X}_{t_{k+1}}$ with respect to $\mathbf{X}_{t_k}$ is available then so is the adjoints. Differentiating equation (5.4.1) on both sides of the equality yields the recursive formula for $i, j = 1, \ldots, d$,

$$\left[\nabla_{\mathbf{x}_k} \mathbf{X}_{t_{k+1}}\right]_{i,j} = \frac{\partial \mathbf{X}^{(i)}_{t_{k+1}}}{\partial \mathbf{X}^{(j)}_{t_k}} = 1_{i=j} + \frac{\partial \mu_i}{\partial x_j}\left(t_k, \mathbf{X}_{t_k}\right) \Delta t_{k+1} + \sum_{\ell=1}^{p} \frac{\partial \Sigma_{i,\ell}}{\partial x_j}\left(t_k, \mathbf{X}_{t_k}\right) \sqrt{\Delta t_{k+1}} Z^{(\ell)}_{k+1}. \tag{5.4.3}$$

Assuming that the derivatives of μ and Σ are available, the above expression is known as long as the values $\mathbf{X}_{t_k}$ and Z_{k+1} are known. In other words, a general algorithm for deriving adjoints are as follows: first a *forward pass* of equation (5.4.1) is conducted where the relevant values at each point in time is saved. Secondly a *backwards pass* of equation (5.4.2) is conducted, where the partial derivatives in equation (5.4.3) are computed at each point in time.

The derivation above relies on two unfortunate assumptions: the dynamics of the state process must be of the particular form in equation (5.4.1) which rules out more complicated path dependent processes and the derivative of μ and Σ were assumed known in equation (5.4.3). The fundamental idea behind AAD is that all computations can fundamentally be broken down into various operations on elementary functions which are all differentiable. The ideas used above can therefore be generalised to compute gradients of arbitrary (differentiable) transformations.

5.5 Test of Approximation Performance

Before venturing into replication algorithms it must first be investigated whether or not the methods described in the previous chapters can reliably reproduce the Δ of a derivative book. While [6] did perform several analyses of their method's performance in multivariate settings, there was no investigation of the *evolution* of the error as the dimension was increased. This chapter also contains an investigation of a network solely trained on adjoints, and additionally a new type of derivative book is utilised.

Once again, the performance of the three methods is investigated for derivative books of increasing dimension. Two derivative books are considered: one with d at-the-money call options and one with d butterfly spreads. Each of the derivatives is written on one of the d underlying instruments and the maturity is thirteen weeks. The exposure to each derivative is d^{-1} so as to keep the Δs of similar magnitude. The butterfly spread is included to ensure that the activation function of the network is vastly different from the pricing function of the derivative[2].

The instrument process is the Black and Scholes model with spot values normalised to one hundred. The correlation structure of the d-dimensional Brownian motion is randomly generated in such a way that the volatility of each instrument lies uniformly distributed between 20% and 30%. The interest rate is 2% and the process is simulated under the Q measure.

[2]The price of a call option and softplus activation function are very alike which could aid significantly in training.

The training data is generated as follows: for some initial value of the state process $\mathbf{X}_0 \in \mathbb{R}^d$ we compute some sample of starting values of size m. That is, we simulate[3] m different initial values and collect them in a matrix $\mathbf{X}_0 \in \mathbb{R}^{m \times d}$. For each of the m initial values, an associated terminal value $\mathbf{X}_T \in \mathbb{R}^{m \times d}$ is simulated according to the underlying dynamics of the model. The terminal values are then transformed into a payoff of the derivative book $\mathbf{Y} \in \mathbb{R}^m$. Then the adjoints of the payoff $\overline{\mathbf{X}}_0 \in \mathbb{R}^{m \times d}$ are computed either analytically if possible or through the algorithm described in chapter 5.3. With the three matrices $\mathbf{X}_0$, $\mathbf{Y}$ and $\overline{\mathbf{X}}_0$ available we are ready to train the three types of networks.

1. The *payoff network* is a feedforward network with a d dimensional input and one dimensional output. The network is trained with $\mathbf{X}_0$ as feature set and $\mathbf{Y}$ as label set. As explained in chapter 5.1, the network should converge towards the value function of the derivative book. An approximation of the Δ of the book can then be obtained by computing the derivative of the network using AAD.

2. The *adjoint network* is a feedforward neural network with a d dimensional input and d dimensional output. The network is trained with $\mathbf{X}_0$ as feature set and $\overline{\mathbf{X}}_0$ as label set. As explained in chapter 5.2, the network should converge towards the Δ of the derivative book. This network is not part of the analysis in [6].

3. The *twin network* is a feedforward neural network with a d dimensional input and one dimensional output. The network is trained with $\mathbf{X}_0$ as feature set. The label set consists of both $\mathbf{Y}$ and $\overline{\mathbf{X}}_0$ in order to minimise the objective function in equation (5.3.1). The network should converge towards the value function of the derivative book. An approximation of the Δ of the book can then be obtained by computing the derivative of the network using AAD.

As in [6], we fix the network architectures at four layers and twenty units. The activation function for the payoff- and twin network is the softplus function and the activation function for the adjoint network is the sigmoid function[4].

The Euclidean distance is the natural choice of performance measure as we are mostly concerned about large deviations from the true pricing function. As the exposure in the derivative book is appropriately scaled by dimension it should facilitate reasonable comparison across dimensions. Formally, the performance measure for m test samples is

$$\frac{1}{m} \sum_{i=1}^{m} \left\| \nabla_{\mathbf{x}} h_\theta \left(\mathbf{x}^{(i)} \right) - \Delta \left(\mathbf{x}^{(i)} \right) \right\|.$$

The performance measure is computed on 2^{17} (128k) test samples.

Facilitating a fair comparison between the three methods is difficult as they are exposed to different amounts of data. For m samples and d dimensions the payoff network is exposed to md features and m payoffs. The twin network is exposed to md features, m payoffs and md adjoints and the adjoint network is exposed to md features and md adjoints. One possible approach is to use m samples for the twin network and then for the payoff and adjoint network respectively,

$$\frac{2d+1}{d+1} m \quad \text{and} \quad \frac{2d+1}{2d} m.$$

[3] The initial values are drawn from a truncated log-normal distribution with mean $\mathbf{X}_0$ and standard deviation of around 15% of the initial value.

[4] This is the natural choice as the derivative of the softplus function is the sigmoid function.

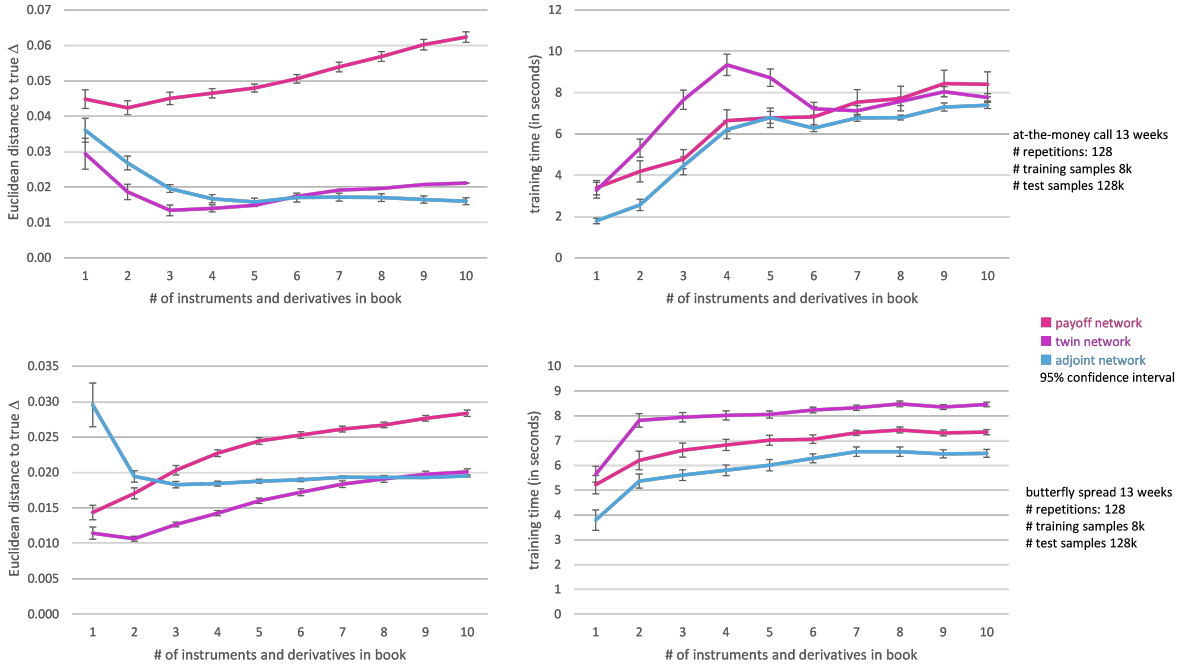


Figure 5.1: Results of the experiment described in chapter 5.5

Another approach is to recognise that construction of the features, payoffs and adjoints differs in time complexity during simulation and one should therefore aim to keep the time constant for construction of each data set. This is the method utilised in [6] in which the payoff network gets exposed to four times as much data as the twin network.

A third approach is to recognise that the training time differs between the three methods. As the bottleneck in terms of computational time is the training time this seems like the most reasonable aspect to keep constant and is therefore the method utilised in this chapter. We keep the amount of data equal in all three methods at 2^{13} (8k) samples and measure the computational time. The experiment is repeated 128 times and an average is computed over the measurements.

The results illustrated for $d = 1, \ldots, 10$ in figure 5.1. Beginning with the call test (top row), it is clear that the payoff network is not a viable option. The error grows linearly in the dimension and it is at no point close in performance to the other two network types despite the training time being similar.

The twin and adjoint network showcase similar performance with the twin network being slightly more precise at low dimensions and the adjoint network being slightly more precise for high dimensions. The error seems to stabilise at a level supirior to the payoff network as the dimension increases. The training time of the adjoint network is substantially shorter for low dimensions, but similar for higher dimensions.

For the butterfly test (bottom row), the twin network is consistently the best in terms of precision of all three network types. It does suffer from a deterioration in performance as the dimension increases, but it is not as severe as for the payoff network. The network has the slowest training time which is unsurprising as it involves the most data and requires backpropagation before computing the loss.

The error in the payoff network grows quickly as the dimension increases and, as opposed to the two other networks, it does not seem to flatten at any point. The network is barely faster to train than the

twin network, so there is no reason to prioritise it.

The adjoint network behaves more strangely than the others. It is consistently the fastest to train despite being the only one with a multidimensional output. For low dimensions, it is much worse than the two other networks, but it quickly stabilises and the performance is henceforth relatively unaffected by dimension. For high dimensions, it performs as well as the twin network despite being around two seconds faster to train.

5.6 Test of Replication Performance

The next step involves testing the three types of networks in a hedging context. A similar study is carried out in [42] where, amongst others, the performance of the twin network is compared relative to the payoff network. The paper examined the hedge error for a call option in the Bachelier model and found that the twin network was vastly superior.

This chapter extends the results of the paper in similar ways as before. The most important extension is as usual that the performance is tested for multivariate books. The two types of derivative books are identical to the ones utilised in chapter 5.5. The underlying instrument is likewise modelled exactly as in chapter 5.5.

The training procedure is altered as a new network must be trained at each readjustment date. Some collection of size m of starting values of the state process is simulated and stored in a matrix $\mathbf{X}_0 \in \mathbb{R}^{m\times d}$. For each sample, a full path of the state process is simulated at the discrete time points $\mathbf{X}_{t_k} \in \mathbb{R}^{m\times d}$ for $k = 0, \dots, n$. These paths can be converted into a sample of payoffs $\mathbf{Y} \in \mathbb{R}^m$ and adjoints $\overline{\mathbf{X}}_{t_k} \in \mathbb{R}^{m\times d}$. The kth network is then trained on the collection of samples $\left(\mathbf{X}_{t_k}, \mathbf{Y}, \overline{\mathbf{X}}_{t_k}\right)$ according to the specific type of network.

Once n trained networks h_k for $k = 0, \dots, n-1$ are available the training samples are discarded and a new collection of sample paths are simulated beginning from the initial value $\mathbf{X}_0 \in \mathbb{R}^d$. The paths are once again simulated at each point in time, but no adjoints are computed. This new collection of training data $\mathbf{X}_{t_k} \in \mathbb{R}^{\ell\times d}$ and $\mathbf{Y} \in \mathbb{R}^\ell$ is then utilised to conduct a hedge experiment where the portfolio strategy $\mathbf{a}_{t_k}$ is derived from the network $h_k\left(\mathbf{X}_{t_k}\right)$ depending on the specific type.

The same three types of network architectures as in chapter 5.5 are used for the tests. The preprocessing and training optimisations are also kept identical. The networks are trained on 8k samples and subsequently used as portfolio strategy on 2^{18} (256k) sample paths.

In terms of performance measurement two types of measures are needed. On the one hand we are interested in whether or not the networks achieve the same wealth distribution as regular Δ hedging *on average*. However, from a risk perspective it is the *tail behaviour* that is interesting. It is therefore both relevant to measure the percentage difference between the average terminal wealth and 95% expected shortfall relative to Δ hedging. As *any* deviation from Δ hedging is unsatisfactory, the average is taken over the absolute value of the measurements.

The results are depicted in figure 5.2. For the call test (top row) there is little doubt that the payoff network deviates too strongly from the true Δ to be used for reliable Δ hedging. Both the expected shortfall (top left) and average wealth (top right) grow linearly as the dimension increases. However, it should be mentioned that the deviation in average wealth is generally very small for all three network types.

The adjoint network also seems highly affected by the dimension, but the pattern is strangely oscillating. In general the risk (top left) deviates at around 3% from the analytical Δ hedge. The difference in average wealth (top right) is also by far the highest in the univariate setting and generally worse than

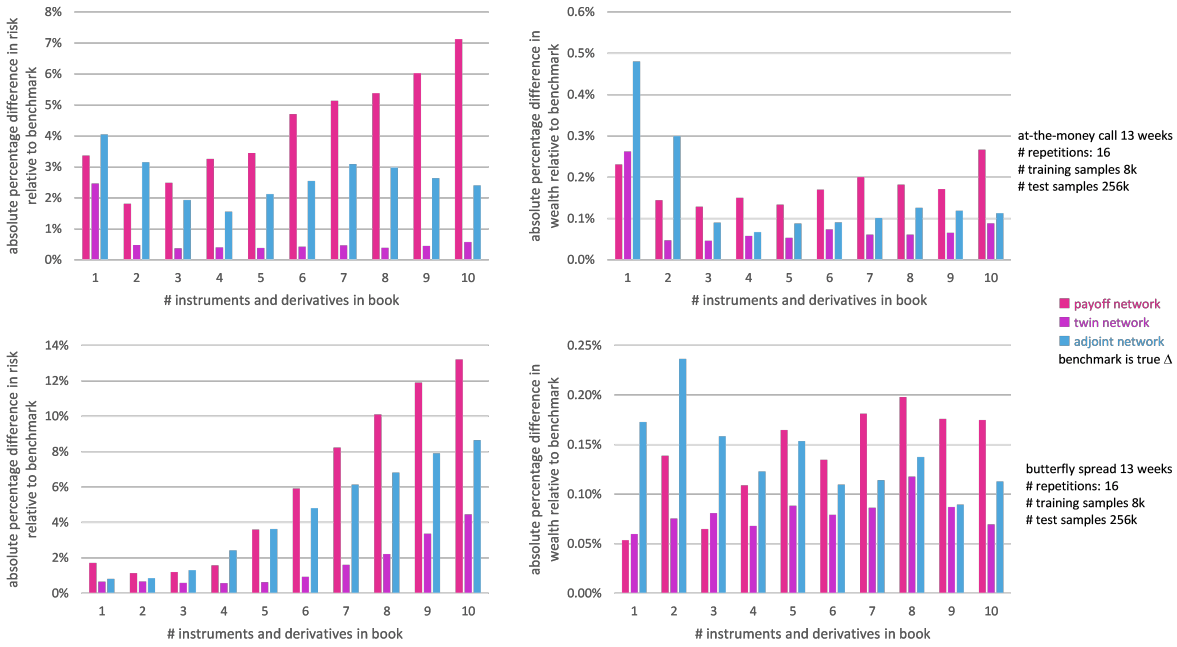


Figure 5.2: Results of the experiment described in chapter 5.6

the twin network.

The twin network is unmistakably superior to the other two types. For dimensions higher than two, the deviation in risk (top left) remains fixed at around 0.3%. The performance in both risk and average wealth generally seems completely unaffected by dimension and is consistently the lowest. The deviation in risk and average wealth is much higher for one dimension, but this behaviour is also found for the two other network types.

For the butterfly test (bottom row) there is a consistent increase in risk (bottom left) for all three network types. The deviation grows the fastest for the payoff network and the slowest for the twin network. For the twin network, the growth in deviation is relatively stagnant at 0.5% for the first five dimensions and only starts increasing at the higher dimensions.

The deviation in average wealth (bottom right) generally grows with the dimension for the payoff network albeit the measurements are subject to quite a bit of noise. The opposite can be said for the adjoint network where the deviation seems to decrease. Once again the twin network is most stable to dimension increases as there does not seem to be a general growth in error.

5.7 An Alternative Method for Risk Optimal Hedging

As we saw in chapter 4.5, finding the risk optimal portfolio strategy by parameterising it with a neural network requires long training times in multivariate settings. The solution is also impossible to interpret. In this chapter the ideas from [5] and [6] are combined to remedy the issues. The idea does not seem to have been investigated anywhere before now.

As already touched upon in the original discussion of figure 4.2 the simple affine transformation of the continuous time Δ is vastly superior to the complex neural networks for high dimensions. They were

orders of magnitude faster to train and the solution is an easily interpretable affine transformation of the Δ. The obvious issue is that analytical solutions of Δs are rare and certainly not available for general derivative books. However, as we have seen in chapter 5.5, utilising a twin network yields precise approximations of Δs for general derivative books. The approximation scales to higher dimensions without introducing further computational time nor significant error. A natural idea is therefore to compute the Δ using a twin network and then subsequently transforming this approximation using the linearised Δ strategy from chapter 3.2.

This chapter contains an investigation of this exact idea. Once again a d dimensional instrument process is simulated each week over thirteen weeks. The derivative book consists of d butterfly spreads (as they were the most difficult to predict) with the jth derivative written on the jth instrument. Trading the instrument is now subject to proportional transaction costs of 1%. The d dimensional instrument process is simulated exactly as in the previous two chapters.

At first a training set consisting of 8k sample paths with adjoints is simulated under the risk neutral measure. A twin network of four layers and twenty units is subsequently trained to approximate the Δ of the derivative book at each point in time. Hopefully this approximation is stable across dimensions.

A new training set consisting of 256k sample paths is simulated under the statistical measure. The Δ at each point in time is computed using the approximation from the network. The following three types of hedge strategy predictors are trained on the training set,

1. *deep network w. softplus* refers to the same type of network as in the experiments from chapter 4.5 with the softplus activation function.

2. *deep network w. relu* refers to the same type of network as in the experiments from chapter 4.5 with the relu activation function.

3. *feature map w. approximated* Δ refers to the linearised Δ strategy from chapter 3.2 with the aforementioned twin network approximated Δ.

The training time of each predictor is also recorded. As the twin network requires an initial training this training time is of course added to the total training time. The portfolio strategy predictors are trained with the exact same settings as in chapter 4.5.

The results are illustrated in figure 5.3. The evolution of performance is clear. For the first dimension the deep networks performs markedly better than the twin network approximation. After dimension two the deep networks starts lacking significantly behind and at four dimensions the performance is worse than regular Δ hedging for both activation functions. This is consistent with the findings from chapter 4.5. The strategy utilising the approximated Δ is however completely robust to dimensionality increases and in fact seems to *increase* its performance as the dimension grows. The total training time of the feature map w. approximated Δ is half that of the deep networks.

The feature map w. approximated Δ is clearly superior to the deep networks in terms of risk performance as well as training time. By separating the computation of the Δ and risk optimisation into two distinct optimisation problems, the approximation of the risk optimal strategy becomes much better. Furthermore, the solution is interpretable as an affine transformation of the Δ removing the *black box* aspect of the deep network.

5.7.1 Increasing Speed with Linear Regression

If speed is of the essence, there is a faster alternative to utilising the twin network to approximate the Δ in the univariate setting. As the bottleneck in terms of computational time is the training of the twin network, one could replace the parametric family by a linear predictor which is polynomial in the

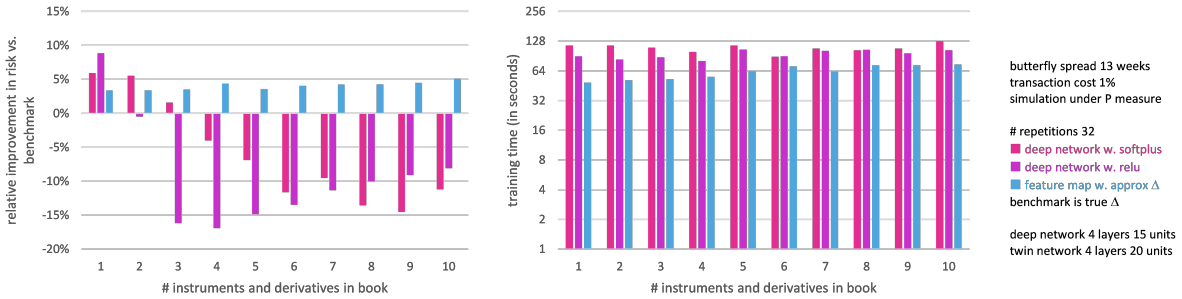


Figure 5.3: Results of experiment described in chapter 5.7.

features. As we saw in chapter 5.3.1 there exists an analytical solution for linear regression with adjoints, so the entire first training step is essentially skipped.

While the precision of the Δ approximation will deteriorate it might not be too important when transaction costs are present, as the approximated Δ is linearly transformed during the subsequent application of the linear transformation from chapter 3.2.

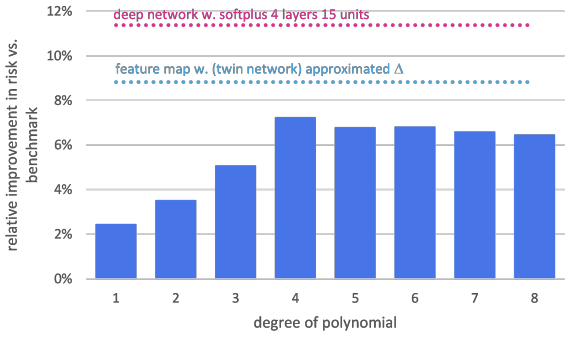


Figure 5.4: Results of the experiment described in chapter 5.7.1.

To test the method, we will conduct a similar experiment as before with an at-the-money call option and a polynomial linear predictor of various degrees. The results are illustrated in figure 5.4. The entire procedure requires between seven and nine seconds of training which is around an order of magnitude faster than utilising the deep network as parametric family. The method does unsurprisingly not reach the level of the deep network or twin network feature map, but the best performing polynomial does manage to improve upon Δ hedging by over 7%. In other words, we have developed a method to compute model-free, interpretable portfolio strategies which improve upon Δ hedging and requires less than ten seconds to train.

Chapter 6

Hedging with Jump Risk

Recent research suggests that similarities between hedging under jumps and rough volatility can be drawn [45]. The authors intended to study the approximation algorithm from chapter 4.2 in the context of hedging in rough volatility models, but quickly found that the model behaved as if incomplete. As rough volatility models resemble jump models when discretised, they suggest that rough volatility models should be hedged with options. They were unable to test the hypothesis as that would require computing the value of millions of options which is not yet feasible in rough volatility models. However, the simpler question of whether the algorithm can handle hedging with options under jump risk also remains unanswered.

Further, the idea of utilising options as hedge instruments for non-jump models is also relevant for hedging of options with high Γ such as barriers. While the diffusion risk can theoretically be eliminated by Δ hedging, the time discretisation bias is too vast when the underlying process is close to the barrier. Both dynamic [7, ch. 9.3] and static [46] hedging strategies utilising options exist for handling this issue.

This short chapter showcases how to adapt the performance of the algorithm from chapter 4.2 in the presence of jump risk in the underlying instrument process. As hedging under jumps is significantly more complex than Δ hedging, some work is required to develop an appropriate benchmark against which to test the algorithm. The presence of jumps also renders the model incomplete thus fundamentally distinguishing it from the simpler setting of chapter 4.5.

6.1 Continuous Time Hedging under Jump Risk

A jump is a discontinuous movement in the underlying instrument process such that no trading is possible during it. The empirical evidence for the presence of jumps in various instruments is quite strong and jump models further allows for a possible explanation of short-term skews in the volatility surface [47, ch. 5].

We shall follow the approach of [48] and model the jumps as a compound Poisson process. The Black and Scholes model is augmented with a jump component as follows,

$$dS_t = (\alpha - \lambda\kappa)\, S_t dt + \sigma S_t dW_t^P + (J_t - 1)\, S_t dN_t.$$

The bank account is assumed to provide the riskless rate of return r. The parameter λ is the intensity of the compound Poisson process and κ is the expected value of $J_t - 1$ under Q. As such, when modelling under Q, we must enforce $\alpha = r$ as S_t/B_t would otherwise not be a martingale. We will make the rather unrealistic assumption that the distribution of the jump size J_t and Poisson process N_t is unchanged when moving from P to Q. Simulation of the process is achieved at relatively minuscule extra computational cost especially if the jump process is assumed to follow a log-normal distribution [38, ch. 3.5.1].

The Brownian motion is often seen to model the systematic risk of the instrument such as disequilibrium between supply and demand or small changes in economic outlook, whereas the compound Poisson process is seen to model the non-systematic risk such as arrival of important new information about the instrument or major changes in the economy. As such, a reasonable assumption is that these two components are independent. The compound Poisson process is also assumed independent of the jump size J_t for convenience. The economic justification for this is arguably less clear.

We consider constructing a portfolio consisting of $a_t \in \mathbb{R}$ units of the underlying instrument and $\xi_t \in \mathbb{R}^d$ units of d additional hedging instrument $\mathbf{I}_t \in \mathbb{R}^d$. The additional instruments are crucially assumed to depend explicitly on the underlying instrument such that their dynamics depend on the jump process. The agent is assumed short an underlying derivative with value process V_t and she wishes to hedge the risk of the liability. In appendix B.1 it is shown that the value process of the portfolio must satisfy

$$\begin{aligned} d\left(\frac{V_t}{B_t}\right) = [\cdots]\, dt + \left[a_t + \xi_t^\top \nabla_s \left(\frac{\mathbf{I}_t}{B_t}\right) - \nabla_s \left(\frac{H_t}{B_t}\right)\right] \sigma S_t dW_t^P \\ + \left[a_t \Delta_J S_t + \xi_t^\top \Delta_J \left(\frac{\mathbf{I}_t}{B_t}\right) - \Delta_J \left(\frac{H_t}{B_t}\right)\right] dN_t^P, \end{aligned} \tag{6.1.1}$$

where e.g. $\Delta_J(H_t/B_t)$ refers to the change in value before and after a jump of size J. The agent will wish to eliminate both the systematic and non-systematic risk. The systematic risk is eliminated by setting a_t as the solution to

$$a_t + \xi_t^\top \nabla_s \left(\frac{\mathbf{I}_t}{B_t}\right) - \nabla_s \left(\frac{H_t}{B_t}\right) = 0. \tag{6.1.2}$$

As for the jump risk, there is no possible way to perfectly hedge it. As the jump size J_t is not observable before it is too late and the jump distribution is assumed continuous, it would require a continuum of hedge instruments to remove the risk entirely [47, ch. 5].

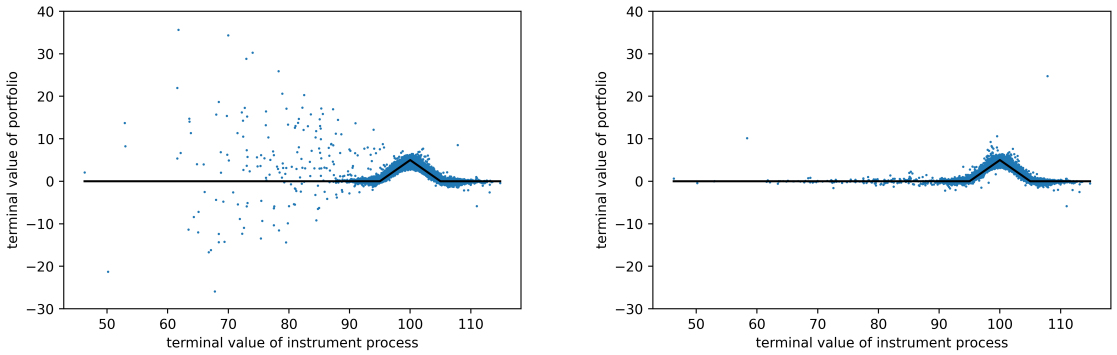
We shall adopt an approximated version of the method suggested in [49] in order to reduce the jump risk. We can approximate the continuum of jump sizes by a discrete distribution taking values in $J_1, \ldots, J_d$. If the holdings in the underlying instrument is chosen in accordance with equation (6.1.2), then the jump risk can be removed by setting ξ_t as the (unique) solution to the linear system of equations,

$$\forall i \in \{1, \ldots, d\}: \quad \left[\frac{\Delta_{J_i}(\mathbf{I}_t/B_t)}{\Delta_{J_i} S_t} - \nabla_s \left(\frac{\mathbf{I}_t}{B_t}\right)\right]^\top \xi_t = \frac{\Delta_{J_i}(V_t/B_t)}{\Delta_{J_i} S_t} - \nabla_s \left(\frac{H_t}{B_t}\right). \tag{6.1.3}$$

The jump sizes are chosen as the inverse cumulative distribution function of J_t evaluated at the first d samples of a one-dimensional Sobol sequence, as this arguably is the best discrete representation one can make of a continuous distribution.

In equation (6.1.2) the agent chooses a_t such that the change in portfolio value due to a non-jump movement exactly offsets the change in value of the derivative. If a jump of size J_i occurs there will be a discontinuous rate of change in both the portfolio and the derivative which is not accounted for by the gradients; ξ_t is chosen to fix this mismatch.

The eigenvalues of the matrix must be kept relatively large if the solution of ξ_t is to be kept within a reasonable range of values. The economic interpretation is that one wants the (average) Γ of the ith instrument to be large during a jump from S_t to $J_i S_t$ and the Γ's of the other instruments to be small. If the Γ is large, then the corresponding (absolute) difference on the left hand side of equation (6.1.3) will also be large. Intuitively, we wish to hold a variety of instrument where the ith instrument is non-linear around $J_i S_t$ and relatively linear everywhere else. Obviously, call and put options are ideal for this



(a) regular Δ hedging

(b) jump risk minimisation

Figure 6.1: Relative value of terminal wealth and payoff of butterfly call spread for two types of portfolio strategies. The x-axis represents the terminal instrument value S_T and the y-axis represents the terminal value of the value process $V_T/B_T + H_0/B_0$ (blue dots) and derivative payoff (black line).

purpose.

To gain an understanding of the importance of hedging jump risk, profit-and-loss plots are constructed for regular Δ-hedging and hedging with the aforementioned jump risk minimisation. The exact details of the procedure is explained in chapter 6.3. For now note that the risk minimisation utilises eight put options as hedging instruments to protect a short position in a butterfly spread.

The performance in illustrated in figure 6.1. Clearly the Δ strategy incurs massive losses for the paths where a large negative jump emerged. Remarkably, the aforementioned procedure seems to eliminate the jump risk almost entirely. However, the strategy requires massive volumes of trading which is unfit under transaction costs. The interesting question is, whether or not the neural networks can locate a better strategy.

6.2 Adapting the Approximation Algorithm

From a theoretical point of view, adapting the algorithm from chapter 4.2 to include derivatives as hedge instruments is straight forward. As the derivatives can simply be considered as part of the instrument process, no alteration to the algorithm is needed. At first, the underlying instrument process is simulated and the price of the hedge instruments are computed for each sample path and time point. The collection of underlying and derivative instruments can then be considered a multivariate instrument process. The multivariate instrument process is then used to hedge some *other* collection of liabilities using the algorithm from chapter 4.2.

It is important to note that the algorithm is now distinctively *not* model free anymore. While the only necessary specification in chapter 4 was the ability to simulate the instrument process, we now must demand that an efficient calculation of the values of the hedge derivatives is available. As the training data must consist of several hundreds of thousands different values of the underlying instrument, so too must we calculate an equivalent amount of derivative prices for each additional derivative hedge instrument.

If some approximation of the values of the derivatives is available it must be guaranteed that the subsequent values are arbitrage free. Even the slightest deviation from perfectly arbitrage-free values

will be picked up by the algorithm and the risk will diverge towards negative infinity.

6.3 Testing the Algorithm in Multivariate Environments

The performances of the algorithms can be tested in a similar manner as for Markovian derivatives. Once again we shall consider daily hedging of a fourteen day maturity butterfly spread written on a univariate underlying instrument. Using a butterfly spread ensures that the liability is sensitive to jumps in both directions and therefore particularly susceptible to jump risk. The parameters of the underlying instrument are

$$T = \frac{14}{250}, \quad S_0 = 100, \quad r = 2\%, \quad \alpha = 5\%, \quad \sigma = 15\%, \quad \lambda = 0.25.$$

The jump process is assumed log-normal with mean -20% and standard deviation 15%. The prices of call- and put options are known in (semi-)analytical form due to this assumption (see appendix B.2).

The performance of the algorithms is tested for an increasing number of additional call- and put hedge instruments to showcase the (hopefully) decreasing jump risk. The strikes of the hedge instruments are chosen as $S_0 J_i$ for $i = 1, \dots, d$ where the jump sizes are chosen as described at the end of section 6.1. As the liability decreases in value for jumps in either direction, the underlying instruments are chosen as puts for strikes below S_0 and calls for strikes above S_0. This is a highly idealised setting, since such options are naturally not necessarily available. The strikes are (successively added from the left as the number of hedge instruments increases),

$$K \in \{82, 91, 74, 78, 97, 86, 69, 72\}.$$

An unrealistic aspect of this experiment is that the strikes of the hedge instruments do not adapt to the value of the underlying instrument across time. If the underlying instrument increases or decreases in value, some of the hedge instruments will be (numerically) worthless or linear. Under natural circumstances different options of different strikes would be available at different points in time and this problem would therefore not subsist. Nevertheless, it still allows us to gauge the predictors' ability to protect a position against jump risk.

The issue it that the benchmark algorithm from chapter 6.1 is not well-suited for this setting as near worthless derivatives results in holdings of several millions which leads to massive losses in discrete time. To remedy this, we only estimate the risk of this particular method with eight available hedge instruments and remove the rows where the difference on the left hand side of equation (6.1.3) is below 0.05. An approximate solution can then be computed using the Moore-Penrose pseudo inverse instead.

Once again the underlying instrument is simulated across a time discretised interval. At each sample path and point in time we compute the value of the underlying hedge instruments. The subsequent $(d+1)$-dimensional process is then treated as if it was a Markovian instrument process from chapter 4.2. The test encompasses four different kinds of portfolio strategies:

1. *deep network* refers to the same type of network as in the experiments from chapter 4.5.

2. *discrete jump risk minimisation* refers to the technique described in chapter 6.1. This method is parameter free and therefore requires no training.

3. *feature map w. discrete jump risk minimisation* refers to the affine transformation as described in equation (4.3.2), *but with the Δ replaced by the hedge ratios from the discrete jump risk minimisation.*

4. *continuous-time* refers to the continuous time Δ of the derivative book. As a butterfly spread consists of long- and short positions is call options, this quantity is known in (semi-)analytical form. This is the benchmark.

For all the experiments, the 95% expected shortfall is chosen as risk measure. The networks are trained for 2^{18} training samples and subsequently tested on 2^{18} test samples. These levels were chosen to ensure proper generalisation from training to test set. The mini-batch size is 2^{10} and each network is trained with early stopping and decreasing learning rates.

Proposition 2.3.5 is no longer applicable as the contingent claim is not attainable. Nevertheless, the test without transaction costs is conducted under the risk neutral dynamics as in chapter 4.5.1. The training times are once again recorded, but for the feature map we must include the time it takes to solve problem (6.1.2) and (6.1.3) which is substantial. The results are shown in the left column of figure 6.2.

The deep network fares much better than in the simple Markovian case. At one hedge instrument the risk is 10% lower than the benchmark and this figure rises steadily towards 23% as the number of hedge instruments is increased. At two additional hedge instruments, the performance has already reached the level of the discrete jump risk minimisation utilising eight hedging instruments. The dimensionality issues from chapter 4.5 does not seem to be present at least not *relative* to the now much less optimal benchmark. At eight additional hedge instruments the deep network manages to improve upon the Δ-hedge expected shortfall by 0.85 at an arbitrage-free price of 2.38. The training times are also relatively unaffected.

Despite the underlying poor performance of its feature set at low dimensions, the feature map yields risk levels which are at least 10% better than Δ-hedging. The major issue, which was not present in chapter 4.5, is that the training time scales with the number of additional hedge instruments leading to excessive training times. This is due to the very slow generation of the feature set. Approximating ξ_t in equation (6.1.3) requires, for each sample path and time point, the computation of the prices for each $J_1, \dots, J_d$ as well as the Δs. This procedure runs for over 14 minutes in the eight dimensional case, so the deep network is a much better alternative.

For proportional transaction costs of 1% where simulation is conducted under the P-measure, see the right column of figure 6.2. Unsurprisingly, the discrete jump risk minimisation is now completely useless due to the very frequent trading so it is omitted[1]. As for the deep network and feature map, the performance seems to be relatively identical. Despite transaction costs, the improvement in risk seems to rise at approximately the same rate as in the non-transaction cost case.

[1] It underperforms Δ hedging by well over -100%.

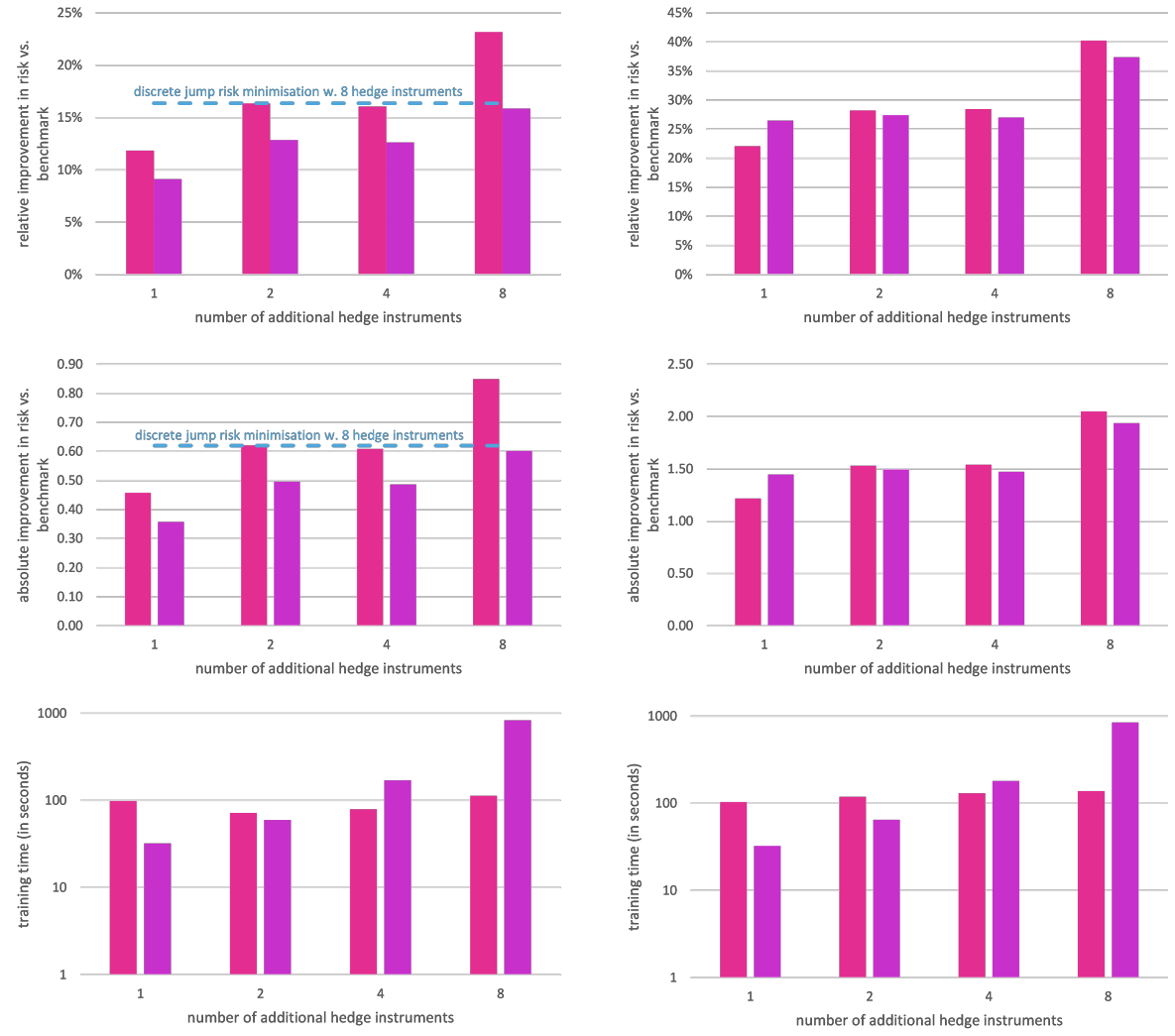


Figure 6.2: Results of the experiment described in chapter 6.3.

Chapter 7

Never Forget Path Dependence

When replicating Markovian derivatives as in chapter 4, the only relevant information was contained in the current state and previous holdings (in the case of transaction costs). Once the distribution of the underlying instruments or the derivative becomes path dependent, the relevant information is stored through the entire trajectory of the sample path. In principle one could use the entire intermediate trajectory as feature set

$$\mathbf{x}_k = \mathbf{S}_{t_0} \otimes \cdots \otimes \mathbf{S}_{t_k},$$

but this approach is of course infeasible for even small time horizons or high-dimensional instrument processes. The problem can be recast as a Markovian problem by extending the feature set with the relevant path dependent information. If hedging d barrier options the feature set could for example be extended with a d-dimensional binary vector indicating whether or not the barrier of the i'th derivative was crossed. Thus the methods developed in chapter 4.2 and 5 works perfectly well for non-Markovian derivatives.

Recasting the problem as Markovian leads to the unpleasant necessity of deciding *how* to recast it. Continuing with the barrier example, the information can in fact be encoded by the integers $\{1, \dots, 2^d\}$ in a one-dimensional vector instead which reduces the input dimension from $2d$ to $d + 1$. On the other hand, if multiple barriers are written on the same instrument, it might be more efficient to encode the running maximum. In other words, when working with non-Markovian derivatives, a lot of manual labour is required every time a new derivative is introduced. In [5] the authors propose the interesting idea of automating this labour by outsourcing the task to the neural networks themselves. This leads to the idea of having the neural networks *learn* which information is relevant to *memorise* across time.

The term *memorise* stems from the field of recurrent neural networks in which a single network must learn to store information across time. The setup from chapter 4 is only semi-recurrent as a separate network is responsible for each time step, so a more fitting term is therefore arguably that the networks must learn to *pass information along* to the next network in the chain. This chapter will discuss and test various approaches to memorising.

When the network must learn to memorise, we will distinguish between two types of memory:

1. *accumulative memory*: when the networks must maintain a running score of some cumulative value over time.

2. *event memory*: when the networks must observe specific events during the path and pass the information forward.

The first setting is typical for Asian derivatives in which running averages must be maintained, while the second setting is typical of barrier options, where knowledge of whether the barrier was crossed in the past is crucial for the optimal behaviour. It is important to distinguish between the two types of derivatives as they each introduce different challenges. The former requires that the memory is very precise as the exact values are typically important and the ladder involves the surprisingly difficult task of *doing nothing* - that is, if a network receives a signal of some event having occurred in the past, but there is no need for further update of the signal, then the network should simply pass the signal along. If the model structure consists of say thirty networks which all slightly alter a signal which should otherwise not have been altered, the signal will vanish over sufficiently long stretches of time.

7.1 Further Misdeeds of Δ Hedging

A path dependent derivative is an $\mathcal{F}_T$-measurable contingent claim Z where the payoff depends on either the entirety or a subset of the sample path of an underlying instrument process. Formally, let the path of an Itô-process be denoted as

$$\widetilde{X}_t(\omega) \triangleq \{X_s(\omega) | 0 \leqslant s \leqslant t\}.$$

For a path dependent derivative, the price process is a function of the entire trajectory of the path until time t. A way to prove this is through the functional Itô calculus, as this can be utilised to prove a Feynman-Kac theorem for continuously monitored derivatives [50]. As this theory is not as well-known as regular Itô calculus, the relevant results can be found in appendix C. The proof is omitted, but we can use the result to prove that the payoff of the derivative can be replicated by holding the path-wise space derivative defined in appendix C.2. In order to prove this, assume that the price function ϕ is smooth in a functional Itô sense,

$$\frac{H_t}{B_t} = \phi\left(t, \widetilde{\mathbf{S}}_t\right).$$

Applying the functional Itô formula from equation (C.2.1), utilising that H_t/B_t is a martingale under Q and that the numeraire has no diffusion term yields that the Q-dynamics are

$$d\left(\frac{H_t}{B_t}\right) = B_t \nabla_{\mathbf{s}} \phi\left(t, \widetilde{\mathbf{S}}_t\right) D\left[\frac{\mathbf{S}_t}{B_t}\right] \mathbf{\Sigma}_t dW_t^Q = B_t \nabla_{\mathbf{s}} \phi\left(t, \widetilde{\mathbf{S}}_t\right)^\top d\left(\frac{\mathbf{S}_t}{B_t}\right). \tag{7.1.1}$$

By comparing the expression with equation (2.1.4) it is clear that the replicating portfolio strategy $\mathbf{a}_t$ is the path-wise space derivative multiplied by the numeraire. Note that the proof assumed that the functional Itô formula can be extended to multiple dimensions exactly as with the classical Itô formula. In [50] this is not proven and the possibility of a multidimensional formula is only mentioned as a possible extension.

As the functional derivative is intuitively just the regular derivative of the price process with respect to the spot value assuming the entire past trajectory remains fixed, the above result essentially just implies that we should hedge path dependent derivatives with regular Δ hedging. This, of course, also means that all of the problems with Δ hedging described in chapter 4.1 carry over to the path dependent case.

7.1.1 Simulation of Path Dependent Derivatives

When simulating path dependent derivatives it is important to distinguish between two types: *discretely- and continuously monitored derivatives*. Both types are used for the numerical tests later in this chapter, so both types are presented now.

A simple extension of the Markovian derivatives are the discretely monitored derivatives. The derivatives are inherently path dependent, but as the name suggests, a discretely monitored derivative only depends on the trajectory of the state path at exogenously chosen discrete time points.

Simulation of the discretely monitored derivatives is very similar to the simulation of Markovian derivatives. It must be ensured that the n simulated time points at least encompass the m chosen monitoring dates as they will be used to compute the final payoff of the derivative. As long as the simulation of the state process $\mathbf{X}_t$ is exact at the m time points, then the distribution of the final payoff can be simulated *without error*.

A continuously monitored derivative is what is typically meant by a path dependent derivative. The payoff of the derivative depends on the entire trajectory of the underlying state path until maturity. Simulating continuously monitored derivatives is in general impossible as this would require simulating in continuous time. Thus any simulation of a continuously monitored derivative is fundamentally a discretely monitored approximation. This is often referred to as *discretisation bias* in the simulation and it can be quite substantial [38, ch. 3.2.2].

To showcase further issues with Δ hedging, we can consider hedging an up-and-out call option, also known as a *reverse knock-out* (see appendix A.3 for a description). The derivative is very popular as it provides massive leverage, but it is notoriously difficult to dynamically replicate. The maximum value of the Γ diverges towards ∞ as the time to maturity tends to zero. This uncontrollably high Γ results in huge discretisation bias and essentially renders dynamic Δ hedging useless around the barrier.

Two strategies are considered: one with daily hedging and one with hedging eight times a day. It is important to remember that the convergence is towards a discretely monitored approximation of the actual distribution of the derivative. The underlying instrument is simulated 32 times a day in both circumstances with the following parameters

$$T = \frac{14}{250}, \quad S_0 = 100, \quad K = 100, \quad B = 110, \quad r = 2\%, \quad \mu = 5\%, \quad \sigma = 20\%.$$

The results are illustrated in figure 7.1. There is little doubt that the losses are most significant around the barrier. Even for hedging eight times a day the risk is nowhere near eliminated.

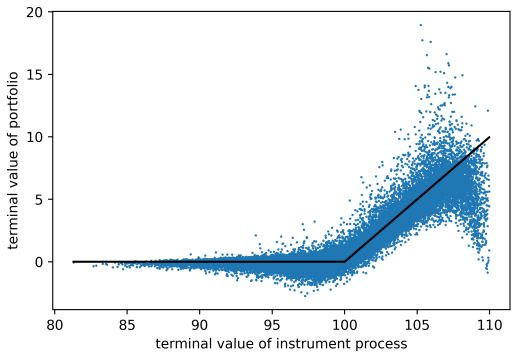
7.2 Adapting the Approximation Algorithm

This chapter develops some sufficient adaptations of the algorithm from chapter 4.2 to allow for automatic (learned) extraction of relevant information along the path of the state process (memory). Similarly as with the prior chapters, the presence of a state process $\mathbf{X}_t$ and liability Z is assumed. The difference is that the joint distribution of the state process and liability is now allowed to be path dependent.

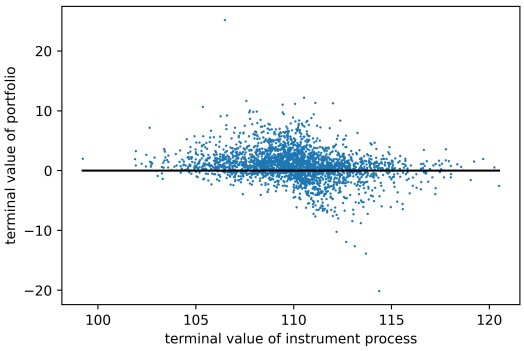
Once again the state process is appropriately preprocessed and organised into feature sets, where $\mathbf{x}_k$ represents the information available to the kth network h_k at time t_k. The issue is now that $\mathbf{x}_k$ does not contain sufficient information to properly hedge the liability. As an example, if the liability is a barrier option, the state process at time t_k does not contain information about whether or not the barrier was crossed during the prior $k-1$ time points[1].

The interesting aspect of this is that while $\mathbf{x}_k$ might not contain enough information, the past feature sets $\mathbf{x}_0, \ldots, \mathbf{x}_{k-1}$ coupled with $\mathbf{x}_k$ does. As each of the prior networks $h_0, \ldots, h_{k-1}$ are exposed to one of the past feature sets, the challenge lies in somehow having these networks pass the relevant information along to h_k. This is what is referred to as memory.

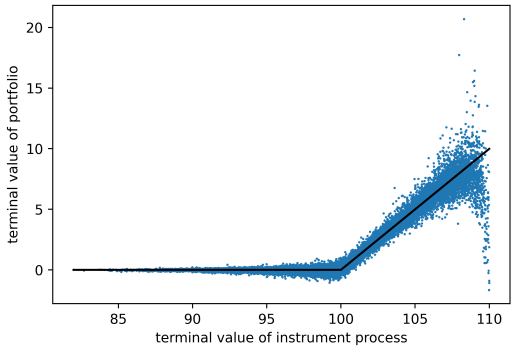
[1] This is not entirely true. If for example the underlying instrument is very close to the barrier at time t_k the predictor might be able to infer with relatively high confidence that the barrier probably was crossed in the near past.



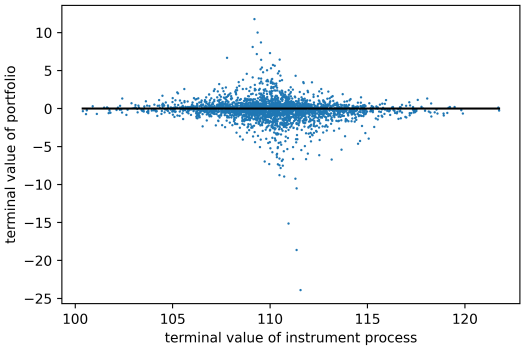
(a) barrier not crossed with $n = 14$



(b) barrier crossed with $n = 14$



(c) barrier not crossed with $n = 112$



(d) barrier crossed with $n = 112$

Figure 7.1: Relative value of terminal wealth and payoff of reverse knock-out call option for various degrees of discrete time points. The instrument was simulated 32 times daily. The x-axis represents the terminal value of the instrument S_T and the y-axis represents the terminal value of the value process $V_T/B_T + H_0/B_0$ (blue dots) and derivative payoff (black line).

7.2.1 Recurrent Neural Networks

If neural networks are used as the parametric family in equation (4.2.1) the problem is structurally similar to minimisation using recurrent neural networks. The authors in [5] and [3] even refers to the problem as *semi-recurrent*. The structure can be transformed into a bona fide recurrent network if the information set is designed properly. This can motivate ways to solve hedging problems involving path dependent derivatives.

Instead of having n different neural networks specify the portfolio strategy at each point in time, let a single network decide the strategy. Formally, set $h_k = h$ for all $k = 0, \dots, n-1$. As the portfolio strategy depends on the time, this must now be a part of the information set. If the holdings from the previous period is included in the information set, then the portfolio strategy has the following recurrent structure

$$\mathbf{a}_{t_k} = h\left(\mathbf{x}_k \otimes \mathbf{a}_{t_{k-1}}; \theta\right),$$

which is exactly the structure of a recurrent neural network [16, ch. 10.1]. On the surface, it seems as if using the inherently simpler recurrent neural networks instead of the more general formulation in equation (4.2.1) has several advantages. In general a recurrent neural network has strong approximation properties as the recursive structure allows for much higher complexity with fewer layers and nodes [14]. However, the result assumes that the intermediate values $\mathbf{a}_{t_1}, \dots, \mathbf{a}_{t_{n-1}}$ are discarded, which is not the case here. Also, one could hope that the simpler structure would allow for faster backpropagation, but this is also void as the depth of the computations are unchanged.

There is nothing in this structure which allows for memory *yet*, but the architecture has inspired the memory architecture in chapter 7.2.3.

7.2.2 Memory as a Learnable Feature

As already argued, the problem can be made Markovian by simply including the relevant path information in the feature set. One can therefore hope that the networks can learn to construct their own feature sets and pass it along. This is accomplished by letting the networks output an internal (memory) state $\mathbf{y}_k$ in addition to the actual output $\mathbf{a}_{t_k}$. This state is then appended to the input of the next network,

$$\mathbf{a}_{t_k} \otimes \mathbf{y}_k = h_k\left(\mathbf{x}_k \otimes \mathbf{a}_{t_{k-1}} \otimes \mathbf{y}_{k-1}; \theta_k\right). \tag{7.2.1}$$

At each iteration in time, the output $\mathbf{a}_{t_k}$ is used to accumulate the terminal wealth as before, while the internal state $\mathbf{y}_k$ is passed along to the next network in line. The memory vector $\mathbf{y}_0$ is initialised at some arbitrary value[2]. The higher the dimension of $\mathbf{y}_k$ the more complex information can be passed along. In [45], this structure is successfully utilised in hedging a European call option under rough volatility, but it has not been tested in conjunction with path dependent derivatives before.

The highly recurrent structure of equation (7.2.1) begs the question if modelling the entire structure through a recurrent neural network is more suitable. The issue with this approach relates to the *vanishing gradient problem*. To showcase the problem, we can restrict our attention to the memory vector and ignore $\mathbf{a}_{t_k}$. In the highly simplified setting of shallow neural networks with no bias unit, a linear activation function and no feature set, the kth network will output the memory vector,

$$\mathbf{y}_k = \mathbf{W}_k \mathbf{y}_{k-1} = \left(\prod_{j=1}^{k} \mathbf{W}_j\right) \mathbf{y}_0.$$

[2]Within the field of memory in recurrent neural networks it is often argued that the initial state of some memory representation should in fact be a learnable parameter as well. This is not utilised in the thesis, but it might be an easy way to improve performance.

If the network is recurrent, the above simplifies to $\mathbf{W}^k\mathbf{y}_0$ as the same network is utilised at each point in time. If the matrix $\mathbf{W}$ admits an eigenvalue decomposition[3] $\mathbf{P}\mathbf{\Lambda}\mathbf{P}^{-1}$, then the product can be written as

$$\mathbf{y}_k = \mathbf{P}\mathbf{\Lambda}^k\mathbf{P}^{-1}\mathbf{y}_0.$$

Thus the eigenvalue matrix will either explode or vanish depending on the magnitude of the eigenvalues. As the gradient through this graph also contains this term, the gradient will behave similarly resulting in either very slow or unstable learning [16, ch. 8.2.5, 10.7]. Thus if recurrent networks are to be used, a more advanced method of retaining memory must be incorporated. This is the topic of chapter 7.2.3.

7.2.3 Long Short-Term Memory

In [5], the authors propose utilising *Long Short-Term Memory (LSTM)* networks to accommodate path dependent derivatives. The LSTM network is an extension of the recurrent neural network structure from chapter 7.2.1 specifically designed to avoid the vanishing gradient problem [51, 52].

The entire structure surrounding a single time step in an LSTM network is often referred to as a *cell*. The cell at time t_k consists of an output gate $\mathbf{o}_k$ and memory gate $\mathbf{y}_k$ which together forms the output $\mathbf{a}_{t_k} = \mathbf{h}_k$,

$$\mathbf{h}_k = \mathbf{o}_k \odot \tanh\left(\mathbf{y}_k\right).$$

The output gate is a vector of real values between zero and one determining to which degree the memory is to be shut off in the output. This allows the network to store information in $\mathbf{y}_k$ without having it necessarily affect the current output. The output gate is modelled through a network structure

$$\mathbf{o}_k = \sigma\left(\mathbf{W}_o\mathbf{x}_k + \mathbf{U}_o\mathbf{h}_{k-1} + \mathbf{V}_o\mathbf{y}_k + \mathbf{b}_o\right),$$

where σ is the sigmoid activation function and $\mathbf{V}_o$ is a diagonal matrix. The memory vector is iteratively updated throughout time by partially forgetting the past and updating with new experience,

$$\mathbf{y}_k = \mathbf{f}_k \odot \mathbf{y}_{k-1} + \mathbf{i}_k \odot \tanh\left(\mathbf{W}\mathbf{x}_k + \mathbf{U}\mathbf{h}_{k-1} + \mathbf{b}\right).$$

Here the forget gate $\mathbf{f}_k$ and input gate $\mathbf{i}_k$ are real valued vectors between zero and one modulating to which degree the past is to be forgotten and the present is to be remembered. The gates are likewise modelled through the network structure

$$\begin{aligned}\mathbf{f}_k &= \sigma\left(\mathbf{W}_f\mathbf{x}_k + \mathbf{U}_f\mathbf{h}_{k-1} + \mathbf{V}_f\mathbf{y}_{k-1} + \mathbf{b}_f\right),\\ \mathbf{i}_k &= \sigma\left(\mathbf{W}_i\mathbf{x}_k + \mathbf{U}_i\mathbf{h}_{k-1} + \mathbf{V}_i\mathbf{y}_{k-1} + \mathbf{b}_i\right),\end{aligned}$$

where once again the matrices $\mathbf{V}_f$ and $\mathbf{V}_i$ are diagonal. By appropriatly shutting off new input through $\mathbf{i}_k$, the network can easily retain important information in the memory vector over long stretches of time.

Note importantly that all the weight matrices are *not* time dependent and the LSTM network therefore much more closely resembles a recurrent neural network. This structure is however much more complex usually involving many more parameters than the semi-recurrent counterparts. Due to the recurrent nature, the specific time step t_k must be included in the feature set $\mathbf{x}_k$, as the cell is otherwise incapable of differentiating different time steps.

More abstractly, the entire LSTM network maps a sequence of features $\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}$ into a sequence of outputs $h_0, \ldots, h_{n-1}$ in a manner that respects the constraints of not looking into the future. As the output is essentially just another sequence, a different LSTM network can be applied once more in a manner similar as to how deep feedforward networks consist of multiple *layers*.

[3] It most certainly does as the weights are initialised randomly.

The outputs are scaled between minus one and one due to the hyperbolic tangent function. To allow more expressive power for the network, the authors in [5] extends the network by multiplying each output with a (time dependent) matrix and adding a bias vector. This massively increases the amount of parameters in the model as each time step requires a separate matrix and bias vector.

The alternative *Gated Recurrent Units (GRU)* from [53] was also tested alongside the LSTM networks. Empirical evidence suggests that these units perform as well as the LSTM networks despite having a simpler architecture [54]. The fundamental difference is that no memory vector is explicitly maintained in the GRUs, but instead the kth activation is

$$\mathbf{h}_k = \mathbf{z}_k \odot \mathbf{h}_{k-1} + (1 - \mathbf{z}_k) \odot \widetilde{\mathbf{h}}_k$$

for some *update gate* $\mathbf{z}_k$ and *target gate* $\widetilde{\mathbf{h}}_k$. However, this architecture was unable to reach the level of either the architecture from chapter 7.2.2 nor the LSTM networks arguably due to the lacking memory vector. These results are therefore omitted.

7.3 Effect of Memory in Univariate Setting

Before performing numerical tests, we shall once again verify that the algorithm produces sensible solutions in univariate settings. The details of training are reserved for chapter 7.4, so for now, simply consider the problem of hedging the reverse knock-out option used as example in chapter 7.1.1.

In order to hedge the barrier, it is important for the kth network to receive information regarding whether or not the barrier was crossed in the interval $[0, t_k)$. We therefore expect the LSTM network from chapter 7.2.3 to learn to remember this information. To test this, two types of predictors were trained: a deep network without any memory and the LSTM network. In figure 7.2, the trained network's average portfolio strategy across time is illustrated on the test sample paths where the barrier was crossed on the third day.

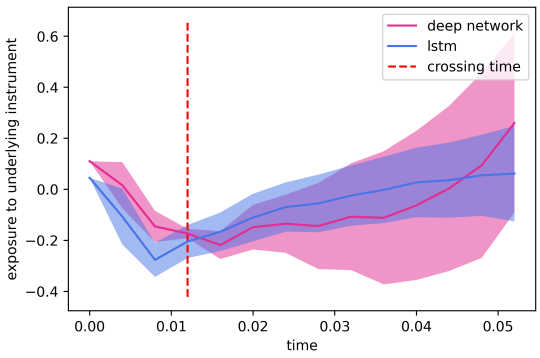


Figure 7.2: Results of the experiment described in chapter 7.3. The shaded regions indicate one standard deviation from the mean.

There is a clear change in behaviour for the LSTM network when the barrier is crossed. The average holdings in the underlying instrument are sharply decreasing until day three after which the holdings slowly begin to rise (on average) towards zero. Most importantly, *this behaviour is maintained long after the barrier is crossed* which indicates that the information is memorised across time. The deep network, on the other hand, only behaves similarly to the LSTM network *around the time when the barrier is crossed*, but clearly does not maintain the behaviour afterwards as the subsequent networks have no way of knowing that the barrier was crossed.

7.4 Testing the Algorithm in Multivariate Environments

This chapter contains numerical experiments testing whether it is possible for the predictors described in chapter 7.2 to learn to extract sufficient information along the sample paths to hedge a path dependent

liability. As usual, it is of the utmost importance to conduct the tests in multivariate environments to study the algorithms and predictors robustness to high dimensions.

As in [5], we are particularly interested in the prospect of risk optimal hedging of barrier options due to the shortcomings of Δ hedging. However, from a purely investigative point of view, it is very difficult to assess the performance of the algorithm for the very same reason, since outperforming Δ hedging is not a particularly challenging feat. A much more challenging problem is to outperform Δ hedging in the context of Asian derivatives as the Γs are much smaller. Such an additional test is therefore included beforehand. The two tests will therefore also respectively involve event- and accumulate memory.

The deep network from the previous chapters is still part of the test and new ones with incorporated memory are included. To summarise, the predictors are as follows:

1. *deep network* refers to the same type of network as in the experiments from chapter 4.5.
2. *feature map w. true* Δ refers to the Δ feature map as described in equation (4.3.2).
3. *deep memory network* refers to a 15 unit feedforward neural network with four layers and learnable memory features as described in chapter 7.2.2.
4. *LSTM* refers to an LSTM network with two LSTM cells with 15 units in addition to a linear output layer.
5. *continuous-time* refers to the continuous time Δ of the derivative book. This is the benchmark.

Note that the feature map with true Δ has no concept of memory and therefore does not adhere to the ambitions of this chapter. The inclusion mostly serves as a second benchmark to gauge how close the memory networks are to risk optimal behaviour.

The predictors are trained in accordance with chapter 4.2 with the adaptations described in chapter 7.2. The training set consists of 2^{18} sample paths and everything is tested on 2^{18} new samples after training. In the setting without transaction costs, the 95% expected shortfall is chosen and in the setting with transactions costs the mean-variance measure with a risk aversion of one is chosen.

7.4.1 Asian Derivatives

To ensure an analytical solution for the Δ is available, the Asian derivative is chosen as a European call option written on the (discretely monitored) geometric average of the underlying instrument process. As this derivative is not part of the common derivative repertoire, a detailed description is available in appendix A.2. In short, for the m monitoring dates, the payoff of the derivative is

$$Z = \left(\prod_{j=1}^{m} S_{t_j}^{t_j - t_{j-1}} - K \right)^+ .$$

The experiments will proceed similarly as in chapter 4.5 with testing being conducted for increasing dimensions of the instrument process. For each instrument, the book contains a call option written on the discretely monitored geometric average of the instrument process with a strike equal to S_0^T. Once again the exposure to each derivative is normalised by d^{-1} to keep the price identical across dimensions. The underlying instrument process is chosen exactly as with the Markovian experiments except that the volatility is normalised to 40% and the derivatives are hedged monthly for one year. The increase is simply to offset the inherent low volatility of the derivatives.

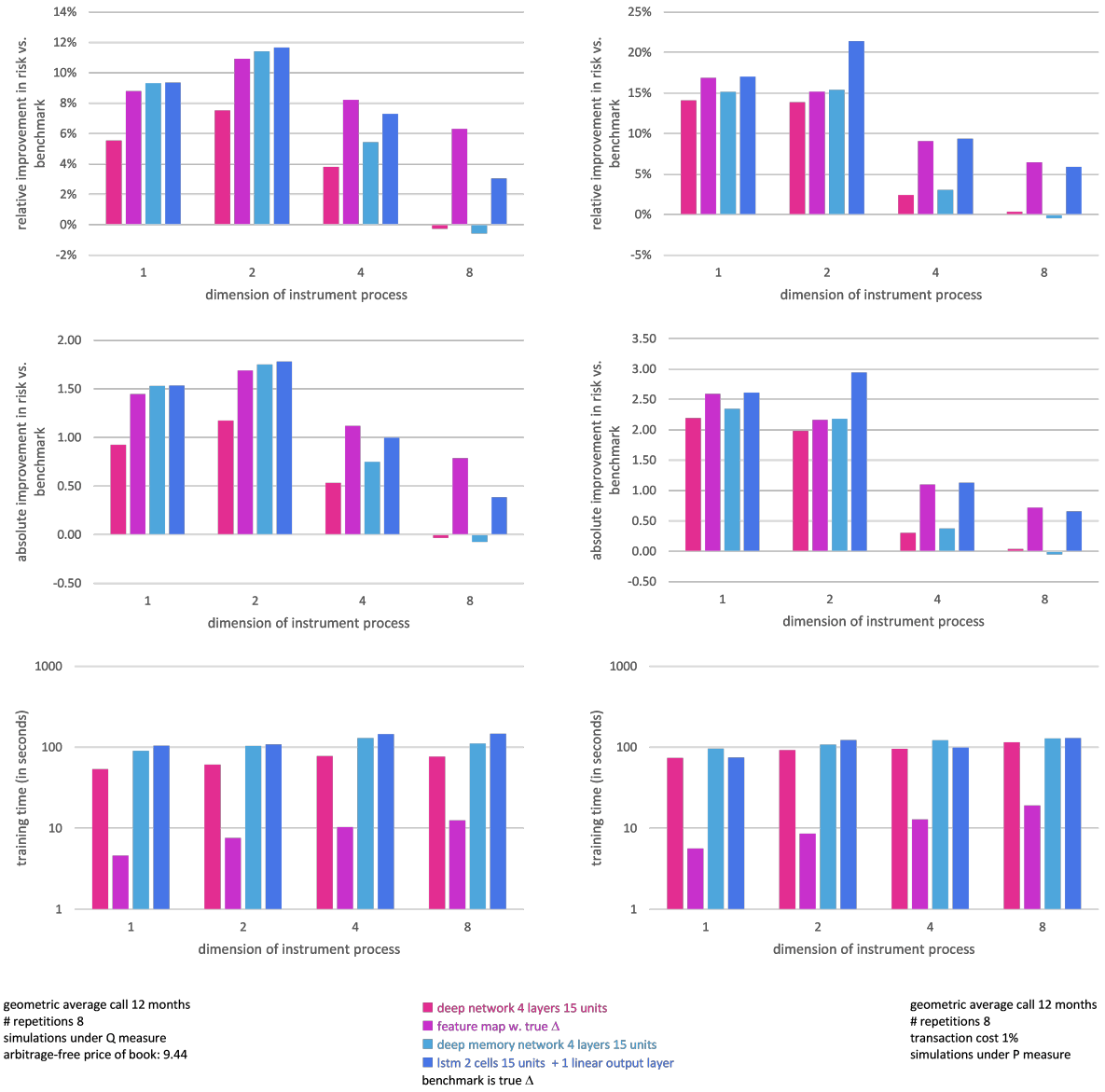


Figure 7.3: Results of the experiment described in chapter 7.4.1

Proposition 2.3.5 is still valid for path dependent derivatives, so we can once again use Δ hedging as a strong benchmark if we simulate under the Q measure without transaction costs. The results of such an experiment is depicted in the left column of figure 7.3.

The deep network without memory fares surprisingly well despite only being exposed to the current value of the instrument process at each point in time. It consistently outperforms Δ hedging by around 6% for low dimensions and reaches a similar level of risk in eight dimensions.

The deep memory network remarkably reaches the level of the feature map with true Δ for low dimensions. Despite its very simple architecture, the network is capable of memorising relevant information along the sample path. Unfortunately, the performance deteriorates for higher dimensions.

The LSTM network is the best performing predictor for low dimensions and at eight dimensions, it still somehow manages to *learn* to remember enough information about the past to *improve* upon the benchmark by a few percentage points. The training time is equivalent to the deep memory network. Unfortunately, the feature map with true Δ is significantly better at high dimensions, which shows that even the LSTM network does not reach the optimal solution.

Overall the results indicate that it is possible to construct a fully automated algorithm for computing risk optimal portfolio strategies without having to manually encode the relevant path information. While the performance generally does deteriorate across dimensions it does not seem as prominent as in the prior chapters.

A rather surprising aspect of figure 7.3 is the fact that the deep network without memory still manages to outperform the benchmark. Increasing the amount of monitoring dates should increase the level of path dependence of the derivative and as a result, the non-memory networks should fail to achieve any reasonable solution. A single test of a univariate derivative book hedged 60 times throughout the year without repetitions was performed and the results were quite clear: the non-memory deep network underachieved by -6%, the deep memory network by -1% and the LSTM still improved upon the benchmark by 2%.

The results of a similar experiment with proportional transaction costs of 1% and simulations under the P-measure can be found in the right column of figure 7.3. There is little difference between the non-memory and memory networks for low dimensions, but as the dimension grows, the difference increases. The deep memory network showcase slightly better performance than the deep network, but arguably not enough to justify the extra complication in the architecture. The LSTM network performs as well or better than the feature map (and all the others) at all dimensions and is clearly the favourable choice.

The experiments show that the architecture of the networks has an important effect on the performance especially in higher dimensional settings. It is clear that the memory networks do manage to memorise important information, but the performance of the non-memory network also shows, that apparently path dependent options can be hedged quite effectively without memory under transaction costs.

7.4.2 Barriers

To ensure as challenging a problem as possible, the barrier option is chosen as the (continuously monitored) reverse knock-out option (see appendix A.3 for a detailed description). The up-and-out call option also known as the reverse knock-out has the payoff,

$$Z \triangleq \begin{cases} (S_T - K)^+ & \sup_{0 \leqslant t \leqslant T} S_t \leqslant B \\ 0 & \text{otherwise} \end{cases}, \tag{7.4.1}$$

where it is implicitly assumed that $K < B$ as the derivative is otherwise worthless. As it is not possible to simulate the derivative exactly, the test essentially involves a discretely monitored approximation. The analytical Δ is available for this option, but as described in chapter 7.1.1 the Γ is so excessive that Δ hedging is useless.

The experiments proceed similarly as in the previous chapter with testing being conducted for increasing dimensions of the instrument process. For each of the d instruments, the book contains a reverse knock-out option with strike S_0 and a knock-out barrier of $S_0 + 10^4$. Once again the spot values are normalised to one hundred and the marginal volatility to 20%. The derivatives expires in two weeks and hedging is conducted daily.

To ensure a sufficient approximation of the derivative, the instrument process is simulated sixteen times daily[5]. Remember importantly that the memory networks only have access to one in sixteen of the simulated time points.

Proposition 2.3.5 is still valid for path dependent derivatives, so we can once again use Δ hedging as a strong benchmark if we simulate under the Q measure without transaction costs. The results of such an experiment is illustrated in the left column of figure 7.4.

The feature map with true Δ is now consistently the worst of the predictors. The obvious explanation is that the strategy strongly relies on the continuous time Δ providing a solid foundation to extend upon. As Δ hedging is riddled with risk for reverse knock-out options, so too is the linearised Δ strategy.

All three networks perform relatively identically for low dimensions. In the univariate setting, they each improve upon the benchmark by at least 40%. Looking at the absolute improvement, the networks improve upon the risk by 1.40 despite the arbitrage-free price only being 1.50. This figure slowly decreases until eight dimensions where the improvement is 5% for the non-recurrent networks. For the LSTM network the performance for eight dimensions is once again much more robust for high dimensions achieving an improvement of 10%.

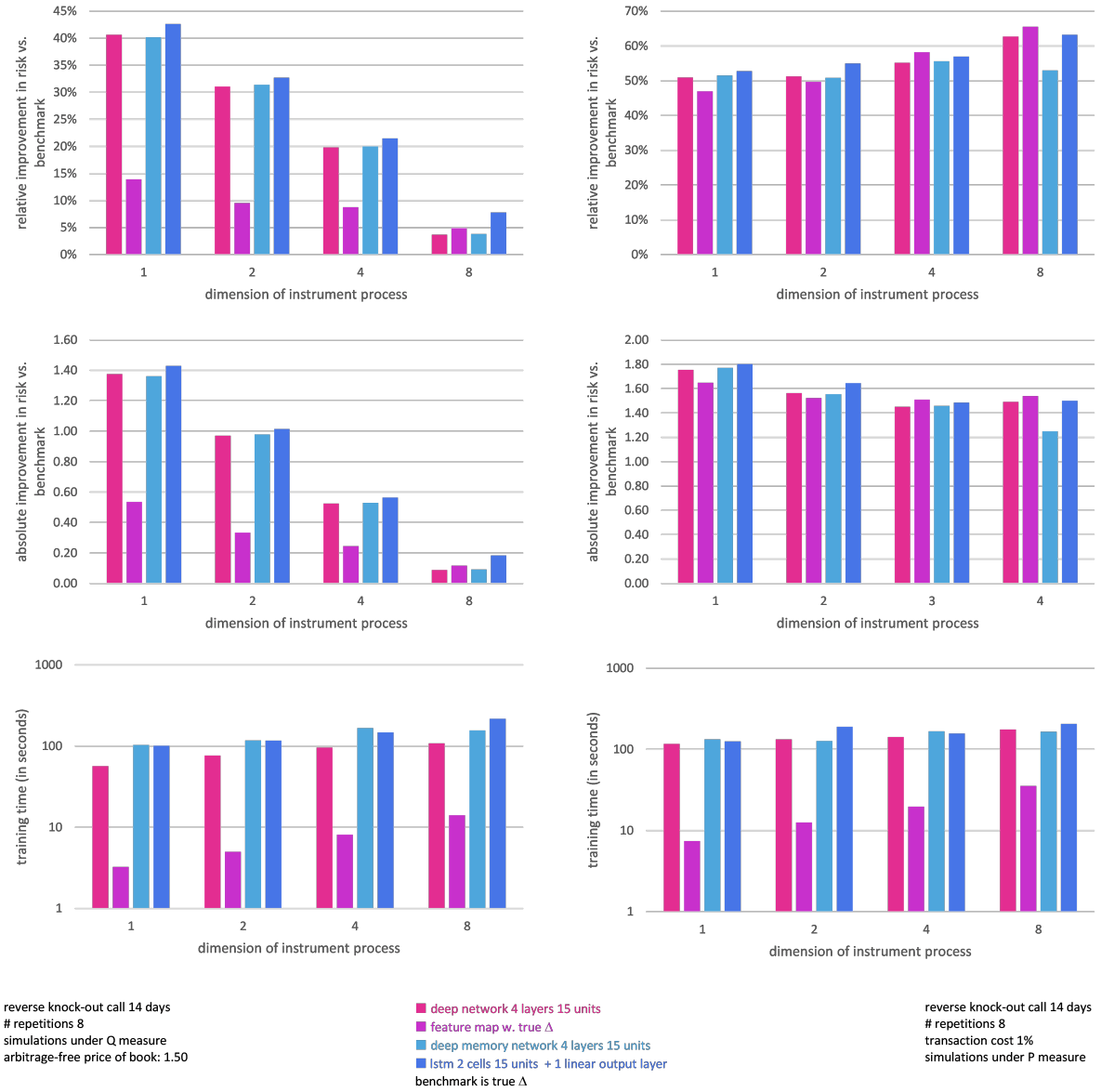
There is essentially no difference between the deep network and deep memory network at any dimension, and for the low dimensions, there effectively seems to be no benefit in modelling memory at all. One might just as well utilise predictors designed for Markovian problems.

The major issue with barrier options, and essentially the reason for conducting the test with Asian derivatives in the previous chapter, is that we have no way of knowing why memory has little effect. As we have no solid benchmark to test against, it could just be that *all* three networks achieves risk optimal behaviour. Alternatively it *could* be that the networks are incapable of learning event memory and therefore performs similarly to the deep network despite it being far from optimal. As we saw in the previous chapter that memory networks can learn to memorise important path information, the most likely explanation is arguably that it is possible to hedge barrier options very risk effectively without necessarily needing information regarding the past.

The results of a similar experiment with proportional transaction costs of 1% and simulations under the P-measure can be found in the right column of figure 7.3. There is essentially no difference between the four predictors as they all massively improve upon the benchmark by at least 50%. At eight dimensions the improvement has grown to 60%. This is no surprise as Δ hedging trades extremely aggressively around the barrier which is very ineffective with transaction costs.

[4] At this level, the barrier is crossed for 5% of the sample paths under P.

[5] This level ensures that the discounted average payoff of the derivative when simulated under the risk neutral measure achieves the correct price within the first two decimals.



reverse knock-out call 14 days
repetitions 8
simulations under Q measure
arbitrage-free price of book: 1.50

deep network 4 layers 15 units
feature map w. true Δ
deep memory network 4 layers 15 units
lstm 2 cells 15 units + 1 linear output layer
benchmark is true Δ

reverse knock-out call 14 days
repetitions 8
transaction cost 1%
simulations under P measure

Figure 7.4: Results of the experiment described in chapter 7.4.2.

Chapter 8

Conclusion

The thesis has investigated various uses of statistical learning theory to approximate risk optimal portfolio strategies for hedging liabilities in markets with frictions such as time discretisation and transaction costs. In particular, there has been a keen focus on constructing tests of performance in multivariate settings.

The central problem of the thesis, the liability owner's problem, was formulated in chapter 2. It was shown that the problem can be solved irrespective of the initial cash-injection, which allowed us to formulate prices of derivatives in the absence of the usual idealised and frictionless market assumptions. It was also proven that if the market is sufficiently efficient, then Δ-hedging solves the problem. This allowed us to construct a tight lower bound on the optimal solutions in the later numerical tests.

Selected topics from statistical learning theory were presented in chapter 3. In particular, we showed that a certain type of minimisation problem, dubbed the statistical learning problem, can be approximately solved by parameterising the solution and utilising gradient descent. Two types of parametric families were considered: neural networks and linear feature maps.

In chapter 4, we followed [5] and recognised the liability owner's problem as a special case of the statistical learning problem. As a consequence, we were able to approximate optimal portfolio strategies by gradient descent optimisation. The optimal strategies were approximated by neural network as in [5] as well as with the novel linear predictors. The two predictors were thoroughly tested and it was found that the neural networks were adept at approximating univariate solutions, but underperformed relative to Δ hedging in multivariate environments if the training times were kept short. The linearised Δ strategy performed as well as the neural networks and did not suffer from the same dimension issue.

The linearised Δ strategy was investigated further in chapter 5. At first, following [37] and [44], we recognised that the approximation of prices and sensitivities of general derivative books can also be considered a statistical learning problem. We considered the contribution of [6] who recognised that combining both methods leads to better approximation of the Δ. We extended the findings in the paper by testing the evolution of the approximation capability across dimensions and found that the twin network was the most stable. Inspired by [42], we also tested the performance of the approximation when used as a substitute for Δ hedging and found that the performance was relatively stable across dimensions here as well. This allowed us to formulate a new algorithm to approximate risk optimal portfolio strategies which was stable across dimensions, faster to train, and interpretable as an affine transformation of the Δ.

In chapter 6, we removed the assumption of completeness in the market by introducing jump risk. In particular, we investigated how well the neural networks can learn to utilise other options as hedge instruments. It was discovered that the networks are far superior to naive jump risk hedging. There was no indication that the networks suffered from the same dimension issues in this setting, but the conclusion stands on a weak foundation, as proposition 2.3.5 is not valid in incomplete markets.

Finally, in chapter 7, the possibility of constructing a general algorithm for approximating risk optimal portfolio strategies for path dependent derivatives was investigated. This approach requires that the networks can learn to memorise relevant information from the trajectory of the state process. For this purpose we followed [5] who proposed using LSTM networks and [45] who proposed a semi-recurrent architecture. While we did find evidence that the networks do not reach the optimal solution, they still vastly outperformed regular Δ hedging, especially when transaction costs are present.

Overall, it is clear from the experiments that it is possible to leverage the theory of statistical learning to solve hitherto unsolvable problems within the field of computational finance. The performances of the algorithms are highly dependent upon the specific problem and the best results are achieved when also leveraging results from traditional arbitrage theory.

Appendix A

Selected Results from the Black and Scholes Model

This appendix contains the relevant information on the Black and Scholes model used throughout the thesis. In the notation from chapter 2.1 the Black and Scholes model is characterised by α_t, $\mathbf{\Sigma}_t$ and r_t being constant. The dynamics under the P-measure can then for $i \in \{1, \ldots, d\}$ be expressed as

$$dS_t^{(i)} = \alpha^{(i)} S_t^{(i)} dt + S_t^{(i)} \sum_{j=1}^{k} \mathbf{\Sigma}_{ij} dW_j^P(t).$$

Under the Q measure the $\alpha^{(i)}$ is replaced by the interest rate. The interpretation is much easier if it is modelled through a correlated Brownian motion. Let the ith row of $\mathbf{\Sigma}$ be denoted by $\mathbf{\Sigma}_i$ such that the above can be expressed as [7, ch. 4],

$$dS_t^{(i)} = \alpha^{(i)} S_t^{(i)} dt + \|\mathbf{\Sigma}_i\| S_t^{(i)} d\widehat{W}_i^P(t), \qquad d\widehat{W}_i^P(t) \triangleq \frac{\mathbf{\Sigma}_i}{\|\mathbf{\Sigma}_i\|} dW^P(t).$$

When hedging derivatives with a single underlying instrument it is therefore possible to draw from the one-dimensional theory. The solution to the stochastic differential equation can be solved explicitly [7, ch. 5.2] for $s < t$ as

$$S_t^{(i)} = S_s^{(i)} \exp\left[\left(r - \frac{1}{2}\|\mathbf{\Sigma}_i\|^2\right)(t-s) + \|\mathbf{\Sigma}_i\|\left(\widehat{W}_i^Q(t) - \widehat{W}_i^Q(s)\right)\right]. \tag{A.0.1}$$

When referring to the univariate setting one often writes $\sigma \triangleq \|\mathbf{\Sigma}_i\|$. The following appendix assumes a univariate setting.

A.1 The European Call- and Put Option

The European call- and put option is arguably the most well-known Markovian derivative. For a strike price K the owner of the call option is endowed with the option to buy the underlying instrument at time T for the price of the strike. Similarly for the put option, where the owner has the option to sell the instrument for the price of the strike. If we let $\theta \in \{-1, 1\}$ indicate whether the derivative is a call or put, then the payoff of the derivative at time T can be expressed as

$$Z = \max\left\{\theta\left[S_T - K\right], 0\right\}.$$

The price process is famously known in analytical form [7, ch. 7]. As the Δ is the relevant quantity, the exact form of the price process is omitted. The Δ in the notation of chapter 2.1 is

$$\Delta\left(t, X_t\right) = \frac{\theta}{B_t}\Phi\left(\theta\left[\frac{\ln\left(S_t/K\right)}{\sigma\sqrt{T-t}} + \frac{\sigma\sqrt{T-t}}{2}\right]\right). \tag{A.1.1}$$

Here Φ denotes the cumulative distribution function of the standard normal distribution. The Γ can likewise be expressed in analytical form as

$$\Gamma(t, S_t) = \frac{1}{S_t B_t \sigma\sqrt{T-t}}\varphi\left(\frac{\ln\left(S_t/K\right)}{\sigma\sqrt{T-t}} + \frac{\sigma\sqrt{T-t}}{2}\right),$$

where φ refers to the density of the standard normal distribution[1]. As discussed at the end of chapter 4.1 the approximation error in the presence of time frictions is highest when the Γ is at its peak. This happens exactly in the point

$$S_t = K\exp\left[-\frac{3}{2}\sigma^2\sqrt{T-t}\right] \leqslant K.$$

Thus the time discretisation is highest when the instrument is slightly below the strike and the gap tightens as time moves closer to maturity.

A.2 Derivatives Written on the Geometric Average

Derivatives written on the geometric average of the underlying instrument is a special case of the Asian options. They are seldom used in practice [38, ch. 3.2.2], but they allow for analytical formulas in the Black and Scholes model, so they are perfect in numerical tests. While the geometric options is only used in discrete form in this thesis, the most pedagogical way to introduce them is as an approximation to the continuously monitored case.

Consider a bank account with the continuously compounded interest rate as the log of the instrument value. Such a bank account has the dynamics

$$dG_t = G_t \ln S_t dt.$$

The process G_t can in other words be interpreted as the continuously compounded geometric average of the underlying process. A geometric average option is then some option written with G_t as the underlying instrument. Note that such an option is path-dependent. In the Black and Scholes model the distribution of G_t can be found in analytical form. It is a well-known fact from interest rate theory that conditional on time $s < t$ the process G_t can be written as

$$\begin{aligned} G_t &= G_s \exp\left[\int_s^t \ln S_u du\right] \\ &= G_s S_s^{t-s} \exp\left[\left(r - \frac{1}{2}\sigma^2\right)\int_s^t (u-s)\, du + \sigma\int_s^t W^Q(u) - W^Q(t) du\right], \end{aligned}$$

where the last equality only holds in the Black and Scholes model as equation (A.0.1) is used. The distribution of the integral of the Brownian motion can be derived using the product rule from Itô's formula[2] and the results from [7, ch. 4.6] as

$$\int_s^t W^Q(u) - W^Q(s) du = \int_s^t (t-u)\, dW^Q(u) \sim \mathcal{N}\left(0, \frac{(t-s)^3}{3}\right).$$

[1] Not to be confused with ϕ: the pricing function from equation (4.1.1)

[2] Use $d\left(uW_u\right) = udW_u + W_u du$, integrate on both sides and subtract $W_t(T-t)$.

The time integral inside the exponent is of course found by integration. In other words, conditional on time s, the entire expression inside the exponential term at time t is distributed according to the log-normal distribution with parameters

$$\mu(s,t) \triangleq \left(r - \frac{1}{2}\sigma^2\right)\frac{(t-s)^2}{2}, \qquad \nu^2(s,t) \triangleq \sigma^2\frac{(t-s)^3}{3}.$$

Thus when pricing and hedging Markovian derivatives written on G_t one can simply price the derivative as if the derivative was written on the underlying instrument, but with an altered spot, interest rate and volatility. One has to be careful when calculating the Δ though as this quantity is still the derivative with respect to the underlying instrument, not the geometric average.

As the price process of the option requires knowledge of the running (continuously monitored) geometric average it is impossible to compute in practice. An alternative discrete approximation is the discretely monitored counterpart. Denote the m monitoring dates as

$$0 < t_1 < \cdots < t_m = T$$

and set $t_0 = 0$. We can then at time t_k define the (discretely monitored) geometric average as the approximated version of the (continuously monitored) geometric average

$$\forall k \in \{0,\ldots,m\}: \quad \widehat{G}_{t_k} \triangleq \prod_{j=1}^{k} S_{t_j}^{(t_j - t_{j-1})} \approx \exp\left[\sum_{j=1}^{k}\int_{t_{j-1}}^{t_j} \ln S_u du\right] = G_{t_k},$$

with the convention that the product is equal to one for $k = 0$. We can now derive the distribution of the discretely monitored geometric average. For this purpose, let $\Delta t_j \triangleq t_j - t_{j-1}$ for $j \geqslant 1$. The first step is recognising that conditional on time t_k for $k \in \{0,\ldots,m-1\}$ the terminal value G_T can be written as

$$\widehat{G}_T = \widehat{G}_{t_k} S_{t_k}^{T-t_k} \exp\left[\left(r - \frac{1}{2}\sigma^2\right)\sum_{j=k+1}^{m}(t_j - t_k)\,\Delta t_j + \sigma\sum_{j=k+1}^{m}\left(W_{t_j} - W_{t_k}\right)\Delta t_j\right].$$

Exactly as with the continuous case, we can rewrite the convoluted sum involving the Brownian motion as

$$\sum_{j=k+1}^{m}\left(W_{t_j} - W_t\right)\Delta t_j = \sum_{j=k+1}^{m}(T - t_{j-1})\,\Delta W_{t_j} \sim \mathcal{N}\left(0, \sum_{j=k+1}^{m}(T - t_{j-1})^2\,\Delta t_j\right).$$

We can therefore proceed exactly as in the continuous case, but with the altered mean and variance as

$$\widehat{\mu}(k) \triangleq \left(r - \frac{1}{2}\sigma^2\right)\sum_{j=k+1}^{m}(t_j - t_k)\,\Delta t_j, \quad \widehat{\nu}^2(k) \triangleq \sigma^2\sum_{j=k+1}^{m}(T - t_{j-1})^2\,\Delta t_j.$$

These expressions are simply the discrete approximations of the integrals computed in the continuous case. Thus at the specific monitoring dates, one can compute derivative prices in the same manner as with the continuous case. Computing the value of the option for some $t \in [t_k, t_{k+1})$ is more difficult, but not necessary for the purposes of the thesis.

A.3 Barrier Options

It is possible to derive analytical formulas for barrier options in the Black and Scholes model. Deriving the price process for all four types of barrier options is long and convoluted so it is omitted. This

appendix will mostly serve as a general overview of how the derivatives are priced. The up-and-out call option also known as the reverse knockout has the payoff,

$$Z \triangleq \begin{cases} (S_T - K)^+ & \max\limits_{0 \leqslant t \leqslant T} S_t \leqslant B \\ 0 & \text{else} \end{cases},$$

where it is implicitly assumed that $K < B$ as the derivative is otherwise worthless. An elegant derivation of the price process can be found in [55]. In short, consider a claim with payoff g and its *reflected* counterpart $\widehat{g}$,

$$g(x) \triangleq \nu(x) 1_{(x \leqslant B)}, \qquad \widehat{g}(x) \triangleq \left(\frac{x}{L}\right)^p g\left(\frac{L^2}{x}\right),$$

where $p \triangleq 1 - 2r/\sigma^2$. Let further $h \triangleq g - \widehat{g}$ denote the *adjusted payoff*. The barrier option can then be replicated utilising the follows strategy: hedge the claim h and if the instrument crosses the barrier, sell all holdings in the instrument.

The *reflection* theorem allows one to compute the price process of the reflected claim if the price process of g is known. In other words, the problem is reduced to pricing a T-claim which is much simpler. The price processes of the down-and-out contract and the in-contracts can be derived similarly or through various parity relations. See [7, ch. 18] for details.

Appendix B

Selected Results from the Merton Jump Diffusion Model

As mentioned in chapter 6.1 the Merton jump diffusion model extends the Black and Scholes model with a jump component,

$$dS_t = (r - \lambda\kappa) S_t dt + \sigma S_t dW_t^Q + (J_t - 1) S_t dN_t.$$

Assuming $\ln J_t$ is normally distributed with mean α and variance δ^2 it follows that the jump compensation κ must be

$$\kappa = E^Q [J_t - 1] = \exp\left[\alpha + \frac{1}{2}\delta^2\right] - 1.$$

If not, S_t/B_t would not be a martingale under Q.

B.1 Derivation of the Portfolio Dynamics under Jump Risk

In this appendix we shall derive the dynamics in equation (6.1.1). For this purpose the Itô-formula for jump diffusion processes must be utilised. For some well defined stochastic process X_t which is a solution to the stochastic differential equation

$$dX_t = \mu_t dt + \sigma_t dW_t + \eta_t dN_t$$

the transformed process $f(t, X_t)$ will be a solution to [56, ch. D.3]

$$df(t, X_t) = \left[\nabla_t f + \mu_t \nabla_x f + \frac{1}{2}\sigma_t^2 \nabla_{xx} f\right] dt + \sigma_t \nabla_x f dW(t) + [f(t, X_{t-} + \eta_t) - f(t, X_{t-})]\, dN(t),$$

where X_{t-} is the limit from the left in time. Consider a portfolio consisting of a_t units in the underlying instrument S_t, $\xi_t \in \mathbb{R}^d$ units in the other hedge instruments $\mathbf{I}_t$, and short one unit of some (Markovian) liability Z with price process H_t. Denote the value process of the portfolio as V_t. The holdings of the back account are assumed such that the portfolio is self-financing. From equation (2.1.3) it follows that the dynamics are

$$d\left(\frac{V_t}{B_t}\right) = a_t d\left(\frac{S_t}{B_t}\right) + \xi_t d\left(\frac{\mathbf{I}_t}{B_t}\right) - d\left(\frac{H_t}{B_t}\right).$$

A direct application of the Itô-formula for jump processes yields the dynamics of the hedge instruments,

$$d\left(\frac{\mathbf{I}_t}{B_t}\right) = \left[\nabla_t\left(\frac{\mathbf{I}_t}{B_t}\right) + \left(\alpha^P - \lambda^P\kappa^P\right) S_t\nabla_s\left(\frac{\mathbf{I}_t}{B_t}\right) + \frac{\sigma^2 S_t^2}{2}\nabla_{ss}\left(\frac{\mathbf{I}_t}{B_t}\right) - r\frac{\mathbf{I}_t}{B_t}\right] dt$$
$$+\sigma S_t\nabla_s\left(\frac{\mathbf{I}_t}{B_t}\right) dW_t^P + \Delta_J\left(\frac{\mathbf{I}_t}{B_t}\right) dN_t^P,$$

where $\Delta_J \mathbf{I}_t/B_t$ refers to the difference in the value of $\mathbf{I}_t/B_t$ before- and after a jump of size J_t. The same method applied to H_t/B_t yields identical dynamics with $\mathbf{I}_t$ replaced by H_t. Inserting the terms into the dynamics of V_t/B_t and ignoring the dt terms yields

$$d\left(\frac{V_t}{B_t}\right) = [\cdots]\, dt + \left[a_t + \xi_t^\top\nabla_s\left(\frac{\mathbf{I}_t}{B_t}\right) - \nabla_s\left(\frac{H_t}{B_t}\right)\right]\sigma S_t dW_t^P$$
$$+\left[a_t\Delta_J S_t + \xi_t^\top\Delta_J\left(\frac{\mathbf{I}_t}{B_t}\right) - \Delta_J\left(\frac{H_t}{B_t}\right)\right] dN_t^P.$$

B.2 Valuation of Markovian Derivatives

It is possible to express the value of Markovian derivatives as a function of values in the regular Black and Scholes model. A natural consequence of this is that one can derive analytical formulas for call- and put options. The general formula is a direct consequence of the law of total expectations,

$$\begin{aligned}\phi(t,s) &= E^Q\left[\frac{Z}{B_T}\middle| S_t = s\right] \\ &= \sum_{n=0}^{\infty} P\left(N_T - N_t = n\right) E^Q\left[\frac{Z}{B_t}\middle| S_t = s, N_T - N_t = n\right] \\ &= \sum_{n=0}^{\infty} \frac{(\lambda(T-t))^n}{n!} e^{-\lambda(T-t)} E^Q\left[\frac{Z}{B_t}\middle| S_t = s, N_T - N_t = n\right].\end{aligned}$$

It follows from the log-normal assumption on the jump process that conditional on $N_T - N_t = n$ the distribution of S_T remains log-normal as products of log-normal distributions are themselves log-normal. To see this, simply note

$$S_T = S_t \exp\left[\left(r - \lambda\kappa - \frac{1}{2}\sigma^2\right)(T-t) + \sigma\left(W_T - W_t\right) + \sum_{j=1}^{N_T - N_t} \ln J_{t_j}.\right],$$

where t_j refers to the times when the Poisson process N_t jumped in the interval from t to T. As $\ln J_t$ is an independent sequence of normally distributed random variables it follows that conditional on $N_T - N_t = n$ the expression inside the exponential function follows a normal distribution with mean and variance,

$$\mu_n \triangleq \left(r - \lambda\kappa - \frac{1}{2}\sigma^2\right)(T-t) + n\alpha, \quad \nu_n^2 \triangleq \sigma^2(T-t) + n\delta^2.$$

One can then match terms with equation (A.0.1) and subsequently compute the expectations inside the infinite sum as if the claim was priced in the Black and Scholes model with the altered parameters. The convergence is very rapid as the density function of the Poisson distribution converges towards zero fast.

Appendix C

Functional Itô Calculus

C.1 The Classical Itô-integral

This appendix contains a brief review of the classical theory of Itô calculus. The reader is assumed to be familiar with the theory, so the appendix mostly serves as an introduction of the notation as well as a reference point to understand the novelty of the theory in appendix C. All proofs and further reference can be found in [8, ch. 3.1].

Let $(\Omega, \mathcal{F}, P)$ be a probability space with an associated filtration $\mathbf{F} \triangleq (\mathcal{F}_t)_{t\in[0,T]}$, where the filtration is assumed solely generated by a Brownian motion W_t. The object of study is the definition of an integral with respect to a function $f : [0,\infty) \times \Omega \longrightarrow \mathbb{R}$. We say that such a function is a member of the set $\mathcal{V}([a,b])$ if the function is $\mathbb{B} \times \mathcal{F}$-measurable, adapted to the filtration $\mathbf{F}$ and satisfies the condition

$$E\left[\int_a^b f(t,\omega)^2 dt\right] < \infty.$$

The workhorse object of Itô integrals is the class of *elementary functions*. A function $\phi \in \mathcal{V}([a,b])$ is elementary if it can be written on the form

$$\phi(t,\omega) = \sum_{j=0}^{n-1} e_j(\omega)\, 1_{[t_j,t_{j+1})}(t)$$

for some discrete division $a = t_0 < \cdots < t_n = b$ and $\mathcal{F}_{t_j}$-measurable functions e_j. As the elementary functions are constant on each partition, the *Itô-integral* of an elementary function ϕ is then naturally defined as

$$\int_a^b \phi(t,\omega) dW_t(\omega) \triangleq \sum_{j=0}^{n-1} e_j(\omega)\left(W_{t_{j+1}}(\omega) - W_{t_j}(\omega)\right).$$

The fundamental insight in Itô-calculus is that for any function $f \in \mathcal{V}([a,b])$ there exists a sequence of elementary functions $\phi_n \in \mathcal{V}([a,b])$ such that

$$E\left[\int_a^b |f(t,\omega) - \phi_n(t,\omega)|^2 dt\right] \longrightarrow 0.$$

In other words the function f can be approximated arbitrarily well by a elementary functions in the above sense. As the Itô-integral is already defined for elementary functions, the natural extension to f

is therefore

$$\int_a^b f(t,\omega)dW_t(\omega) \triangleq \lim_{n\to\infty} \int_a^b \phi_n(t,\omega)dW_t(\omega).$$

This limit always exists and is in fact a member of the $L^2(P)$-space as the Itô-integrals of the sequence of elementary functions constitute a Cauchy sequence. If we restrict the class of functions such that they are members of $\mathcal{V}([0,t])$ for all $t \leqslant T$, we can define the all-important *Itô-process* as a stochastic process defined on $(\Omega, \mathcal{F}, P)$ which can be written on the form

$$X_t(\omega) \triangleq X_0 + \int_0^t \alpha(t,\omega)\,dt + \int_0^t \Sigma(t,\omega)\,dW_t(\omega). \tag{C.1.1}$$

Such a process can be chosen to be continuous in time and is a martingale with respect to $\mathbf{F}$ if and only if $\alpha = 0$. The definition can be further generalised to multiple dimensions and the definition of $\mathcal{V}$ can be relaxed [8, ch. 3.3].

At this point one would typically mention the *Itô-* and *Feynman-Kac* formula and showcase how these can be used to derive an explicit expression for the hedge ratios in the *martingale representation theorem*. However these results all relies on the contingent claims being *Markovian*, which is not sufficient to encompass the class of derivatives used in the thesis. To gain an understanding of how to hedge *pathwise derivatives* we need to extend these results to functionals of the entire path. This is the purpose of appendix C.

C.2 Extension to Functionals

The classical Itô-formula assumes that the transformation of the Itô-process is only a function of the current value of the process. This is sufficient to compute the dynamics of Markovian derivatives as their price process can be proven to only depend on the time and current value of the underlying instrument. Once the class of derivatives is extended to include pathwise derivatives, that is derivatives whose value depends on the entire history of the underlying instrument so far, we lose the ability to express the dynamics. The obvious issue lies in the fact that the transformation is no longer a function of time and space, but a *functional*.

A function $f : [0,t] \longrightarrow \mathbb{R}$ is said to be càdlàg if it is right-continuous with left limits. If we let Λ_t denote the space of càdlàg functions on $[0,t]$ we can define the space

$$\Lambda \triangleq \bigcup_{0\leqslant t\leqslant T} \Lambda_t.$$

We naturally define a functional as a transformation $\phi : \Lambda \longrightarrow \mathbb{R}$. The obvious example of such a functional is the payoff of a pathwise derivative as it associates a real number to the realisation of a path $\widetilde{X}_T(\omega)$, which we already argued is a member of $\Lambda_T \subset \Lambda$. This can further be extended to the value of the pathwise derivative as this associates a real number to the realisation of a path up to some time t.

In order to extend the Itô-formula to functionals, the idea of continuity and differentiability must be formalised. For this, the idea of a distance between two members of Λ are necessary. While members of Λ are fundamentally just càdlàg functions, one would naturally draw on the idea of function norms. However, the domains of the functions do not necessarily have to be the same interval, so care must be taken. We thus define the distance between two members $f_t \in \Lambda_t$, $g_s \in \Lambda_s$ for $t \leqslant s$ as

$$d(f_t, g_s) = \|f_{t,s-t} - g_s\|_\infty + s - t,$$

where $f_{t,s-t}$ is the extension of f to the domain $[0,t]$ achieved by fixing the endpoint $f_t(t)$ across the interval $[t,s]$. More precisely, for some member $f_t \in \Lambda_t$ we define the extended member $f_{t,\delta t} \in \Lambda_{t+\delta t}$ as

$$f_{t,\delta t}(s) = \begin{cases} f_t(s) & 0 \leqslant s \leqslant t \\ f_t(t) & t < s \leqslant t + \delta t \end{cases}.$$

Similarly, we can define the shifted member $f_t^h \in \Lambda_t$ as $f_t(s)$ for $s < t$ and $f_t(t) + h$ for $s = t$. Intuitively, $f_{t,\delta t}$ can be considered a shift in time and f_t^h a shift in space. A functional ϕ is then continuous at $f_t \in \Lambda_t$ if there for all $\epsilon > 0$ exists an $\delta > 0$ such that for all $g_s \in \Lambda_s$ it holds that

$$d(f_t, g_s) < \delta \Leftrightarrow |\phi(f_t) - \phi(g_s)| < \epsilon.$$

Naturally, we further say that ϕ is continuous if it is continuous for all $f \in \Lambda$. With the definitions at hand formalising the notion of differentiability is straightforward

$$\begin{aligned} \nabla_x \phi(f_t) &\triangleq \lim_{h \to 0} \frac{\phi\left(f_t^h\right) - \phi(f_t)}{h}, \\ \nabla_{xx} \phi(f_t) &\triangleq \lim_{h \to 0} \frac{\nabla_x \phi\left(f_t^h\right) - \nabla_x \phi(f_t)}{h}, \\ \nabla_t \phi(f_t) &\triangleq \lim_{\delta t \to 0^+} \frac{\phi(f_{t,\delta t}) - \phi(f_t)}{\delta t}. \end{aligned}$$

The similarities to regular derivatives of functions is obvious. These derivatives are linear and satisfy the product and chain rule[50].

We are now ready to move onto a probabilistic view and consider the càdlàg functions as realisations of a path of some Itô-process X_t as defined in equation (C.1.1). Let the path of an Itô-process be denoted as

$$\widetilde{X}_t(\omega) \triangleq \{X_s(\omega) | 0 \leqslant s \leqslant t\}.$$

As already mentioned, the transformation $t \mapsto X_t(\omega)$ is continuous and therefore trivially càdlàg on $[0,t]$. We can therefore consider the path as a transformation $\widetilde{X} : [0,t] \times \Omega \longrightarrow \Lambda_t$. A functional evaluated at a path is thus a map $(t,\omega) \mapsto \phi\left(\widetilde{X}_t(\omega)\right)$ and therefore fits into the setup from chapter C.1 as long as it satisfies the regularity conditions.

The main result of [50] is a generalisation of Itô's formula for functionals of paths. Let X_t be an Itô-process and let ϕ be a smooth[1] functional. Then for all $T \geqslant 0$ it holds that

$$\phi\left(\widetilde{X}_T\right) = \phi\left(\widetilde{X}_0\right) + \int_0^T \nabla_t \phi\left(\widetilde{X}_t\right) dt + \int_0^T \nabla_x \phi\left(\widetilde{X}_t\right) dX_t + \frac{1}{2} \int_0^T \nabla_{xx} \phi\left(\widetilde{X}_t\right) d\langle X \rangle_t. \tag{C.2.1}$$

The formula is a generalisation of the classical formula as the derivatives coincide if $\phi\left(\widetilde{X}_t\right) = h(t, X_t)$. The paper further derives an analogous extension of the classical Feynman-Kac formula. This result is however not needed in this thesis and is therefore omitted.

[1] A functional ϕ is said to be smooth if it is differentiable in time and twice differentiable in space with these derivatives themselves continuous.

Bibliography

[1] William A McGhee. An Artificial Neural Network Representation of the SABR Stochastic Volatility Model. Working Paper, NatWest Markets, December 2018.

[2] Patrick S Hagan, Deep Kumar, Andrew S Lesniewski, and Diana E Woodward. Managing Smile Risk. *The Best of Wilmott*, 1:249–296, 2002.

[3] Blanka Horvath, Aitor Muguruza, and Mehdi Tomas. Deep Learning Volatility: A Deep Neural Network Perspective on Pricing and Calibration in (Rough) Volatility Models. *Quantitative Finance*, 21(1):11–27, January 2021.

[4] Christian Bayer, Peter Friz, and Jim Gatheral. Pricing Under Rough Volatility. *Quantitative Finance*, 16(6):887–904, June 2016.

[5] Hans Bühler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep Hedging. *arXiv:1802.03042 [math, q-fin]*, February 2018.

[6] Brian Huge and Antoine Savine. Differential Machine Learning. *arXiv:2005.02347 [cs, q-fin]*, September 2020.

[7] Tomas Björk. *Arbitrage Theory in Continuous Time*. Oxford university press, 2009.

[8] Bernt Øksendal. Stochastic Differential Equations. In *Stochastic Differential Equations: An Introduction with Applications*. Springer, Berlin, Heidelberg, 2003.

[9] Hans Föllmer and Thomas Knispel. Convex Risk Measures: Basic facts, Law-invariance and Beyond, Asymptotics for Large Portfolios. In *World Scientific Handbook in Financial Economics Series*, volume 4, pages 507–554. WORLD SCIENTIFIC, July 2013.

[10] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. *arXiv:1811.03378 [cs]*, November 2018. arXiv: 1811.03378.

[11] Magnus Frandsen and Tobias Pedersen. Applications of Deep Learning in Option Pricing and Calibration. Technical report, University of Copenhagen, April 2020.

[12] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer Feedforward Networks with a Nonpolynomial Activation Function can Approximate Any Function. *Neural Networks*, 6(6):861–867, January 1993.

[13] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal Approximation of an Unknown Mapping and its Derivatives using Multilayer Feedforward Networks. *Neural Networks*, 3(5):551–560, January 1990.

[14] Matus Telgarsky. Representation Benefits of Deep Feedforward Networks. *arXiv:1509.08101 [cs]*, September 2015.

[15] Yuxi Li. Deep Reinforcement Learning. *ArXiv*, 2018.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017.

[18] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the Convergence of Adam and Beyond. *arXiv:1904.09237 [cs, math, stat]*, April 2019.

[19] Martín Abadi, P. Barham, J. Chen, Z. Chen, Andy Davis, J. Dean, M. Devin, Sanjay Ghemawat, Geoffrey Irving, M. Isard, M. Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, D. Murray, B. Steiner, P. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Y. Yu, and Xiaoqiang Zhang. TensorFlow: A System for Large-scale Machine Learning. In *OSDI*, 2016.

[20] Leslie N. Smith. Cyclical Learning Rates for Training Neural Networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521(7553):436–444, May 2015.

[22] Xavier Glorot and Yoshua Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, March 2010.

[23] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[24] Siddharth Krishna Kumar. On Weight Initialization in Deep Neural Networks. *arXiv:1704.08863 [cs]*, May 2017.

[25] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient Backprop. *Neural Networks: Tricks of the Trade*, pages 9–48, 2012.

[26] I. T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer-Verlag, New York, 1986.

[27] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, March 2015.

[28] Alexander Sokol and Anders Rønn-Nielsen. *Advanced Probability*. University of Copenhagen: Department of Mathematical Sciences, 2013.

[29] Fischer Black and Myron Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81(3):637–654, May 1973.

[30] Klaus Bjerre Toft. On the Mean-Variance Tradeoff in Option Replication with Transactions Costs. *The Journal of Financial and Quantitative Analysis*, 31(2):233–263, 1996.

[31] Hayne E. Leland. Option Pricing and Replication with Transactions Costs. *The Journal of Finance*, 40(5):1283–1301, 1985.

[32] Yuri M. Kabanov and Mher M. Safarian. On Leland's Strategy of Option Pricing with Transactions Costs. *Finance and Stochastics*, 1(3):239–250, July 1997.

[33] Yonggan Zhao and William T. Ziemba. Hedging Errors with Leland's Option Model in the Presence of Transaction Costs. *Finance Research Letters*, 4(1):49–58, March 2007.

[34] Stewart Hodges and Anthony Neuberger. Optimal Replication of Contingent Claims Under Transaction Costs. *The Review of Futures Markets*, 8(2), 1989.

[35] A. E. Whalley and P. Wilmott. An Asymptotic Analysis of an Optimal Hedging Model for Option Pricing with Transaction Costs. *Mathematical Finance*, 7(3):307–324, 1997.

[36] Jan Palczewski, Rolf Poulsen, Klaus Reiner Schenk-Hoppé, and Huamao Wang. Dynamic Portfolio Optimization with Transaction Costs and State-Dependent Drift. *European Journal of Operational Research*, 243(3):921–931, June 2015.

[37] F. Longstaff and Eduardo S. Schwartz. Valuing American Options by Simulation: A Simple Least-Squares Approach. *The Review of Financial Studies*, 14(1):113–147, 2001.

[38] Paul Glasserman. *Monte Carlo Methods in Financial Engineering*. Stochastic Modelling and Applied Probability. Springer-Verlag, New York, 2003.

[39] Antoine Savine. Fuzzy Logic for Financial Derivatives, 2016.

[40] Mark Broadie and Paul Glasserman. Estimating Security Price Derivatives Using Simulation. *Management Science*, 42(2):269–285, February 1996.

[41] Andrei Nikolaevich Tikhonov. On the Solution of Ill-posed Problems and the Method of Regularization. In *Doklady Akademii Nauk*, volume 151, pages 501–504. Russian Academy of Sciences, 1963. Issue: 3.

[42] Magnus Frandsen, Tobias Pedersen, and Rolf Poulsen. Delta Force: Option Pricing with Differential Machine Learning. Working Paper, University of Copenhagen, 2021.

[43] George AF Seber and Alan J. Lee. *Linear Regression Analysis*, volume 329. John Wiley & Sons, 2012.

[44] M. Giles and P. Glasserman. Smoking Adjoints: Fast Evaluation of Greeks in Monte Carlo Calculations. *Risk*, 2006.

[45] Blanka Horvath, Josef Teichmann, and Zan Zuric. Deep Hedging under Rough Volatility. *ArXiv*, 2021.

[46] Peter Carr, Katrina Ellis, and Vishal Gupta. Static Hedging of Exotic Options. *The Journal of Finance*, 53(3):1165–1190, June 1998.

[47] Jim Gatheral. *The Volatility Surface: A Practitioner's Guide*. John Wiley & Sons, March 2011.

[48] Robert C. Merton. Option Pricing when Underlying Stock Returns are Discontinuous. *Journal of Financial Economics*, 3(1):125–144, January 1976.

[49] C. He, J. S. Kennedy, T. F. Coleman, P. A. Forsyth, Y. Li, and K. R. Vetzal. Calibration and Hedging under Jump Diffusion. *Review of Derivatives Research*, 9(1):1–35, January 2006.

[50] Bruno Dupire. Functional Itô Calculus. *Quantitative Finance*, 19(5):721–729, May 2019.

[51] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[52] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, October 2000.

[53] Kyunghyun Cho, B. V. Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. *EMNLP*, 2014.

[54] J. Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *ArXiv*, 2014.

[55] Rolf Poulsen. Barrier Options and Their Static Hedges: Simple Derivations and Extensions. *Quantitative Finance*, 6(4):327–335, August 2006.

[56] David Lando. *Credit Risk Modeling: Theory and Applications*. Princeton University Press, 2009.