Deep learning approach to hedging

Candidate Number: 1023650

University of Oxford

A thesis submitted in partial fulfillment of the MSc in

Mathematical and Computational Finance

June 21, 2018

Acknowledgements

I would like to express my sincere gratitude to my supervisor for his guidance and encouragement at all stages of this thesis.

A very special appreciation is due to my family for their understanding, patience and support.

Abstract

The principal aim of this thesis is to investigate the potential of state-of-the-art deep learning techniques applied in the context of hedging. We con rm an ability of a neural network algorithm to not only replicate theoretical hedging parameters but also to design strategies that are optimal in the light of various criteria. We investigate conjectured exibility of the approach and nd limitations that arise with increasing diversity of underlying processes. Finally, the algorithm is used as a nancial engineering tool for the problem of hedging under the presence of transaction costs.

Contents

Chapter 1

Introduction

As presented by Black and Scholes [5], the price of a contingent claim is equal to the smallest amount of capital needed to hedge this claim. In nancial literature this is known as pricing-hedging duality which gave a momentum both to the nance industry and research in mathematical nance. However, the involved and in sense unpredictable nature of any nancial market makes it challenging to develop a consistent hedging theory for incomplete or markets with frictions. For example, one of the basic assumptions of the vast majority of proposed models is that the underlying dynamics of prices are xed and known. This leads to a strong dependence of both the price and hedging parameters on the chosen model. In this setting any hedging strategy, which is implemented on the real data is prone to multiple errors that may emerge from the various assumed simpli cations. Even though the impact of mis-calibrated model parameters or simplifying assumptions can be to some extent measured and appropriately dealt with, it is tremendously di cult to quantify the wrong choice of a model in the rst place.

In order to address these problems, and as a part of the post 2008 crisis aftermath, robust methods grew in popularity. In order to hedge a given position a trader is looking for the cheapest model independent superhedge - a trading strategy that does not necessarily replicate the payo of a contingent claim, but whose future time value is greater than the one of the risky position.

Another di culty arising in the context of theoretical modelling of hedging is the presence of market frictions such as transaction costs or liquidity constraints. Multiple attempts have been made to propose mathematically tractable models. These e orts were initiated by Leland [15] in 1985. His idea, discussed in greater detail later in this thesis, was based on an attempt to modify the idea of delta-hedging of Black and Scholes under the presence of small transaction costs and regular rebalancing. Howard, Whalley and Willmott [10]

extended this idea to arbitrary option portfolio and showed that the value of such portfolio is the solution to a modified nonlinear version of the Black-Scholes PDE. Davis, Panas and Zariphopoulou [7] laid theoretical foundations for the utility based approach to hedging which were later extended by Whalley and Whillmot [17] who proposed an easy to implement and asymptotically optimal strategy of no-transaction region. All of these methods suffer, however, from strong assumptions on the underlying dynamics.

In this paper we attempt to address these deficiencies by use of state-of-art deep learning techniques. The application of deep neural networks to the problem of hedging was proposed in [6] where authors lay down theoretical foundations and provide compelling examples for the effectiveness of the algorithm. As the proposed method is model independent we consider the algorithm to be a potentially effective robust tool.

The thesis is structured as follows. First we expand upon the structure of the model and justify its properties. Then we use the Black Scholes model setting as a benchmark for the algorithm. Next, in Chapter 4 we fit out model to a diverse set of paths calibrated to the same market in order to get insight into the conjectured robustness. Then, in the last chapter, we apply the algorithm in a setting with transaction costs and present its ability as a financial engineering tool.

Chapter 2

Theoretical prerequisites

2.1 Setting: Discrete time market

In this paper we consider a discrete-time nancial market with horizon T and trading dates $0 = t_0 < t_1 < \quad < t_n = T$. We x a probability space and denote by X the set of all probability measures on . Any new information that becomes available at time t_k is denoted by I_k with values in $\mathbb{R}^r$. This may include prices of any tradable instruments, liquidity limitations, trading signals, risk limits, various statistics etc. Therefore we introduce the process $I = (I_k)_{k=0,\dots,n}$ which generates the ltration $F = (F_k)_{k=0,\dots,n}$ which is the richest available feature set for any decision made at t_k.

Let us assume that the market consists of d hedging instruments with prices given by an $\mathbb{R}^d$-valued F-adapted stochastic process $S = (S_k)_{k=0,\dots,n}$. No assumptions are made both on existence of an equivalent martingale measure or the availability of hedging instruments at a given point in time. The portfolio of contingent claims that the agent attempts to hedge is denoted by a random variable Z that is F_T measurable. For the purpose of convenience we assume that this is a portfolio of European derivatives and that we take a short position in it.

In order to hedge the claim Z we trade in S using an $\mathbb{R}^d$-valued F-adapted stochastic process $\delta = (\delta_k)_{k=0,\dots,n-1}$ where $\delta_k = (\delta_k^1, \dots, \delta_k^d)$. In particular, δ_m^i denotes the agent's position in the ith asset at time t_k. Since all trading is self- nanced, we may want to add some cash at time t_0. The agent's terminal wealth is given by

$$P/L(Z, p_0, \delta) = \quad Z + p_0 + (\delta \quad S)_T \quad C_T(\delta) \tag{2.1}$$

where

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

denotes the trading gains and losses,

$$C_T(\delta) = \sum_{k=0}^{n} c_k(\delta_{k+1} \quad \delta_k)$$

is the total cost of trading up to maturity and p_0 *2* R is the initial capital injection.

2.2 Convex risk measures

In this general setting the market can be incomplete, in which case there is a need to specify an optimality criterion in order to determine an acceptable price of a contingent claim. Such a price would be a minimal cash injection that is needed to be added to a position to implement the optimal hedge in such a way that the overall position becomes acceptable in the light of agent's preferences. We encode this acceptability criterion by the use of convex risk measures. Let us quickly recall ther de nition

Definition 1. *Let* X, X_1, X_2 *2 X be positions in assets. We call* ρ: *X ! R a convex risk measure if it is:*

1. *Monotone decreasing:* $X_1 \quad X_2$ *)* $\rho(X_1) \quad \rho(X_2)$*,*
2. *Convex:* $\rho(\alpha X_1 + (1 \quad \alpha)X_2) \quad \alpha\rho(X_1) + (1 \quad \alpha)\rho(X_2)$ *for* α *2* $[0,1]$*,*
3. *Cash-Invariant:* $\rho(X + c) = \rho(X) \quad c$ *for any* c *2* R*.*

For a convex measure ρ we de ne an optimisation problem

$$\pi(X) = \inf_{2H} \rho(X + (\delta \quad S)_T \quad C_T(\delta)) \tag{2.2}$$

where *H* denotes the space of possible trading strategies in accordance with market restrictions in trading (eg. due to unavailability of certain hedging instruments at given time).

Proposition 1 (see [6], Proposition 3.2, p. 7.)**.** *The functional* π *de ned above is monotone decreasing and cash-invariant. If* C_T *and H are convex, then* π *is a convex risk measure.*

The optimal hedging strategy δ *2 H* is then de ned as minimizer of (2.2). One can think of $\pi(x)$ as the minimal amount of money that has to be added to a portfolio in order to make in acceptable in the light of a risk measure ρ. De ning this as the minimal price excludes the possibility that having no liabilities may have positive expected value (which may be the case in a presence of positive expected returns under the physical measure). Thus we instead focus on the indi erence price $p(Z)$ which is the amount of cash an agent

charges in order to be indi erent between the position Z and no position in liabilities i.e. $\pi(\ Z + p(Z)) = \pi(0)$ holds. By cash-invariance this is equivalent to

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

In a setting with no frictions in the market it is easily seen that this price is consistent with the price of a replicating portfolio (provided that it exists). For the purpose of hedging we focus on the following class of convex risk measures known as Optimised certainty equivalents:

Definition 2. *Let* l: R ! R *be a loss function (continuous, non-decreasing and convex). We de ne the OCE by:*

$$\rho(X) = \inf_{w2\mathsf{R}} fw + \mathsf{E}[l(\ X \quad w)]g,\ X\ 2\ X. \tag{2.3}$$

There are two speci c representatives of this risk measures family that are of particular importance for the model.

1. For a xed $\lambda > 0$ we set $l(x) = \exp(\lambda x) \quad \frac{1+\log(x)}{}$. This leads, after optimising and inserting back into (2.3), to the following convex risk measure

$$\rho(X) = \frac{1}{\lambda}\log \mathsf{E}[\exp(\ \lambda X)] \tag{2.4}$$

   which is known in the literature as the entropic risk measure.

2. Setting $l(x) = \frac{1}{1}\max(x, 0)$ for $\alpha\ 2\ (0, 1)$ leads to what is known as the average value at risk or expected shortfall.

Let us now recall the de nition of a well-known exponential utility function $U(x) = \exp(\ \lambda x)$. The following result clari es its link to the entropic risk measure.

Proposition 2 (see [6], Lemma 3.6, p. 10.)**.** *Let* $q(Z)$ *be a solution of the following indifference pricing problem*

$$\sup_{2H} \mathsf{E}[U(q(Z) \quad Z + (\delta \ \ S)_T \quad C_T(\delta))] = \sup_{2H} \mathsf{E}[U((\delta \ \ S)_T \quad C_T(\delta))]$$

and π *be de ned as in (2.2). Then it follows that* $q(Z) = p(Z)$*.*

This choice of the agent's preferences is very popular in the modern literature and serves as a basis of multiple models that describe hedging with frictions (see [7], [17]).

2.3 Neural networks

We first recall the definition of a feedforward neural network that is a basic component of our model.

Definition 3. *Let $L, N_0, \ldots, N_L \in \mathbb{N}$, $\sigma_i : \mathbb{R} \to \mathbb{R}$ for any $i = 1, 2, \ldots, L$, let $A_i : \mathbb{R}^{N_{i-1}} \to \mathbb{R}^{N_i}$ be affine functions. A function $F : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ defined as*

$$F(x) = F_L \circ \cdots \circ F_1(x)$$

where

$$F_i = \sigma_i \circ A_i \text{ for } i = 1, 2, \ldots, L$$

is called a neural network.

It is important to note that the activation picture σ_i is applied componentwise. L stands for the number of layers, $N_1, \ldots, N_{L-1}$ are the dimensions of the hidden layers and N_0, N_L of the input and output layers respectively.

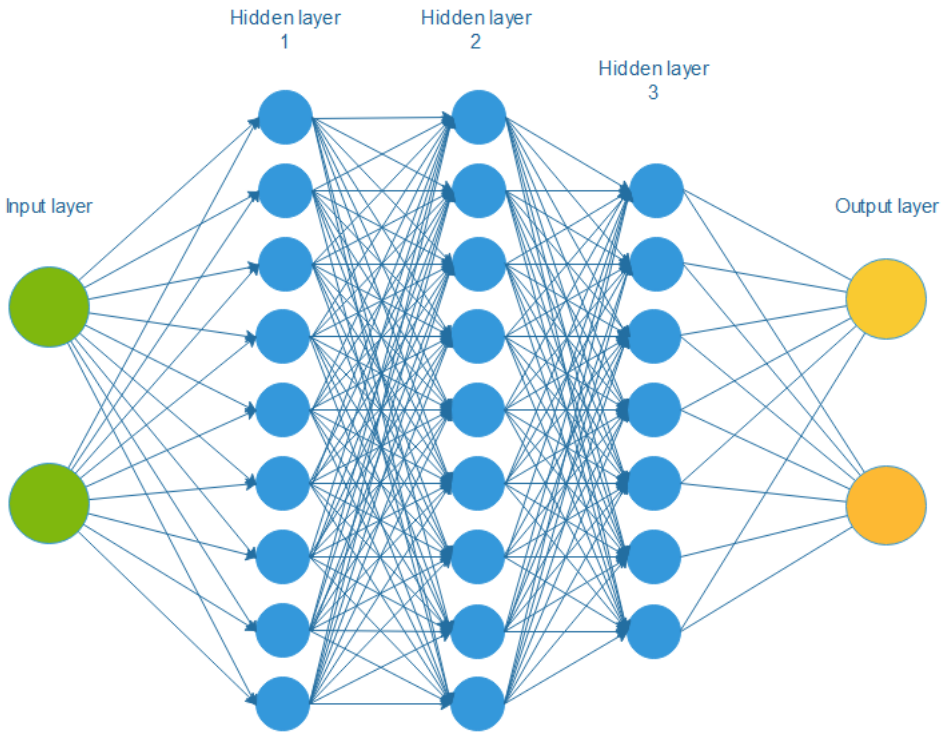

Figure 2.1: An example of a neural network for the following architecture: $L = 4, N_0 = 2, N_1 = N_2 = 9, N_3 = 7$, and $N_4 = 2$.

We also denote by $NN_{1;d_0;d_1}$ the set of all neural networks from R^{d_0} to R^{d_1} with being a set of activation functions for each of the layers. The next result (see [11]) presents the usefulness and power of neural networks.

Theorem 1 (Universal approximation, see [11]). *Suppose that each element of σ is bounded and non-constant. The following statements are true*

1. *For any nite measure μ on $\left(\mathrm{R}^{d0}, B(\mathrm{R}^{d_0})\right)$ and 1 $p < 1$, the set $NN_{1;d_0;d_1}$ is dense in $L^p\left(\mathrm{R}^{d0}, \mu\right)$.*
2. *If in addition every element of is continuously di erentiable over* R*, then $NN_{1;d_0;d_1}$ is dense in $C(\mathrm{R}^{d_0})$ for the topology of uniform convergence on compact sets.*

For computational purposes we choose to work with a sequence of subsets of $NN_{1;d_0;d_1}$ that is denoted by $\{NN_{M;d_0;d_1}\}_{M \in \mathrm{N}}$ and has the following properties:

1. $NN_{M;d_0;d_1}$ $NN_{M+1;d_0;d_1}$ for all $M \in$ N,
2. $\bigcup_{M \in \mathrm{N}} NN_{M;d_0;d_1} = NN_{1;d_0;d_1}$,
3. for any $M \in$ N, one has $NN_{M;d_0;d_1} = \{F : \theta \in \ _{M;d_0;d_1}\}$ where $_{M;d_0;d_1} \in \mathrm{R}^k$ for some $k \in$ N.

One could think of the sequence de ned above as of all neural networks with xed architecture parameterised by a vector θ whose number of dimensions depends on M.

Even though most of market processes to be analysed in this work possess the Markov property, we still require our model to be able to take into account all of the actions that took place in the past. This is crucial for the case when there are frictions in a market such as transaction costs. In order to introduce these dependencies into the modelling process we choose to work with Recurrent Neural Networks. One could think of RNN as of a sequence of regular feed-forward Neural Networks where hidden layers not only output values to next layers horizontally, but also pass them (after potentially applying di erent activation function) to a similar layer at the next time step. This structure makes it possible to analyse time series.

Convex risk measures de ned above are used as loss functions in the model. Before we proceed to layout the speci c structure of the algorithm, we quote the crucial result that proves the feasibility of this method.

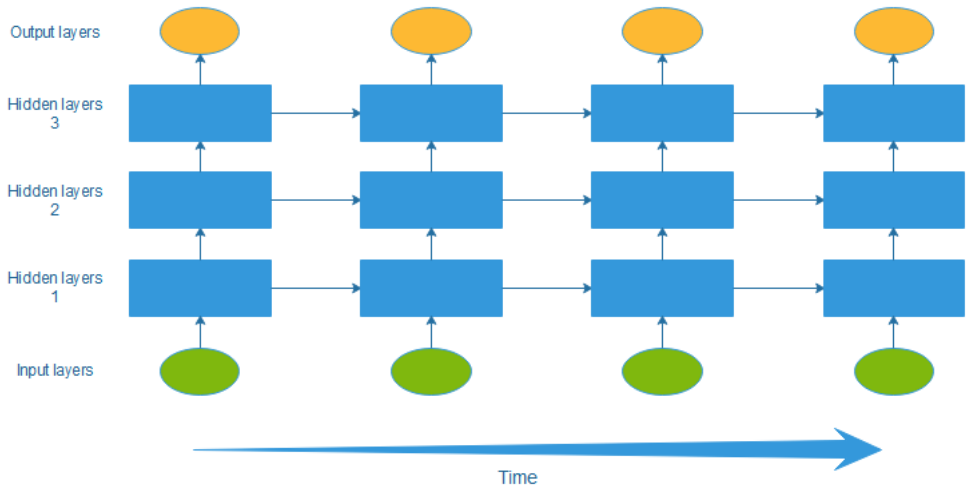

Figure 2.2: An example of a recurrent deep neural network with 3 hidden layers and 4 time steps.

Theorem 2 (see [6], Prop. 4.2, p. 13.). *Let H_M denote the subspace of all strategies δ that can be obtained with the use of networks belonging to $NN_{M;d_0;d_1}$. Then*

$$\lim_{M \to 1} \pi^M(X) = \lim_{M \to 1} \inf_{\delta \in H_M} \rho(X + (\delta \cdot S)_T - C_T(\delta)) = \pi(X).$$

2.4 Numerical approximations of hedging strategies

In order to find a numerical solution for the hedging problem we compute a close-to-optimal hedging strategy $\delta \in H_M$. Recall that H_M denotes a family of neural networks with a fixed architecture. Theorem 2 justifies this approach for a sufficiently flexible network. In order to train the model we first generate a finite number of samples. Finally we define the sample $\Omega = \{\omega_1, \ldots, \omega_N\}$ where $P(\omega_i) = \frac{1}{N}$ for each $i \in \{1, \ldots, N\}$. This space could be understood as a space of underlying asset paths. Unless stated otherwise, we take a training set of size $9 \cdot 10^4$, a validation set of size 10^4, and a test set of size 10^5 respectively[1].

The algorithm has been implemented in Python with the use of TensorFlow to construct and train the neural network alongside with Numpy for the purpose of generating model paths. The exact code can be found in the Appendix. The network parameters are initialised randomly using the method proposed by He [8]. We use the Adam algorithm to train the network [13] with an initial learning rate of 0.0005 that is then adjusted by performance scheduling. Furthermore, we use a batch size of 200 has been used. Selu [14] activation has

[1] In our case, since we can simulate our data without constraints, the size of the test set is arbitrary. We have decided to use such a relatively big number in order to make the graphical presentation clearer.

been applied to the hidden layers [14] and the identity activation was use for the output layer to allow the maximal exibility of the algorithm.

In order to train the network we specify the loss function to be the OCE of (2.3). This function does not return its value based on data labels but it veri es the outputs of consecutive constituents of a recurrent deep neural network and performs back-propagation though time to update parameters (see [4]).

Chapter 3

Black - Scholes model

3.1 Theoretical results

The Black-Scholes model for option pricing seems to have been derived with the idea of delta hedging in mind. The usefulness of this approach comes from the fact that theoretically, under assumptions of a friction-less market and continuous rebalancing, it is possible to completely hedge-out any risk associated with writing a contingent claim since we can replicate its payo with probability 1. A direct consequence of that is another rather remarkable property of the model: the value of a claim does not depend on a drift term of the underlying asset dynamics.

Under the Black-Scholes model the theoretical price of a European call is given by

$$V(t, S_t) = N(d_+)S_t \quad N(d \ )e^{\ r(T \ t)}K$$

where

$$d_+ = \frac{1}{\sigma\sqrt{T \ t}}\left(\log\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T \ t)\right)$$
$$d \ = d_+ \ \sigma\sqrt{T \ t}$$

and $N(\ )$ denotes cumulative distribution function of a standard normal random variable.

The crucial quantity that emerges during the derivation of this formula is

$$_t = \frac{\partial V(t, S_t)}{\partial S_t} = N(d_+).$$

If at any time t 2 $[0, T]$ agent holds $_t$ units of the underlying asset, then she is able to replicate the payo completely with probability 1. In a real market situation, even if no frictions are present, continuous trading is not possible. Hence, there is an inevitable hedging error associated with the inability to keep the portfolio in a delta-neutral state for the whole time of the investment.

3.2 Benchmark: discretized Black-Scholes model

In order to verify the performance of the algorithm we intend to compare it with a discretized Black-Scholes model hedge. We have chosen a time horizon of 30 trading days and allowed daily rebalancing. It follows that T = 30/365, n = 30, t_i = i/365, and δ_{t_i} *2* R. The structure of each component of the recurrent network is as follows: L = 4, N_0 = 1, N_1 = N_2 = 50, N_3 = 30, and N_4 = 1, where L denotes the number of layers and N_i stand for the number of nodes in the layer i. We used expected shortfall as a loss function.

$$ES\ (X) = \frac{1}{1\quad\alpha}\int_0^1 \quad VaR_u(X)\mathrm{d}u, \tag{3.1}$$

where $VaR_u(X)$ denotes the value at risk of a random variable X at the level of u given by

$$VaR_u(X) = \inf\{x\ 2\ \mathrm{R}\colon F_X(x) > u\}$$

where F_X denotes the CDF of X.

We choose the following values of the parameter: α *2* $\{0.5, 0.75, 0.95, 0.99\}$. We assume that the risk-free rate is at the level of 0 and the annual volatility is 0.2. Finally we hedge a portfolio of one short position in an European At the Money Call option for the initial underlying value S_0 = 100. Below we compare the model discretized Black-Scholes hedge and neural network (NN) hedging strategies estimated by our model.

The performance of the algorithm varies tremendously with di erent choices of the parameter α. For the values α *2* $\{0.95, 0.99\}$ our model, even though performing better than the discretized Black-Scholes model in terms of preventing large losses, fails to capture its general structure. On the other hand for α = 0.5 it can be seen that the network is able to replicate the Black-Scholes hedge reasonably well. However, one could notice a light skewness of the NN hedge and larger extreme losses. The latter is quanti ed in the following table that compares the model and neural network (with ES_{50} loss function) hedge in terms of the mean pro t/loss and percentiles of empirical distribution.

Table 3.1: Comparison of the discretized Black-Scholes hedge (theoretical) and neural network strategy with the loss function ES_{50}

Model	Mean	VaR_{99}	VaR_{95}	VaR_{90}	Var_{80}	VaR_{50}
Theoretical	0.0392	-0.9301	-0.5404	-0.3821	-0.2208	0.0394
ES_{50} NN	0.0398	-1.0525	-0.6089	-0.4208	-0.2334	0.0394

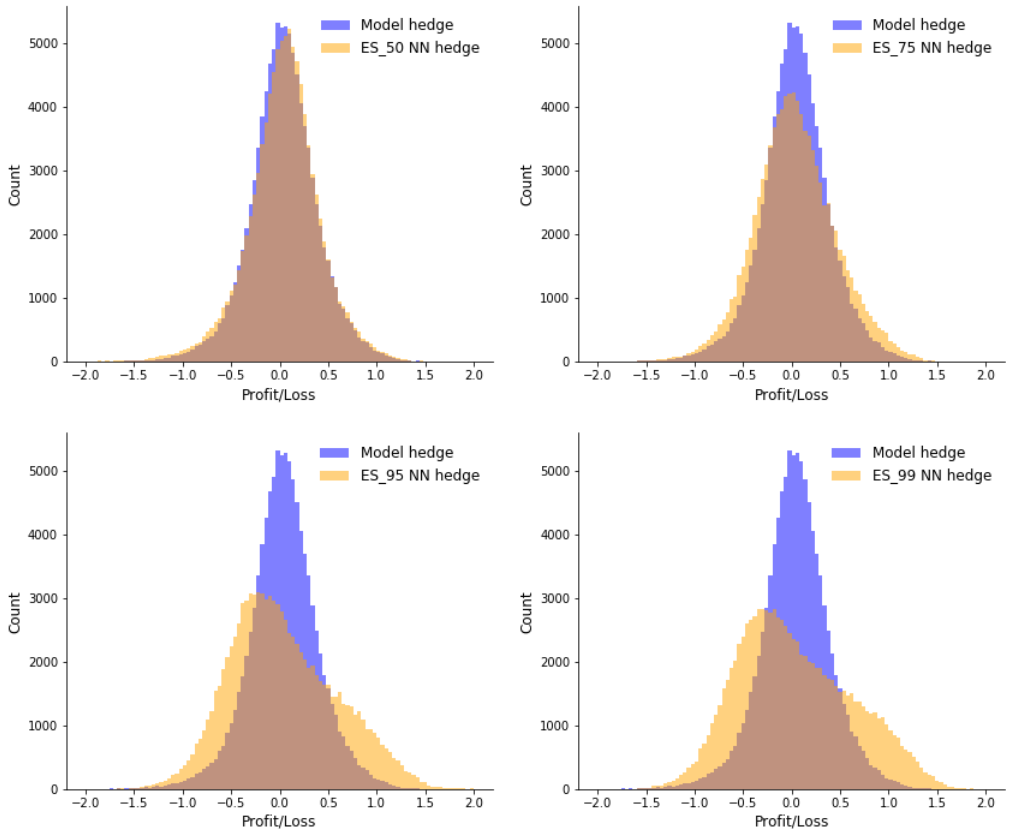

Figure 3.1: Comparison of the theoretical Black-Scholes hedge (blue) and the one returned by the neural network (orange) evaluated for di erent loss functions. ES NN hedge stands for a hedging strategy returned by the network that optimised expected shortfall at the level of α.

In order to attempt to solve this inaccuracy we propose a modi ed loss function of the following form

$$l_{\alpha,\beta}(X) := \frac{1}{1+\beta}\big(ES_{0.5}(X) + \beta ES_{\alpha}(X)\big). \tag{3.2}$$

Convexity of the associated risk measure follows directly from the properties of expected shortfall.

Next we perform a grid search in order to tune the hyperparameters α and β. The search space is a Cartesian product of $A = \{0.99, 0.95, 0.9, 0.75\}$ and $B = \{0.01, 0.02, 0.05, 0.1\}$. We nd that the optimal value of parameters is $(\alpha, \beta) = (0.95, 0.05)$ for which we can see a signi cant improvement compared to the initial ES_50 network hedge. We shall refer to this loss function as the mixed expected shortfall.

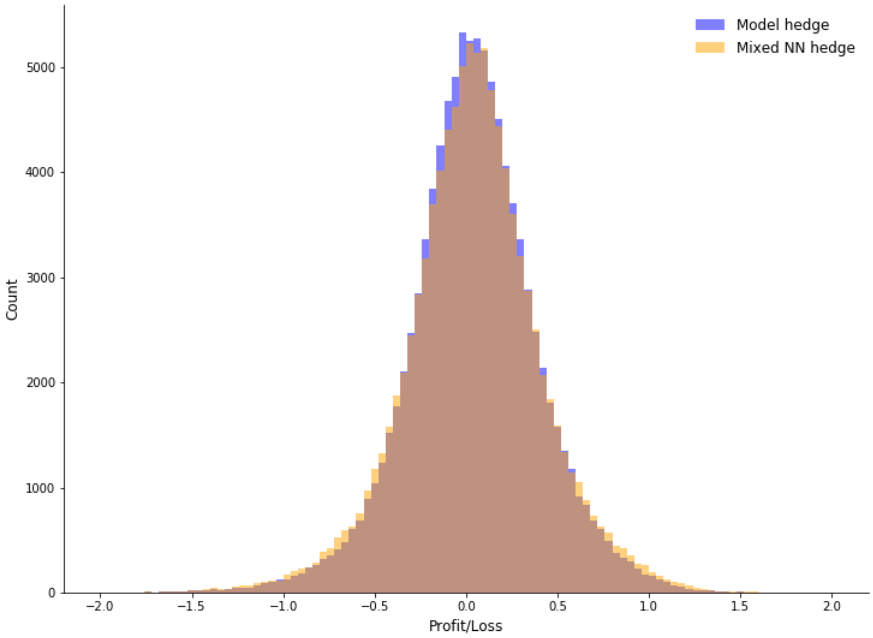

Figure 3.2: Pro t and loss distribution of the improved network strategy compared to the theoretical one.

Table 3.2: Improved hedges comparison. The Mixed NN stand of the hedging strategy obtain by using a (3.2) as a loss function.

Model	Mean	VaR99	VaR95	VaR90	Var80	VaR50
Theoretical	0.0392	-0.9301	-0.5404	-0.3821	-0.2208	0.0394
Mixed NN	0.0398	-0.9612	-0.5762	-0.4096	-0.2351	0.0419

The above result con rms the ability of the algorithm to replicate the theoretical Black-Scholes hedge. We now discuss the strategy that was chosen when expected shortfall with α above 0.9 was minimised. In order to take a closer look at these strategies we chose 4 price paths for which the NN hedging error was signi cantly di erent from the BS one. Outcomes for these cases are summarised below.

Table 3.3: Hedging error for the chosen scenarios.

Model	Path 1	Path 2	Path 3	Path 4
Mixed NN	0.09611	-0.0719	0.0140	0.3301
ES_{99} NN	{0.3745	0.3056	-0.3136	-0.0626

Figure 3.3: Comparison of hedge parameters obtained with various loss function parameters.

We analyse two scenarios where the NN hedge obtained by optimising a restrictive risk measure (such as ES_{99} or ES_{95}) outperforms the BS one and two scenarios where the opposite can be observed. One can notice a speci c relative behaviour of hedging strategies. Up to a certain point, which usually is roughly in the middle of the time horizon, both types of hedges are almost identical. As the maturity time approaches, the Black-Scholes delta (and hence the Mixed NN one as well) approaches either 1 or 0, depending on the value of the underlying asset close to expiry. On the contrary, hedging parameters base on the more restrictive criterion are closer to the value of 0.5 and could, heuristically, be characterised

as retaining some sort of exibility.

There are two scenarios that require special attention. When the asset price is growing steadily (Cases 1 and 3) the ES_{99} NN hedge is outperformed by the Mixed NN hedge (and hence by the Black-Scholes delta-hedge). This is clear since the more restrictive trading leads to poorer replication and there are no bene ts from keeping hedging parameters lower. On the other hand, when the asset price moves rapidly against the previous trend just before the expiry date, the bene ts from exibility are signi cant. Indeed, in Case 4 one can see that keeping hedge parameters lower than it is suggested by the theoretical strategy led to a signi cant improvement of performance at the very end of the investment horizon.

Keeping the quasi-deltas closer to 0.5 prevents severe losses caused by the lack of exibility of discretized Black-Scholes hedging parameters close the the expiry. This is then clear that, if we would decrease a rebalance interval, then we would also reduce this e ect and, as a consequence, the di erence between the two strategies would be smaller. This coincides with theoretical result - as δt approaches 0 we are getting closer to the perfect-hedge situation where for all possible values of α the expected shortfall would lead to the same optimal strategy of continuous delta-hedging.

3.3 Robustness of the Black-Scholes formula

There is a remarkable robustness property of the BS delta hedging when the implied volatility does not coincide with the real underlying asset dynamics (see [12]). Assume that the physical dynamics are as follows

$$\mathrm{d}S_t = \mu_t S_t \mathrm{d}t + \sigma_t S_t \mathrm{d}W_t,$$

where μ_t and σ_t are F_t-adapted processes. The market is not assumed to be complete and the underlying ltration F_t can be bigger that $\sigma(S_t)$. Furthermore, we assume no transaction costs. Suppose that an agent sold an option with a payo $g(S_T)$ at $t = 0$ using di erent volatility σ. She uses then uses this quantity to implement a continuous delta hedging strategy. This means that the value of the option is a solution to the following PDE:

$$\frac{\partial V(t,S_t)}{\partial t} + rS_t\frac{\partial V(t,S_t)}{\partial S_t} + \frac{1}{2}\sigma^2 S_t^2\frac{\partial^2 V(t,S_t)}{\partial S_t^2} \quad rV(t,S_t) = 0,$$
$$V(T,S_T) = g(S_T).$$

Denote the agent's position in the underlying asset by H_t. Then her wealth process X_t is governed by the SDE

$$dX_t = H_t \mathrm{d}S_t + r(X_t - H_t S_t)\mathrm{d}t.$$

De ne a tracking error as

$$R_t = X_t - V(t, S_t).$$

A straightforward application of Ito's lemma and noting that $H_t = \frac{@V(t;S_t)}{@S_t}$ lead to the following result

$$\begin{aligned}
\mathrm{d}R_t &= \mathrm{d}X_t - \mathrm{d}V(t, S_t) \\
&= H_t \mathrm{d}S_t + r(X_t - H_t S_t)\mathrm{d}t - \left(\frac{\partial V(t,S_t)}{\partial t} + \frac{1}{2}\sigma_t^2 S_t^2 \frac{\partial^2 V(t,S_t)}{\partial S_t^2}\right)\mathrm{d}t - \frac{\partial V(t,S_t)}{\partial S_t}\mathrm{d}S_t \\
&= rX_t\mathrm{d}t - \left(\frac{\partial V(t,S_t)}{\partial t} + rS_t\frac{\partial V(t,S_t)}{\partial S_t} + \frac{1}{2}\sigma_t^2 S_t^2 \frac{\partial^2 V(t,S_t)}{\partial S_t^2}\right)\mathrm{d}t \\
&= rR_t\mathrm{d}t - \left(\frac{\partial V(t,S_t)}{\partial t} + rS_t\frac{\partial V(t,S_t)}{\partial S_t} + \frac{1}{2}\sigma_t^2 S_t^2 \frac{\partial^2 V(t,S_t)}{\partial S_t^2} - rV(t,S_t)\right)\mathrm{d}t \\
&= rR_t\mathrm{d}t + \frac{1}{2}(\sigma^2 - \sigma_t^2) S_t^2 \frac{\partial^2 V(t,S_t)}{\partial S_t^2}\mathrm{d}t.
\end{aligned}$$

Thus

$$\mathrm{d}(e^{-rt}R_t) = \frac{1}{2}(\sigma^2 - \sigma_t^2) S_t^2 \frac{\partial^2 V(t,S_t)}{\partial S_t^2}\mathrm{d}t. \tag{3.3}$$

Provided that $\frac{@^2V(t;S_t)}{@S_t^2}$ 0, which is usually the case, as long as we price and hedge a contingent claim with σ^2 σ_t^2 for all t 2 $[0, T]$ then

$$e^{-rT}R_T = \frac{1}{2}\int_0^T (\sigma^2 - \sigma_t^2) S_t^2 \frac{\partial^2 V(t,S_t)}{\partial S_t^2}\mathrm{d}t \quad 0 \text{ a.s..}$$

In a discretized setting we cannot expect positive trading errors with probability 1. Instead, what we anticipate is a positive average hedging error. To further verify accuracy of the NN algorithm we use the network trained in Section 2.2 for an annual volatility of 0.2 that for a purpose of this analysis is considered the "true" volatility of the underlying process. We then use the trained network to hedge an European call option whose underlying dynamics have di erent levels of volatility. This can be treated as a situation where a trader uses a di erent value of implied volatility to price and hedge the claim, made a mistake in calibration, or a model error has been made.

Table 3.4: Implied volatility hedges comparison. The "α TV NN" denotes the model that was trained for volatility 0.2 but evaluated at paths generated with volatility α.

Model	Mean	Std	VaR99	VaR95	VaR90	Var80	VaR50
0.05 TV NN	1.7251	0.2748	1.0668	1.2301	1.3404	1.4758	1.7514
0.10 TV NN	1.1640	0.3510	0.4524	0.6115	0.7100	0.8426	1.14987
0.20 TV NN	0.0398	0.3771	-0.9612	-0.5762	-0.4096	-0.2351	0.0419
0.30 TV NN	-1.0848	0.7547	-3.3782	-2.5223	-2.1110	-1.6516	-0.9381
0.40 TV NN	-2.2072	1.3842	-6.2025	-4.8092	-4.1141	-3.3085	-1.9629

Negative mean hedging error for underestimated implied volatility and the opposite result for overestimated IV coincide with the theoretical findings (3.3). What is more, a hedging strategy based on highly underestimated volatility is not capable of replicating a value of the highly volatile portfolio. As the result, with an increasing distance between the true and assumed volatility the outcome starts to resemble the final payoff where the skewness is the natural consequence of having taken a short position in the European Call option.

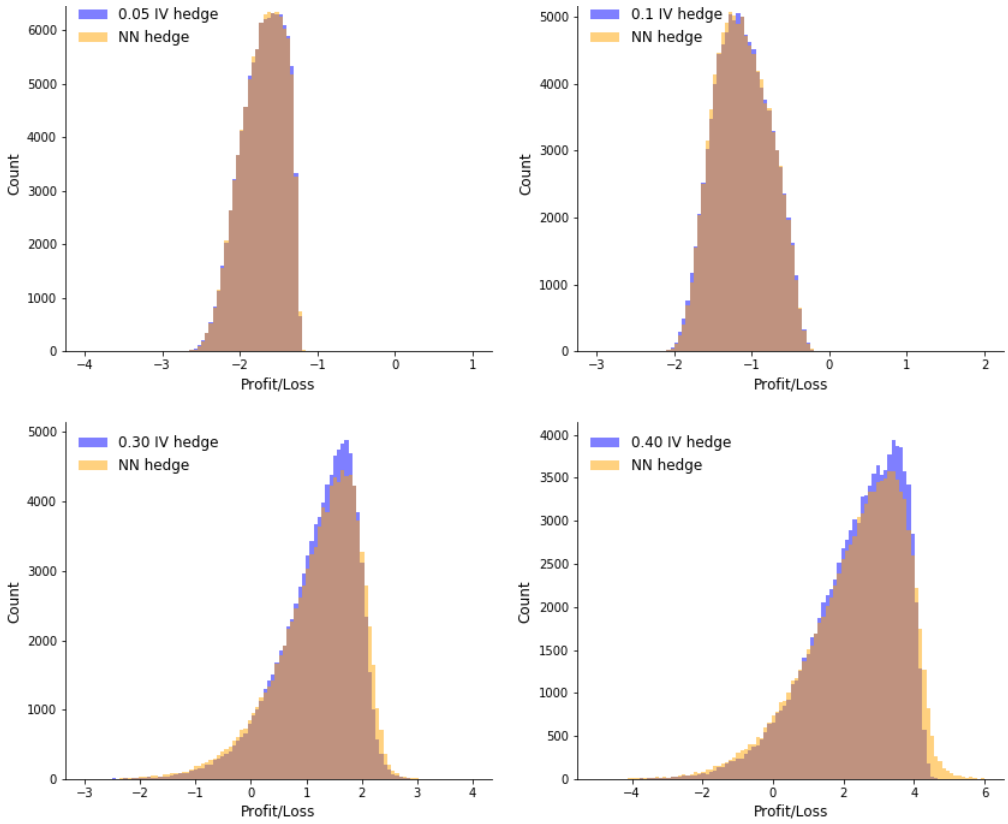

Figure 3.4: Hedging comparison for miscalibrated volatility.

The algorithm has proven itself to be able to replicate the theoretical Black-Scholes parameters with a very high accuracy. This can be seen in very similar quantities of hedging error distributions for both the chosen network hedge and the theoretical one, as illustrated in figure 3.3. What is more, the performance still coincides with theoretical expectations even for a different value of volatility. One can hypothesise, that the algorithms is not only an effective tool for the task of hedging a contingent claim for a given set of market parameters, but could also be used as a robust tool. The latter is further explored in the next chapter.

Chapter 4

Robustness of the neural network hedge

In this chapter we intend to use the neural network in a setting closer to the real market. We have chosen to verify the robustness of the algorithm by tting the model to paths govern by di erent stochastic processes that have been calibrated to the same market.

4.1 Models

4.1.1 Heston model

In his 1993 paper [9] Heston proposed a stochastic volatility model with semi-closed-form solutions. In this model the stock price process is given by the following dynamics

$$dS_t = \mu_t S_t dt + \sqrt{\nu_t} S_t dW_t^1, \ S_0 \quad 0,$$

where ν_t is governed by CIR process

$$d\nu_t = \kappa(\theta \quad \nu_t)dt + \xi\sqrt{\nu_t} dW_t^2, \ \nu_0 \quad 0.$$

Here, W_t^1 and W_t^2 are Brownian motions with a correlation coe cient ρ. If the parameters obey the following condition (know as the Feller condition)

$$2\kappa\theta \quad \xi^2$$

then the process ν_t is strictly positive.

We specify another tradable asset that would be used to complete the market. We choose to work with an idealised variance swap of the following form

$$S_t^2 = N \ E_Q[\int_0^T \nu_u du | F_t]$$

where N 2 N and Q is the risk neutral measure speci ed by the following measure change applied to the physical dynamics SDEs

$$dW_t^{1;Q} = dW_t^1 + \frac{r \quad \mu}{\sigma} dt$$

$$dW_t^{2;Q} = dW_t^2 + \frac{\mu \quad r}{p\overline{\nu_t}} \frac{\rho}{\sqrt{1 \quad \rho^2}} dt$$

that leads to the following process dynamics

$$dS_t = rS_t dt + p\overline{\nu_t} S_t dW_t^{1;Q},$$

$$d\nu_t = \kappa(\theta \quad \nu_t)t + \xi p\overline{\nu_t} dW_t^{2;Q}.$$

This speci c choice of a risk-neutral measure coincides with Heston's original choice for the risk aversion parameter $\lambda = 0$. (see [9]).

It this setup we can easily calculate

$$\begin{aligned} \frac{S_t^2}{N} &= E_Q[\int_0^T \nu_u du j F_t] \\ &= \int_0^t \nu_u du + \int_t^T E_Q[\nu_u j F_t] du \\ &= \int_0^t \nu_u du + \frac{\nu_t \quad \theta}{\kappa}(1 \quad e^{\quad (T \quad t)}) + \theta(T \quad t) \end{aligned}$$

where we use the Fubini's theorem for the second equality and the integrating factor method for the third one.

4.1.2 Bates model

Bates model, which is a natural extension of the Heston model, is a stochastic volatility model where the asset price follows Merton's jump-di usion process. Hence the dynamics are as follows

$$dS_t = \mu_t S_t dt + p\overline{\nu_t} S_t dW_t^1 + J_t dN_t, \; S_0 \quad 0,$$

where N_t is a Poisson process with intensity $\lambda > 0$. J_t is an identically and independently distributed random variable for which it holds that

$$\log(1 + J_t) \text{ s Normal}\Big(\log(1 + \mu_J) \quad \frac{\sigma_J^2}{2}, \sigma_J\Big).$$

The squared volatility SDE remains unchanged

$$d\nu_t = \kappa(\theta \quad \nu_t)dt + \xi p\overline{\nu_t} dW_t^2, \; \nu_0 \quad 0.$$

Since $J_t N_t \quad \mu_J \lambda t$ is a martingale we propose the following measure change for the Bates model

$$dW_t^{1;Q} = dW_t^1 + \frac{r \quad \mu \quad \mu_J \lambda}{\rho \sqrt{\nu_t}} dt$$

$$dW_t^{2;Q} = dW_t^2 + \frac{\mu \quad r + \mu_J \lambda}{\rho \sqrt{\nu_t}} \frac{\rho}{\sqrt{1 \quad \rho^2}} dt$$

that leads to risk-neutral dynamics

$$dS_t = (r \quad \mu_J \lambda) S_t dt + \rho \sqrt{\nu_t} S_t dW_t^{1;Q} + J_t dN_t,$$

$$d\nu_t = \kappa(\theta \quad \nu_t) dt + \xi \rho \sqrt{\nu_t} dW_t^{2;Q}.$$

Again, an idealised variance swap is used as an additional tradable claim. It is important to note that in the presence of jumps the market remains incomplete.

4.1.3 Variance Gamma with CIR stochastic clock

Consider a Levy process $X(t) \quad VG(Ct, G, M)$ that has increments distributed with Variance-Gamma process. We put a stochastic clock $Y_t = \int_0^t y_t dt$ where y_t is a CIR process de ned in the previous models. In the Variance-Gamma model with CIR clock (VG-CIR) the price process is given by

$$S_t = S_0 \exp\Big(X(Y_t) \quad Y_t \psi_X(\ \ i)\Big),$$

where $\psi_X(u)$ is the logarithm of the characteristic function of Variance-Gamma process

$$\psi_X(u) = C \log\left(\frac{GM}{GM + (M \quad G)iu + u^2}\right).$$

Since a Variance-Gamma process can be seen as a di erence of two Gamma processes, for the purpose of simulation we compute $X(t) = \ _t^1 \quad \ _t^2$, where $\ _t^1$ is a Gamma process with mean C and variance $\frac{C}{M}$ and $\ _t^2$ with mean C and variance $\frac{C}{G}$.

4.2 Calibrated parameters

We consider three models of di erent structure that have been calibrated to the same market and quote the results from [16]. These were obtained from the market data available in the year 2003 that consisted of 144 plain vanilla European call options with maturities varying from one month to 5 years.

Table 4.1: Calibrated parameters

Model	Parameters
Heston	ν_0 = 0.0654, κ = 0.6067, θ = 0.0707, ξ = 0.2928, ρ = 0.7571
Bates	ν_0 = 0.0574, κ = 0.4963, θ = 0.0650, ξ = 0.2286, ρ = 0.9900, μ_J = 0.1791, σ_J = 0.1346, λ = 0.1382
VG-CIR	C = 18.0968, G = 20.0276, M = 26.3971, κ = 1.2101, η = 0.5501, λ = 1.7913, y_0 = 1

Table 4.2: Expected value and variance of the price at maturity for each of the models.

	Heston	Bates	VG-CIR
Mean	99.9265	100.0210	99.9172
Std	24.0823	23.8287	23.9734

Examples of sample paths for each of these models can be found in the Appendix A.

4.3 Numerical results: robustness

We choose a time horizon of one year and allow weekly rebalancing., i.e. T = 1, n = 52, t_i = $i/52$ and δ_{t_i} *2* R^2. Furthermore, we specify the mixed expected shortfall risk measure (3.2) for parameters (α, β) = (0.95, 0.5) and choose N = 1000. To simulate Heston and Bates model paths we use Euler-Maruyama with 10 grid points per trading day. The hedged portfolio consist of a short position in a European At the Money Call option.

The distribution of Heston pro t and loss seems to coincide with theoretical expectations and the level of similarity between two hedges also is not surprising either. As jumps - in accordance with calibration parameters - occurred in only 12.09% of Bates model paths, the di usion-based parts of the hedging errors are similar. The minor di erence in tightness is caused by a lower long-term mean variance for the Bates model (Table 5.1). It is also important to stress that, as risk associated with jumps is unhedgable, one can observe severe losses in the left tail of jump-di usion based model. Appropriate quantiles and the di erences in standard deviation are summarised in Table 5.2.

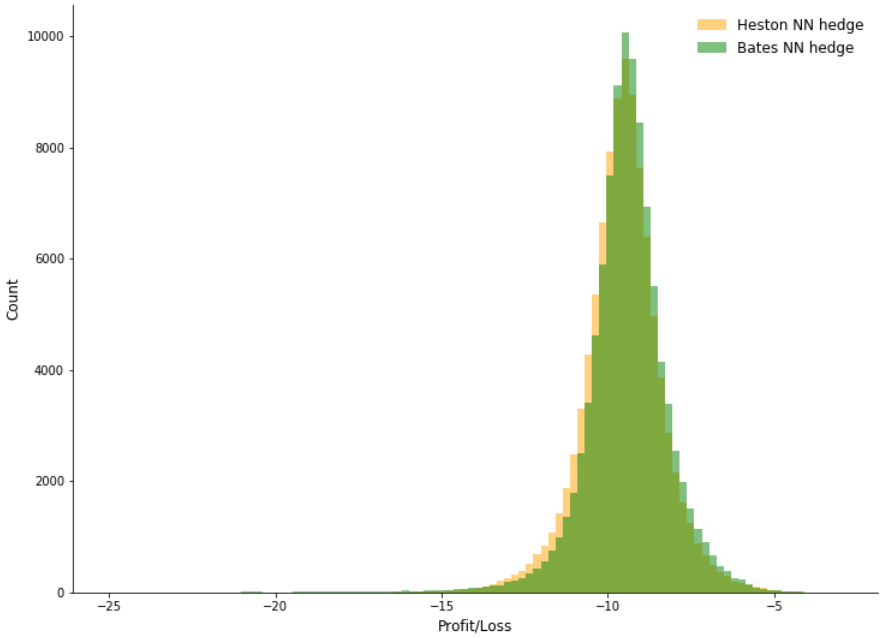

Figure 4.1: Comparison of the hedging errors of the two di erent neural networks that were trained separately for the Heston and Bates models.

Table 4.3: Heston and Bates hedging errors.

Model	Mean	Std	VaR99	VaR95	VaR90	Var80	VaR50
Heston	-9.5706	1.2284	-13.0601	-11.5889	-10.9857	-10.4095	-9.5360
Bates	-9.4673	1.5181	-13.9575	-11.3921	-10.7652	-10.2115	-9.4106

The neural network model captures the behaviour of both of the stochastic volatility models well when tted to both data sets separately. Now we investigate exibility of this approach by training the model on a set that consists of paths for both models. After tting our model to the merged dataset (this model is later referred to as the double NN) we compare the results with those obtained previously (from separate training for each of the underlying dynamics). We nd that the hedging error distribution remains almost unchanged. The only signi cant change (of more than 1% of the initial premium) is observed for the value of VaR_{99}.

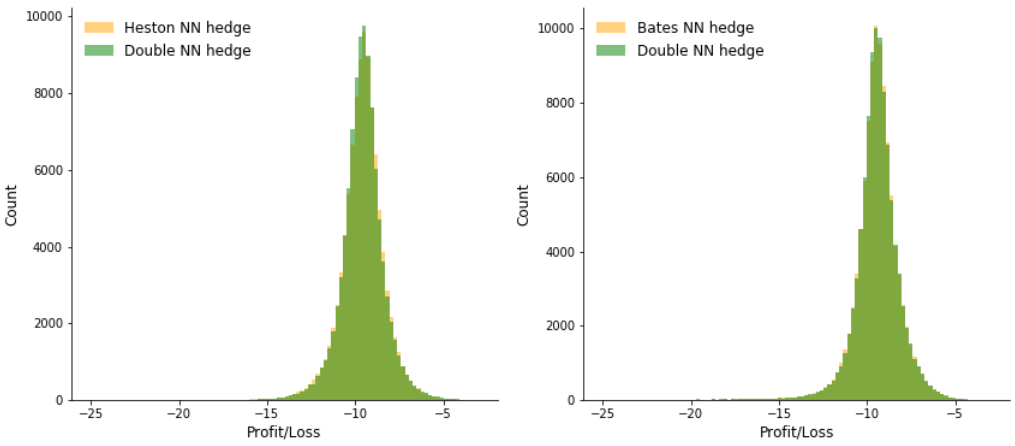

Figure 4.2: Comparison of the double NN hedging errors evaluated separately on Heston and Bates models calibrated to the same market.

Table 4.4: Performance of the NN trained on double sample set. The difference of VaR is expressed as a percent of initial premium.

Model	Differences between the single and double NN:						
	Mean	Std	VaR99	VaR95	VaR90	VaR80	VaR50
Bates	0.0001	-0.0460	1.39%	-0.11%	-0.24%	-0.17%	0.02%
Heston	0.0001	0.0214	-1.11%	-0.77%	-0.50%	-0.30%	0.25%

In order to further verify the robustness of the algorithm we add the third model: Variance Gamma with CIR stochastic clock. Now we fit the network to a training set that consist of sample paths of all aforementioned models. We would like to stress that we have chosen Heston and Bates as models that have a similar structure while the third one exhibits entirely different characteristics. This has a direct impact on the distribution of profit and loss as we expect a much less accurate hedge under the Variance Gamma (VG) which is a pure jump model.

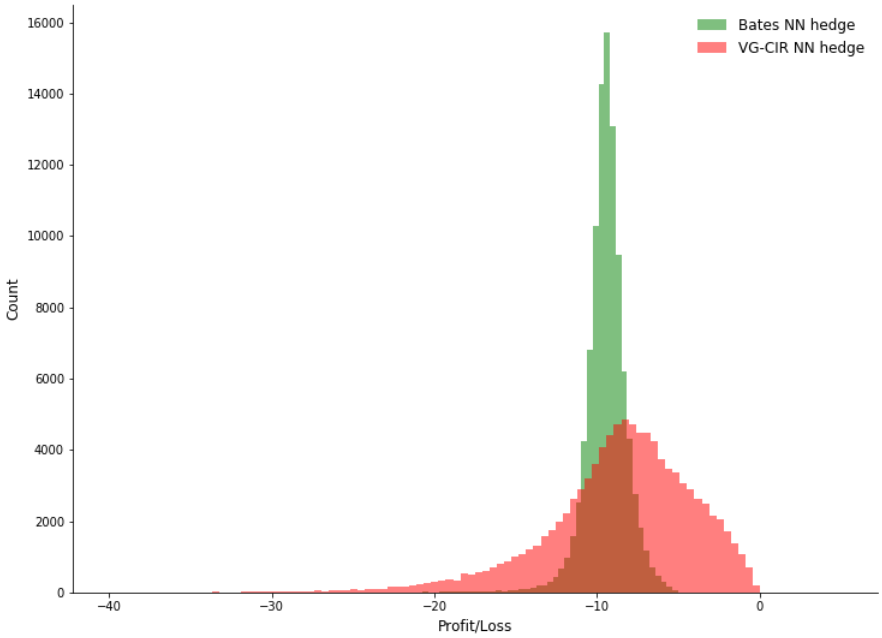

Figure 4.3: Comparison of NN hedging errors between the VG-CIR and Bates model calibrated to the same market.

Again, we train the network on a dataset that consists of paths from all three models (this model is later referred to as the triple NN) and then compare hedges with these obtained by training only for model speci c underlying dynamics. In this case we used 270000 and 30000 paths for the training and validation sets respectively. Separate test sets for each model consist of 100000 paths. We nd that the increased complexity of this task had a signi cant e ect on the hedge quality as can be seen in Figure 4.4. In particular, one can observe a heavier tail and skewness of the triple network hedge applied to the Bates model paths. The change in the realised VaR_{99} was at the level of 4.6740 which equates to 49.40% of the initial premium and indicates a presence of severe extreme losses.

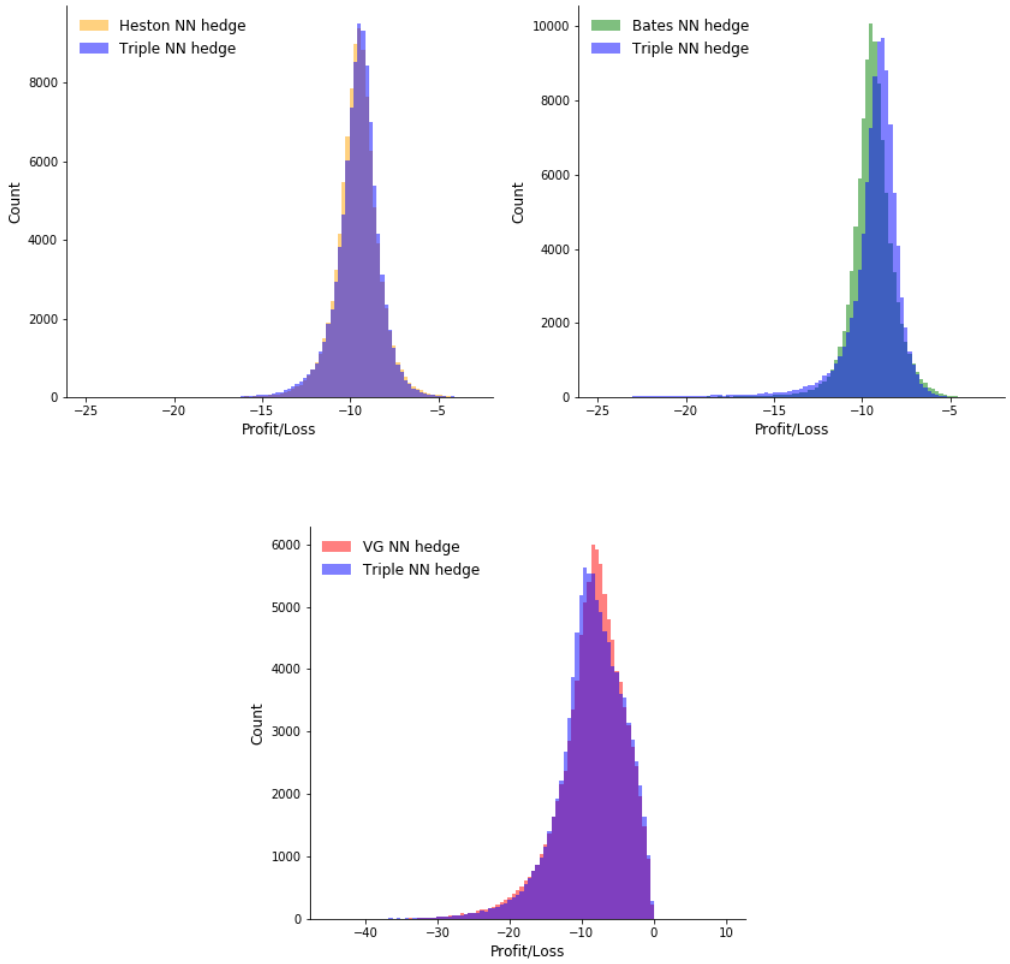

Figure 4.4: Comparison of the triple NN hedging errors evaluated separately on Heston, Bates and VG-CIR models calibrated to the same market.

In naueral network attempt to address these problems, we extend the architecture of the net by adding a fourth hidden layer in front for each component of the recurrent neural network. We choose the same model hyperparameters except for the number of nodes that was rised to 100 in the added layer. The model is tted to the dataset that consisted of path from all three models. The di erences between network hedging errors measured in VaR_{99} are of 10^{-2} order of magnitude. Hence no improvement was made - with very similar parameters for both Heston and VG-CIR hedging error distributions we still encounter di culties with the jump-di usion model.

4.4 Discussion

The series of examples presented above, even though far from a rigorous study, points out that the robustness of our algorithm is somehow limited. Indeed, as the number of di erent

models included in the dataset was increased the overall performance of the algorithm got worsened despite of the attempt to extend the architecture of the net.

Ideally we would expect the neural network not only to learn close-to-optimal hedging parameters for each model, but also to e ectively classify paths on which the net is trained. That could be achieved, for example, by learning quadratic variation of a process or being able to detect a presence of jumps. However, this is challenging in a discretized, market as many of characteristic features are not included in a sampled process path. In order to address these de ciencies, one could try to extend the input set instead. Such augmentation could include a statistic computed from the sample path. As proposed by [1] one could add an estimator of presence of jumps in a discretized path. Another approach would be to consider not only the original dataset, but also its transformations, such as higher order polynomials.

Chapter 5

Markets with frictions

The vast majority of the theoretical results is at least to some extend spoilt when transaction costs are accounted for. For example, the delta-hedging in the Black-Scholes model requires continuous trading which is tremendously expensive in the presence of frictions. In fact, in such a market setting there does not exist a portfolio that perfectly replicates the payo of a claim. Various relaxations were proposed, including the concept of superhedging: instead of attempting to replicate the payo one nds the smallest initial capital that allows to implement a strategy whose terminal wealth majorizes claim's payo . Other approaches, such as variations of the classical PDE-oriented derivations or utility based strategies are also popular.

Trading costs commonly take one of the following forms:

1. Linear transaction costs: $c(n) = \sum_{i=1}^{d} c^i S_t^i |n^i|$,
2. Fixed transaction costs: for $\epsilon > 0$ $c(n) = \sum_{i=1}^{d} c^i \mathbb{1}_{\{|n^i| > 0\}}$,
3. Nonlinear costs associated with market impact.

In the following sections we focus mostly on the linear costs and we attempt to investigate some of the most well known theoretical models that describe hedging with frictions.

5.1 Leland model

The rst in uential paper that discussed trading costs was published by Leland in 1985 (see [15]). Using a formal delta-hedging argument he derived an alternative version of the Black-Scholes equation. In this model, one assumes that costs are of the form of $c(n) = \sum_{i=1}^{d} \kappa S_t^i |n^i|$, trading takes place in intervals of δt, where the both mentioned

quantities are su ciently small, and the ratio $\frac{}{t}$ is of order one. The derived PDE takes the form of

$$\frac{\partial V_t}{\partial t} + rS_t\frac{\partial V_t}{\partial S_t} + \frac{1}{2}\sigma^2 S_t^2\frac{\partial^2 V_t}{\partial S_t^2} \quad \kappa\sigma S_t^2\sqrt{\frac{2}{\pi\delta t}}\left|\frac{\partial^2 V_t}{\partial S_t^2}\right| \quad rV_t = 0$$

and hence one uses a cost-adjusted volatility

$$\hat{\sigma}^2 = \sigma^2\left(1 \quad \frac{2\kappa}{\sigma}\sqrt{\frac{2}{\pi\delta t}}\right)$$

where the sign depends on payo 's convexity. It is crucial to note that the accuracy of this approach relies heavily on the size of κ and δt as the derivation includes approximations.

5.2 The Utility Based Hedging

In modern nance it is common to describe risk preferences by a utility function an agent is trying to maximize. One usually assumes, mostly for the computational ease, that these preferences take a form of

$$U(x) = \quad \exp(\quad \lambda x)$$

where $\lambda > 0$ the absolute risk aversion parameter. It was shown that the option price varies very little with di erent choices of utility functions and that the only crucial factor is the risk aversion parameter [2].

Even though the utility based approach to hedging might seem to be particularly useful, there are certain disadvantages including di culties in implementation and computational ine ciency. Other simpli ed strategies were proposed, one of which is known as the delta tolerance strategy were one constrains the optimal hedging ratio at time t to evolve between two boundaries $^{+}_{t}$ and $_{t}$. As long as the hedge is between these boundaries no rebalancing takes place. However, if the hedge ratio happens to move outside of the no transaction region, then the portfolio is rebalanced. This moves the hedge to the nearest boundary. Whalley and Willmott [17] were the rst to perform an asymptotic analysis under the assumption of small linear transaction costs. With the use of Taylor expansion of the variable κ in powers of $\frac{1}{3}$ they proved that the asymptotically optimal boundaries are of the following form

$$_{t} = \frac{\partial V_t}{\partial S_t} \quad \left(\frac{3}{2}\frac{e^{\quad r(T \quad t)}\kappa S_t \quad ^2_t}{\lambda}\right)^{\frac{1}{3}} \tag{5.1}$$

where

$$_{t} = \frac{\partial^2 V_t}{\partial S_t^2}.$$

As expected, the theoretical Black-Scholes delta-hedging parameter lies in the centre of the no transaction region.

5.3 Theoretical benchmarks

In this section we stay in the Black-Scholes setting from the previous chapter with the exception of an increased rebalancing frequency of 4 trades a day. It follows that T = 30/365, n = 90, t_i = i/1640, and δ_{t_i} 2 R and the hedged portfolio consist of a short position in a European At the Money Call option. We intend to use the no-transaction region based strategy by Whalley and Willmott (WW) adapted to the discretized setting and the Leland hedge as our benchmarks. For the latter the adjusted value of volatility is θ = 0.2128.

Table 5.1: Performance of theoretical hedges adapted to discretized non-frictionless Black-Scholes setting.

Model	Mean	Std	VaR99	VaR95	Ent 0.5	Ent 1
Black-Scholes	-2.6741	0.2450	-3.3766	-3.1160	2.6900	2.7059
Leland	-2.6650	0.2301	-3.2935	-3.0562	2.6784	2.6922
WW (λ = 0.5)	-2.4504	0.5041	-3.3601	-3.2038	2.5128	2.5783
WW (λ = 1)	-2.4600	0.4400	-3.5229	-3.1373	2.5076	2.5551

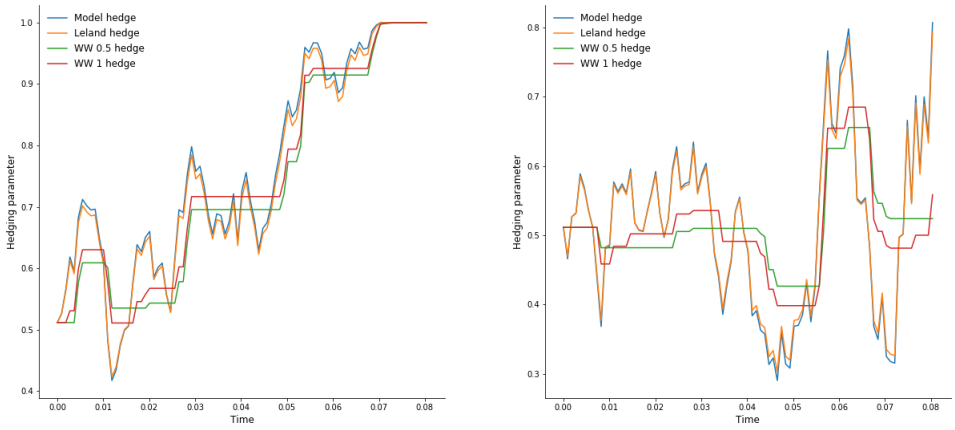

Figure 5.1: No-transaction region hedges for the risk aversion 0.5 and 1 (WW 0.5 and WW 1 respectively) compared to the Black-Scholes and Leland delta-hedging parameters for 2 underlying asset price path.

The above results point to a trade o between mean value and the dispersion of the pro t and loss distribution. Both the Black-Scholes and Leland hedge, as they attempt to replicate the nal payo , result in much lower standard deviation. However, as these strategies are based on extensive trading in the underlying, they lead to signi cantly lower mean loss value. As can be seen in Figure 5.1, the utility based strategies require far less frequent trading. In addition to that, the utility based approach can be seen as notably more risk averse which leads to smaller values of VaR measure.

5.4 Numerical results

We intend to use a neural network to design a hedging strategy that is a mixture of the two approaches described above. We attempt to use the neural network architecture from the previous sections and nd that in the new setting with transaction costs the convergence of the algorithm to any of the theoretical results is not trivial. The entropy based loss function optimisation leads to a hedge that is far more restrictive at the end of the investment horizon i.e. chooses hedging parameters close to the value of 0.5 and, as a result, leads to a very high standard deviation of 0.7435 and 0.6537 for risk-aversion parameters $\lambda = 1$ and $\lambda = 0.5$ respectively. This is similar to the results obtained is Section 3.2 when the expected value with a very high risk-aversion parameter was used as a loss function.

In order to improve on this we would like to design a strategy that is a mixture of the replication and utility-based approach. This requires a greatly extended network architecture (see Figure 5.2).

The training process in now divided into two parts. Firstly, we train our model to capture low level features of a particular market. This means (in the notation used in Figure 5.2) that we evaluate the loss function f_1 for the intermediate layers and then back-propagate to the Input layers. Once the training is done we the freeze gradient optimiser for the layers of the original network (blue background). Next we t a possibly di erent loss function f_2 and train on a modi ed dataset that consists both of the outputs of the intermediate layers as well as of the original input layers. This method of deep learning is a variation of what is known as the transfer learning [4].

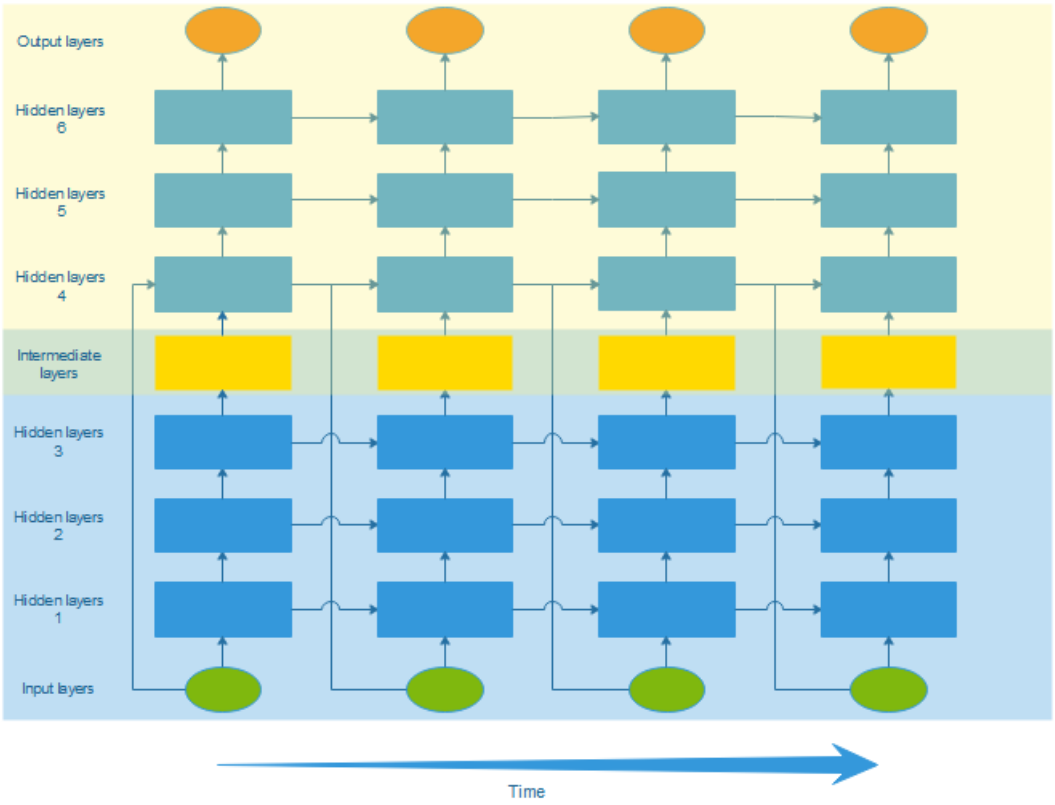

Figure 5.2: The extended network architecture.

In the particular case of hedging in the Black-Scholes setting with proportional transaction costs we specify

$$f_1(X) = l \ ; \ (X)$$

to be the mixed expected shortfall loss used in Section 3. This means, that we want to capture the hedging behaviour for the frictionless market. Next, we specify f_2 as follows

$$f_2(X) = Ent_1(X) + \gamma \sum_{k=0}^{n \ 1} j\delta_{k+1} \quad \delta_k j + \nu \ d(\delta, H^c) \tag{5.2}$$

where $d(x, A)$ denotes the distance between a point x and a set A and Ent_1 is the entropic loss function with the risk-aversion parameter $\lambda = 1$. We additionally penalise aggressive trading but also use the penalty method to make sure that obtained solution lies within a feasible set. One should consider λ and γ to be hyperparameters of the model whereas ν 0 is a Lagrange multiplier. The set H^c is a band of width 0.15 around the close-to-optimal frictionless hedging strategy calculated at the rst training step. The chosen network architecture is $L = 8$, $N_0 = 1$, $N_1 = N_2 = N_5 = N_6 = 50$, $N_3 = N_7 = 30$, $N_4 = N_8 = 1$. The rst 4 layers are used for the initial training and then are frozen for the rest of computations. We also choose $(\lambda, \gamma, \nu) = (1, 10^{\ 3}, 10^8)$.

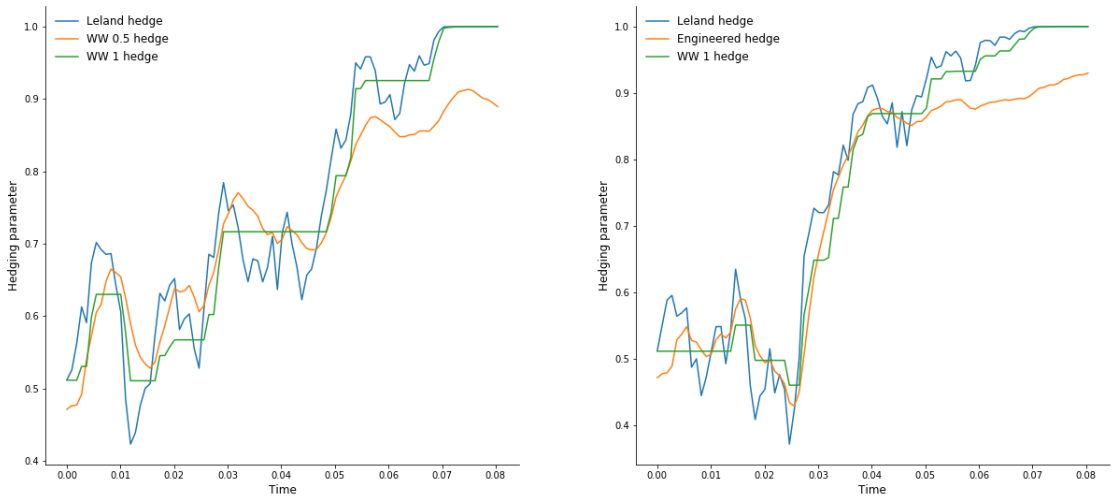

Figure 5.3: Comparison of Leland, Whalley Willmott and our engineered strategy.

The resulting strategy seems to indeed be a mixture of both mentioned approaches, but only up to a certain point is time. Indeed, for over half of the investment horizon the intensity of rebalancing of the engineered strategy is between the other approaches. This is quanti ed by calculating the mean sum of the absolute values of changes in the hedging parameters over the investment horizon. It follows that an average sum of changes in portfolio is at the level of 2.91 for Leland's strategy, 0.82 for Whalley Willmott, and 1.12 for our engineered strategy. Unfortunately, our hedge still su ers (albeit to a smaller extent) from the same characteristic behaviour of a as the pure utility based neural hedge - the position near the end of the investment horizon is too risky in the light of the risk measure. For both presented paths the hedging parameters approach the boundary speci ed by the last component of the loss function (5.2).

With the use of the prepossessing step we managed to capture a low level feature of the market, which in this case is the structure of the theoretical hedge under no frictions, and then use it as the foundation for a new strategy. Even though there is still space for improvement left, this example shows that the network could be further developed used as a nancial engineering tool. We also believe that this approach could be extended greatly with the reinforcement learning techniques. We nd autoencoders to be particularly interesting, as they can learn e cient representations of data. This additional capacity could be seen as a crucial component when there is a log of ambiguity on the setting, as it was shown is Chapter 4.

Chapter 6

Conclusion

6.1 Conclusion

In this dissertation we looked at the state-of-the-art deep learning approach to hedging. We veri ed the performance of an algorithm that was proposed in the literature in the Black-Scholes setting as a benchmark. In addition to a remarkable ability to replicate the discretized delta-hedge, we also discovered that the approach is capable of constructing alternative hedging strategies that are optimal in light of various other criteria such as the expected shortfall with a high risk-aversion parameter. In addition, we found that the results coincide with the theoretical ndings obtained by applying the Black-Scholes formula with a mis-calibrated volatility. This robustness potential was later veri ed by tting the model to a set of paths originating from di erent models that had been rst calibrated to the same market. We discovered that increasing diversity of the training set negatively a ected the performance of the algorithms, especially in the light of a restrictive risk measure such as the value at risk with its parameter at the level above 0.9. Extensions of the network architecture failed to improve on this issue and left potential for further research.

The usefulness of the approach can be seen in a setting with a lot of ambiguity such as non-linear hedging and the pricing problem under the presence of transaction costs. We used the network to engineer a strategy that is a mixture of two types of approaches: Leland's quasi-replicating strategy with adjusted volatility and the utility based no-transaction zone approach. This is achieved by using a variation of the transfer learning method and could be further used in a far more general setting.

6.2 Further research

The rapid development of cutting-edge deep and reinforcement learning techniques make it possible to view a lot of nancial problems from a di erent perspective. With a grow-

ing interest in robust techniques one could further develop the algorithm as a model-free tool. The possible extension includes prepocessing the dataset with the use of autoencoders which could boots the "classi cation" capacity of the algorithm and hence improve on the robustness. In addition to that, techniques originating from time series analysis (such as jump presents estimators) or topological transforms of the dateset could further improve on the current ndings. We nd that the most exciting further work would be to explore the potential of the net as a tool for engineering hedging strategies with desired properties. In addition to that, one could apply neural network techniques in the context of the market impact modelling, or for problems such as liquidation strategies in algorithmic trading. The advantage of this approach might be seen in overcoming the di culties arising from the lack of mathematical tractability of many utility-based models.

Bibliography

[1] Ait-Sahalia, Y. and Jacod, J. (2009): *Testing for Jumps in a Discrete Obsereved Processes*, The Annals of Statistics, Vol. 37, No. 1, pp. 184-222

[2] Andersen, E. D. and Damagaard, A. (1999): *Utility Based Option Pricing with Proportional Transaction Costs and Diversi cation Problems: an Interior-Point Optimization Approach*, Applied Numerical Mathematics, 29, pp. 395-422.

[3] Bates, D. (1996): *Jumps and stochastic volatility: the exchange rate processes implicit in Deutschemark options*, Rev. Fin. Studies, Vol. 9, pp. 69107.

[4] Bengio, Y., Goodfellow I. and Courville A. (2016): *Deep learning*, MIT Press

[5] Black, F. and Scholes, M. (1973): *The pricing of options and corporate liabilities*, Journal of political economy, pp. 637-654

[6] Buehler, H., Gonon, L., Teichmann, J. and Wood, B. (2018): *Deep hedging*, arXiv:1802.03042v1

[7] Davis, M. H. A. and Panas, V. G. and Zariphopoulou, T. (1993): *European Option Pricing with Transaction Costs*, SIAM Journal of Control and Optimization, 31(2), pp. 470-493.

[8] He, K., Zhang, X., Ren, S. and Sun, J. (2015): *Delving Deep into Recti ers: Surpassing Human-Level Performance on ImageNet Classi cation*, arXiv:1502.01852v1

[9] Heston, S. (1993): *A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options*, The Review of Financial Studies. 6 (2): pp. 327-343

[10] Hoggard, T., Whalley, A. E. and Wilmott, P. (1994): *Hedging Option Portfolios in the Presence of Transaction Costs*, Advances in Futures and Options Research, 7, pp. 21-35.

[11] Hornik, K. (1991): *Approximation Capabilities of Multilayer, Feedforward Networks*, Neural Networks, Vol. 4, No. 2, pp. 251-257

[12] El Karoui, N., Jeanblanc-Picque, M. and Shreve, S. E. (1998): *Robustness of the Black and Scholes formula*, Mathematical Finance, 8, pp. 93-126

[13] Kingma, D. P. and J. Ba, (2015): *Adam: a method for stochastic optimisation*, Proceedings of the International Conference on Learning Representations (ICLR)

[14] Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S. (2017): *Self-Normalizing Neural Networks*, arXiv:1706.02515

[15] Leland, H. (1985): *Option Pricing and Replication with Transactions Costs*, The Journal of Finance, Vol. 40, No. 5, pp. 1283-1301

[16] Schoutens, W., Simons, E. and Tistaert, J. (2003): *A Perfect Calibration! Now What?*

[17] Whalley, A. E. and Willmott, P. (2002); *An Asymptotic Analysis of an Optimal Hedging Model for Option Pricing with Transaction Costs*, Mathematical Finance: an international journal of mathematics, statistics and nancial economics, Vol. 7, No. 3, pp. 241-324

Appendix A

Calibrated market paths

Figure A.1: Sample training paths for Heston, Bates, and VG-CIR models calibrated to the same market.

Appendix B

Python code

Below we present the most important parts of our code responsible mostly for the construction and training of the models.

B.1 Packages inclusions and basic functions definitions

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import scipy.stats as ss
import tensorflow as tf

def shuffle_batch(X, batch_size):
    rnd_idx = np.random.permutation(len(X))
    n_batches = len(X) // batch_size
    for batch_idx in np.array_split(rnd_idx, n_batches):
        X_batch = X[batch_idx]
        yield X_batch

def reset_graph():
    tf.reset_default_graph()

def set_seed(seed=42):
    tf.set_random_seed(seed)
    np.random.seed(seed)
```

B.2 Construction and training of the basic RNN

```
reset_graph()

retraining = False
exponential_learning_rate_decay = False

## creating checkpoints
from datetime import datetime
now = datetime.utcnow().strftime("%Y%m%d%H%M%S")
root_logdir = "tf_logs"
logdir = "{}/run {}/".format(root_logdir, now)

## network architecture
```

```
n_steps = 52 # lenth of a time series
n_inputs = 2 # dimensinality of input
n_neurons = [50,50,20,2] # noumber of nodes in layers
activations = [tf.nn.selu,tf.nn.selu,tf.nn.selu,None] # activation fucntions
n_layers = 4

# setting up performance scheduling
max_checks_without_progress = 10
checks_without_progress = 0
if not retraining:
    best_loss = np.infty


##  decay_steps=4000    ### used in case of the exponencial decay
##  decay_rate=1/3

# training parameters
initial_learning_rate = 0.00005
n_epochs = 400
global_step = tf.Variable(0,trainable=False,name="global_step")
batch_size= 200

# defining placegolders
X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
training = tf.placeholder_with_default(False, shape=(), name='training')

# construction of the network
with tf.variable_scope('rnn', initializer=tf.variance_scaling_initializer()):

    # a list of BasicRNNcells
    layers = [tf.contrib.rnn.BasicRNNCell(num_units=n_neurons[layer],
                                          activation=activations[layer])
                                          for layer in range(n_layers)]

    #dynamically unrolled approach
    multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
    outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)

if exponential_learning_rate_decay:
    learning_rate=tf.train.exponential_decay(initial_learning_rate,global_step
    ,decay_steps,decay_rate)
else:
    learning_rate = initial_learning_rate

loss = tf.entropic_loss(100.0,0.5) #loss function: etropy in this case

#defining the optimising algorithm
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss,global_step=global_step)

# defining variables initializer and a saver (to manage graph easily)
init = tf.global_variables_initializer()
saver = tf.train.Saver()

print("Up and running!") # Safety check

# Training
with tf.Session() as sess:

```

```
    # restoring graph if needed. If not we initialize variables.
    if retraining:
        saver.restore(sess, "./my_dissertation_model.ckpt")
    else:
        init.run()


    for epoch in range(n_epochs):
        for X_batch in shuffle_batch(X_train, batch_size):
            sess.run([training_op,outputs], feed_dict={X: X_batch})

        loss_train = sess.run(loss,feed_dict={X:X_train})
        loss_val = sess.run(loss,feed_dict={X:X_validation})

        # performance scheduling
        if not exponential_learning_rate_decay:
            if loss_val < best_loss:
                save_path = saver.save(sess, "./my_dissertation_model.ckpt")
                best_loss = loss_val
                checks_without_progress = 0
            else:
                checks_without_progress += 1
                if checks_without_progress > max_checks_without_progress:
                    print("Early stopping!")
                    break

        if exponential_learning_rate_decay:
            print("{}\tCurrent train loss: {:.6f}\tCurrent val loss: {:.6f}\tBest loss: {:.6f}\tLearning rate: {:.6f}".format(
            epoch, loss_train,loss_val, best_loss, learning_rate.eval()))
        else:
            print("{}\tCurrent train loss: {:.6f}\tCurrent val loss: {:.6f}\tBest loss: {:.6f}\tLearning rate: {:.10f}".format(
            epoch, loss_train,loss_val, best_loss, learning_rate))


# saving final results
with tf.Session() as sess:
    saver.restore(sess, "./my_dissertation_model.ckpt")
    print("Final loss: {:.6f}".format(best_loss))
    outputs_validation = sess.run(outputs,feed_dict={X:X_validation})
    outputs_train = sess.run(outputs,feed_dict={X:X_train})
    outputs_test = sess.run(outputs,feed_dict={X:X_test})
    loss_validation = sess.run(loss,feed_dict={X:X_validation})
    loss_train = sess.run(loss,feed_dict={X:X_train})
    loss_test = sess.run(loss,feed_dict={X:X_test})
```

B.3 Crucial parts of the extended RNN constructing code (after pretraining)

```
[...]

## network architecture
n_steps = 52 # lenth of a time series
n_inputs = 2 # dimensinality of input
n_neurons = [50,50,20,2,50,50,20,1] # noumber of nodes in layers
activations = [tf.nn.selu,tf.nn.selu,tf.nn.selu,None,tf.nn.selu,tf.nn.selu,tf.nn.selu,None] # activation fucntions
```

```
n_layers_frozen=4
n_layers = 8


[...]


# construction of the extended network
with tf.variable_scope('rnn', initializer=tf.variance_scaling_initializer()):

    # a list of BasicRNNcells
    layers = [tf.contrib.rnn.BasicRNNCell(num_units=n_neurons[layer],
                                          activation=activations[layer])
                                          for layer in range(n_layers)]

    #dynamically unrolled approach
    multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
    outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)

[...]
# specyfing trainable layers and passing them to the optimiser
training_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, scope="rnn/
    rnn/multi_rnn_cell/cell_[4567]")
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss, global_step=global_step, var_list=
    training_vars)

# specyfing frozen layers and defining an additional saver to restore them
reuse_vars = tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES, scope="rnn/rnn/
    multi_rnn_cell/cell_[0123]")
restore_saver = tf.train.Saver(var_list=reuse_vars)

# defining variables initializer and a saver (to manage graph easily)
init = tf.global_variables_initializer()
saver = tf.train.Saver()

print("Up and running!") # Safety check

# Training
with tf.Session() as sess:

    # restoring graph if needed. If not we initialize variables.
    if retraining:
        saver.restore(sess, "./my_dissertation_model.ckpt")
    else:
        init.run()
        restore_saver.restore(sess, "./my_pretrained_model.ckpt")

[...]
```