Ludwig-Maximilians-Universität München

Fakultät für Mathematik, Informatik und Statistik

Deep Hedging in varying market models

Masters's Thesis

Henry Constantin Kleineidam

Supervisor: Prof. Dr. Lukas Gonon
Date: 01.08.2022

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Name: Henry Constantin Kleineidam
Date: 01.08.2022

Signature:

Contents

1 Introduction

Financial derivatives are instruments that are based on some underlying financial assets. Institutions such as banks that buy or sell such derivatives wish to price them as accurately as possible and minimize or even eliminate the risk associated with them. This is achieved through hedging. A hedge is an asset position designed to offset the risk of the agents position in a derivative. The amount of capital needed to buy into this hedging position gives the price of the derivative. The Black-Scholes (BS) framework [1] is a mathematical framework to model asset prices and allows for closed form solutions to pricing and hedging under some assumptions, for example the constant volatility and absence of trading restrictions. Applying this to real world scenarios however is very difficult as the complexity of real markets expose the flaws of the central assumptions of the BS framework and as the distance between the model and the real world grows larger the utility of that model grows smaller.

While newer and better models have been developed that address one or many deficiencies of the BS model, such as [2], [3] or [4], these models, as we will see later on, are all very different from one another and are all only as good as the calibration of their parameters. While there are some countermeasures to poorly calibrated parameters or even simplifying assumptions, choosing the wrong model in the first place can be difficult to counteract and very costly.
Assuming now we have chosen the right model with the right parameters, closed form solutions to pricing and hedging can still be very elusive in the more advanced mathematical models especially, if we factor in real world restraints such as trading costs, market impact and liquidity restrictions.
The best way to avoid incorrect assumptions is to make no assumptions at all or at least as few as possible. This brings us the deep hedging approach, using deep neural networks to make hedging and ultimately pricing decisions. Feature sets in this approach may contain prices of hedging instruments, current positions in said hedging instruments, information about previous prices or any other quantifiable information that might be useful. Assumptions about market dynamics only play a role for the input we chose for our network, the network itself both in the training/optimization and in the deployment phase is entirely model free. All we need to do is generate data for our hedging instruments, define an objective function or a loss function, as it is called in machine learning terminology, that we wish to optimize and any restrictions we would like to place on the trading and then we can use modern machine learning methods to obtain a hedging and pricing engine.
The approach outlined in this thesis is largely based on [5] who have introduced this approach and used it to hedge in the Heston model and [6] who have improved the approach

for non-Markovian market models and then applied it to the rough Bergomi (rBergomi) model. The purpose of this thesis is to provide additional proof that the deep hedging approach is both model independent and robust. To this end we will create a mixed market consisting of samples from three different market models, the Heston model, the Merton Jump Diffusion model and the rBergomi model and then try to hedge in this market both with and without trading costs.

1.1 Outline

The rest of this thesis is structured as follows. In chapter 2 and 3 we give the general framework for our hedging and pricing task and a brief introduction to convex risk measures. Chapter 4 outlines how optimal hedging strategies can be approximated by neural networks. Chapter 5 introduces the different market models and how they are calibrated and finally in chapter 6 we will use the deep hedging approach to hedge in all given market models using the same networks and presenting the numerical results.
All numerical experiments will be done in Python code relying on open-source libraries, such as TensorFlow for the machine learning tasks, NumPy for generation of input data, QuantLib and SciPy for calibration and seaborn and matplotlib for the plotting of results. The full list with the respective versions of each library can be found in the appendix.

2 Discrete time market with frictions

2.1 Setting

We consider a discrete-time financial market with frictions, a finite time horizon T and trading days $0 = t_0 < t_1 < \cdots < t_n = T$. We fix a finite probability space $\Omega = \{\omega_1, \ldots, \omega_N\}$ with a probability measure $\mathbb{P}[\{\omega_i\}] > 0$ for all $i = 1, \ldots, N$ and $\mathcal{X} := \{X : \Omega \to \mathbb{R}\}$ is the set of all random real-valued random variables over Ω. The filtration $\mathbb{F} = (\mathcal{F}_k)_{k=0,..,n}$ is generated by an $\mathbb{R}^m$-valued process $I = (I_k)_{k=0,..,n}$, with $\mathcal{F}_k = \sigma(I_0, ..., I_k)$ representing all available market information up to t_k. In our case this information will be entirely related to the tradeable assets and the trading history, i.e. the previous trades. We have d tradeable assets, the prices of which are given by a $\mathbb{R}^d$-valued and $\mathbb{F}$-adapted stochastic process $S = (S_k)_{k=0,...,n}$. These assets will consist of the underlying assets of the derivative that is to be hedged as well as other derivatives with observable market prices related to the underlying, but in principle could be any assets which seem to be a good idea to trade in. Finally we have our $\mathcal{F}_T$ measurable payoff Z that will represent the liability to hedge against. We assume Z to be the payoff of a European option and occur at T, the type of option will not be disclosed to the neural network, the payoff will simply be added to its calculations as input at the appropriate time. We also do not make the assumption that $\mathbb{P}$ is a martingale measure or that any of our assets are martingales.

For simplicity we will assume that all payments are accrued with a risk-free overnight rate, which effectively means that we assume rates of 0 and that all payments occur at maturity.

2.2 Hedging

The task at hand is the minimization of the risk associated with the payoff Z and the calculation of an appropriate price for the derivative. Trading in all available assets the hedging strategy becomes an $\mathbb{R}^d$-valued $\mathcal{F}$-adapted process $\delta = (\delta_k)_{k=0,...,n-1}$ with $\delta_k = (\delta_k^1, \ldots, \delta_k^d)$. The notation reads as follows: We hold δ_k^i shares of the i-th asset at time t_k. Additionally we set $\delta_{-1} = \delta_n = 0$, i.e. we do not hold any assets at the start and we liquidize our position at maturity. Furthermore we assume no cash injections during trading, i.e. the trading strategy needs to be self-financing. Because Ω is finite $\delta_k : \Omega \to \mathbb{R}^d$ is bounded, we define the set of all such trading strategies as $\mathcal{H}$.

We will now take a look at the quantities that determine our hedging success. First up is the trading success:

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

The total tradings costs are given by:

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

where $c_k : \mathbb{R}^d \to \mathbb{R}^+$ for each k is the cost function for each period, for example:

1. *Fixed transaction costs* : $c_k(x) := \sum_{i=1}^{d} c_k^i 1_{|x|>0}$,
2. *Proportional transactions costs* : $c_k(x) := \sum_{i=1}^{d} c_k^i S_k^i |x|$,

with $c_k^i > 0$ being the cost parameter. In both cases c_k is upper semi-continuous and $c_k(0) = 0$, with fixed transaction costs punishing frequent rebalancing and proportional transaction costs punishing high trading volumes.
Because we have a self-financing trading strategy we might need to inject extra cash to cover our liability, in practice this would be the value we sell the option for at the start of our task and is denoted by p_0. Our profit and loss at maturity is now given by:

$$PnL(Z, p_0, \delta) := -Z + p_0 + (\delta \cdot S)_T - C_T(\delta) \tag{2.1}$$

3 Convex risk measures

In a complete market with continuous-time trading, no transaction costs and unconstrained trading, there exists for any liability Z a unique replication strategy δ and a fair price p_0, such that $-Z + p_0 + (\delta \cdot S)_T - C_T(\delta) = 0$ holds almost surely. This is not the case in our setting, since we trade in discrete time with transaction costs. This gives us reason to specify a new way to quantify our hedging success and calculate the price we wish to charge, for us to accept the liability Z. Since we can no longer avoid taking a loss some of the time, we now redefine the price to be the minimal amount of cash that needs to be added to our position in order to make the overall risk acceptable to our perception of risk and consider a position to be acceptable if $\rho(Z) \leq 0$ for a given risk measure $\rho : \mathcal{X} \to \mathbb{R}$.
Here we focus on the key properties and definitions of convex risk measures, however much additional detail is discussed in [7]. Let $X, Y \in \mathcal{X}$, we call $\rho : \mathcal{X} \to \mathbb{R}$ a convex risk measure if it is:

1. Monotone decreasing: $X \geq Y$ then $\rho(X) \leq \rho(Y)$. A better position requires less cash injection.

2. Convex: $\rho(\alpha X + (1-\alpha)Y) \leq \alpha\rho(X) + (1-\alpha)\rho(Y)$ for $\alpha \in [0,1]$. Diversification reduces risk.

3. Cash Invariant: $\rho(X+c) = \rho(X) - c$ for any $c \in \mathbb{R}$. Cash reduces risk by just that much.

Note that because of the third property: $\rho(X - \rho(X)) = 0$, i.e. $\rho(X)$ is the least amount of cash needed to make the position X acceptable. If $\rho(0) = 0$, we call ρ normalized. With $X \in \mathcal{X}$ and $\rho : \mathcal{X} \to \mathbb{R}$ a convex risk measure we get the optimization problem:

$$\pi(X) := \inf_{\delta \in \mathcal{H}} \rho(X + (\delta \cdot S)_T - C_T(\delta)). \tag{3.1}$$

Proposition 1. *π is monotone decreasing and cash-invariant. If $C_T(\delta)$ and $\mathcal{H}$ are convex, then π is a convex risk measure.*

Proof. See Proposition 3.2. in [5] □

We can now define the optimal $\delta \in \mathcal{H}$ as a minimizer of (3.1) and $\pi(X)$ to be the minimal amount of cash that needs to be added to a position X to make it acceptable with respect to the risk measure ρ. Going back to our liability Z we define the indifference price as the solution $p(Z)$ to $\pi(-Z + p(Z)) = \pi(0)$. By cash invariance we get:

$$p(Z) := \pi(-Z) - \pi(0) \tag{3.2}$$

This price $p(Z)$ is the same as the price of the replicating portfilio p_0 in the absence of trading costs and other restrictions.

Lemma 1. *If $C_T \equiv 0$ and Z is attainable, i.e. $\exists \delta^* \in \mathcal{H}$ and $p_0 \in \mathbb{R}$ s.t. $Z = p_0 + (\delta^* \cdot S)_T$ a.s., then $p(Z) = p_0$.*

Proof. For any strategy $\delta \in \mathcal{H}$ cash-invariance of ρ implies

$$\rho(-Z + (\delta \cdot S)_T - C_T(\delta)) = p_0 + \rho(([\delta - \delta^*] \cdot S)_T).$$

Taking the infimum over all strategies on both sides and using the fact that $\mathcal{H} - \delta^* = \mathcal{H}$ we get

$$\pi(-Z) = p_0 + \inf_{\delta \in \mathcal{H}} \rho(([\delta - \delta^*] \cdot S)_T) = p_0 + \pi(0)$$

Plugging this back into (3.2) concludes the proof. □

3.1 Optimized certainty equivalents (OCEs)

The OCE is defined for a given utility function $u : \mathbb{R} \to \mathbb{R}$, i.e. continuous, non-decresing and concave as $OCE(X) = \sup_{\omega \in \mathbb{R}} \{\omega + \mathbb{E}[u(X - \omega)]\}$ and represents the optimal allocation between immediate cash ω and expected utility in the future $\mathbb{E}[u(X - \omega)]$. OCEs and risk measures share a relationship by setting $\rho(X) := -OCE(X)$ and turning the utility function into a loss function $\ell(x) := -u(-x)$.

Lemma 2. *Let $\ell : \mathbb{R} \to \mathbb{R}$ be a loss function, i.e. continuous, non-decreasing and convex. Then*

$$\rho(X) := \inf_{\omega \in \mathbb{R}} \{\omega + \mathbb{E}(\ell(-X - \omega))\}, X \in \mathcal{X} \tag{3.3}$$

is a convex risk measure.

Proof. Let X,Y $\in \mathcal{X}$ be assets.

1. Monotonicity: Suppose $X < Y$. Because ℓ is non-decreasing for any $\omega \in \mathbb{R}$ we have $\mathbb{E}[\ell(-X - \omega)] \geq \mathbb{E}[\ell(-Y - \omega)]$ $\forall \omega \in \mathbb{R}$ and therefore $\rho(X) > \rho(Y)$.

2. Cash-invariance: $\rho(X + m) = \inf_{\omega \in \mathbb{R}} \{(\omega + m) - m + \mathbb{E}[\ell(-X - (\omega + m))]\} = \inf_{\omega \in \mathbb{R}} \{\omega + \mathbb{E}[\ell(-X - \omega)]\} - m = \rho(X) - m$ $\forall m \in \mathbb{R}$.

3. let $\lambda \in [0, 1]$ and set $\gamma = 1 - \lambda$ and we get:

$$\begin{aligned}
\rho(\lambda X + \gamma Y) &= \inf_{\omega \in \mathbb{R}} \{\omega + \mathbb{E}[\ell(-\lambda X - \gamma Y - \omega)]\} \\
&= \inf_{\omega_1 \in \mathbb{R}, \omega_2 \in \mathbb{R}} \{\lambda \omega_1 + \gamma \omega_2 + \mathbb{E}[\ell(\lambda(-X - \omega_1) + \gamma(-Y - \omega_2))]\} \\
&\leq \inf_{\omega_1 \in \mathbb{R}} \inf_{\omega_2 \in \mathbb{R}} \{\lambda(\omega_1 + \mathbb{E}[\ell(-X - \omega_1)]) + \gamma(\omega_2 + \mathbb{E}[\ell(-Y - \omega_2)])\} \\
&= \lambda \rho(X) + \gamma \rho(Y)
\end{aligned}$$

   Where the second to last step holds because ℓ is convex.

□

We will now introduce the convex risk measure we will be using for some of our numerical experiments, namely the entropic risk measure defined by

$$\rho(X) = \frac{1}{\lambda} \log \mathbb{E}[\exp(-\lambda X)],\ \lambda > 0 \tag{3.4}$$

We will now show that this is indeed a convex risk measure, because it is induced by plugging the loss function $\ell(x) := \exp(\lambda x) - \frac{1+\log(\lambda)}{\lambda}$ into the OCE (3.3) and then using the above Lemma.

Proof. Set $\lambda > 0$ and $X \in \mathcal{X}$, starting from (3.3) and plugging in ℓ we wish to minimize $\omega + \mathbb{E}[\exp(\lambda(-X-\omega))] - \frac{1+\log(\lambda)}{\lambda}$ over $\omega \in \mathbb{R}$. We do this by differentiation, where we can switch operations with the expectation, because Ω is finite and all functions are continuous and therefore any expected values are finite .

$$\begin{aligned} 0 &\overset{!}{=} 1 + \mathbb{E}[-\lambda \exp(\lambda(-X-\omega))] \\ \Leftrightarrow\ & \exp(\lambda\omega) = \lambda \mathbb{E} \exp(-\lambda X) \\ \Leftrightarrow\ & \omega = \frac{1}{\lambda} \log(\lambda \mathbb{E}[\exp(-\lambda X)]) \end{aligned}$$

Which is the global minimum, because of convexity. Now we insert this back into (3.3).

$$\begin{aligned} \rho(X) &= \inf_{\omega \in \mathbb{R}} \{\omega + \mathbb{E}[\exp(\lambda(-X-\omega))] - \frac{1+\log(\lambda)}{\lambda}\} \\ &= \frac{1}{\lambda} \log \lambda \mathbb{E}[\exp(-\lambda X)] + \mathbb{E}[\exp(-\lambda X - \log(\lambda \mathbb{E}[\exp(-\lambda X)]))] - \frac{1+\log(\lambda)}{\lambda} \\ &= \frac{1}{\lambda} \log(\mathbb{E}[\exp(-\lambda X)]) + \frac{\log(\lambda)}{\lambda} + \frac{\mathbb{E}[\exp(-\lambda X)]}{\mathbb{E}[\exp(\log(\lambda \mathbb{E}[\exp(-\lambda X)]))]} - \frac{1+\log(\lambda)}{\lambda} \\ &= \frac{1}{\lambda} \log(\mathbb{E}[\exp(-\lambda X)]) + \frac{\mathbb{E}[\exp(-\lambda X)]}{\lambda \mathbb{E}[\exp(-\lambda X)]} - \frac{1}{\lambda} \\ &= \frac{1}{\lambda} \log(\mathbb{E}[\exp(-\lambda X)]) \end{aligned}$$

□

Proposition 2. *Suppose S is a $\mathbb{P}$-martingale, ρ is defined as in (3.3) and π, p are defined as in (3.1),(3.2). Then*

1. *$\pi(0) = \rho(0)$*
2. *$p(Z) \geq \mathbb{E}[Z]$ for any $Z \in \mathcal{X}$*

Proof. See Proposition 3.10 in [5] □

While we will be dealing with scenarios where the second point holds with equality, at least when there are no transaction costs, the increase in risk adjusted price is demonstrated quite nicely in [5] for the conditional value at risk in the Heston model.

4 Neural Networks

In this sections we introduce fully connected (dense) feedforward neural networks, discuss some of their properties and show that using them to approximate hedging strategies is theoretically well-founded.

Let $L, N_0, N_1, \ldots, N_L \in \mathbb{N}$, $\sigma : \mathbb{R} \to \mathbb{R}$ and for any $l = 1, \ldots, L$, let $W_l : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$ an affine function. A neural network is a function $F : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ defined as:

$$F(x) = W_L \circ F_{L-1} \circ \cdots \circ F_1 \text{ with } F_l = \sigma \circ W_l \text{ for } l = 1, \ldots, L-1$$

We chose $W_l(x) = A^l x + b^l$ with $A^l \in \mathbb{R}^{N_l \times N_{l-1}}$ and $b^l \in \mathbb{R}^{N_l}$ and define $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ the set of all such networks mapping $\mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ with activation function σ. The entries of the matrices A^l are called weights and the entries of the vectors b^l are called biases. The activation function is applied component-wise and serves to introduce nonlinearity. Each F_l is called a layer with F_1 being called the input layer, W_L being called the output layer and the layers in between being called hidden layers. Note that sometimes the final output of W_L is being put through another activation function that can be different from σ. The effectiveness of such networks will be shown in the theorem below, additional information can be found in [8].

Theorem 1. (Universal Approximation Theorem) *Let σ be bounded and non-constant, then we have:*

1. *For any finite measure μ on $(\mathbb{R}^{d_0}, \mathcal{B}(\mathbb{R}^{d_0}))$ and $1 \leq p < \infty$ then $\mathcal{NN}^{\sigma}_{\infty,d_0,1}$ is dense in $L^p(\mathbb{R}^{d_0}, \mu)$*
2. *If in addition $\sigma \in C(\mathbb{R})$, then $\mathcal{NN}^{\sigma}_{\infty,d_0,1}$ is dense in $C(\mathbb{R}^{d_0})$ for the topology of uniform convergence on compact sets.*

Proof. See [8] □

Because every component of an $\mathbb{R}^{d_1}$-valued neural network is an $\mathbb{R}$-valued neural network this result easily generalizes to $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$, see [8].
We denote by $\{\mathcal{NN}^{\sigma}_{M,d_0,d_1}\}_{M\in\mathbb{N}}$ a sequence subsets of $\mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$ with the following properties:

- $\mathcal{NN}^{\sigma}_{M,d_0,d_1} \subset \mathcal{NN}^{\sigma}_{M+1,d_0,d_1} \forall M \in \mathbb{N}$
- $\bigcup_{M\in\mathbb{N}} \mathcal{NN}^{\sigma}_{M,d_0,d_1} = \mathcal{NN}^{\sigma}_{\infty,d_0,d_1}$
- for any $M \in \mathbb{N}$, one has $\mathcal{NN}^{\sigma}_{M,d_0,d_1} = \{F^{\theta} : \theta \in \Theta_{M,d_0,d_1}\}$ with $\Theta_{M,d_0,d_1} \in \mathbb{R}^q$ for some $q \in \mathbb{N}$ depending on M.

The sequence above consists of sets of neural networks with an architecture parameterised by a parameter vector θ whose dimensions depend on M, this would include the number of layers, the size of those layers and the weights and biases of these layers.
On the basis of the Universal Approximation Theorem we know that a neural network can solve our optimization problem (3.1), however finding such a network when both the architecture and the parameters are unknown is a task unlikely to be successful.
We now rewrite our optimization problem (3.1) into a form that can be solved by neural networks. To this end we need to put constraints on the space of trading strategies so that it can be represented by neural networks with a fixed architecture.

$$\begin{aligned}\mathcal{H}_M := \{(\delta^{\theta}_k)_{k=0,\dots,n-1} \in \mathcal{H} : \delta^{\theta}_k = F^{\theta_k}(I_0,\dots,I_k,\delta^{\theta}_{k-1}),\\ \theta_k \in \Theta_{M,r(k+1)+d,d})\}\end{aligned} \tag{4.1}$$

$$\begin{aligned}\pi^M(X) :&= \inf_{\delta\in\mathcal{H}_M} \rho(X + (\delta\cdot S)_T - C_T(\delta))\\ &= \inf_{\theta\in\Theta_M} \rho(X + (\delta_\theta\cdot S)_T - C_T(\delta_\theta))\end{aligned} \tag{4.2}$$

With $\Theta_M := \prod_{k=0}^{n-1} \Theta_{M,r(k+1)+d,d}$.
We will discuss the structure of the network chosen for our numerical experiments in greater detail later on, however it is worth noting now that the strategies at each time-step are given as the output of individual layers are only dependent on the information available up until that time-step. Of course hedging when you already know the trajectory of your asset would be a trivial task. The new optimization problem (4.2) is reduced to the finite-dimensional task of finding the right parameters for our network.
What remains to be seen is whether a solution for (4.2) is also a solution for (3.1) or at least close to it. The next proposition deals with this issue, showing that the solution to the optimization problem (4.2) approximates the solution to (3.1) arbitrarily well.

Proposition 3. *For any $X \in \mathcal{X}$ we have:*

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

Proof. See [5] Proposition 4.9. □

With Theorem 1 and Proposition 3 we now have a theoretical foundation for the usage of neural networks to approximate hedging strategies and prices. We now turn our eye towards the task of finding the optimal parameter $\theta \in \Theta_M$. In this thesis we focus on the entropic risk measure, a more general approach can be found in [5]. Inserting (3.4) into (4.2) gives us:

$$\pi(-Z) = \frac{1}{\lambda} \log \inf_{\theta \in \Theta_M} J(\theta),$$

where

$$J(\theta) := \mathbb{E}[\exp(-\lambda[-Z + (\delta^\theta \cdot S)_T - C_T(\delta^\theta)])]. \tag{4.3}$$

The approach to finding a local minima of such a function J is called gradient descent. Starting with an initial guess $\theta^{(0)}$ for the parameter over which we wish to minimize , we define iteratively

$$\theta^{(j+1)} = \theta^{(j)} - \eta_j \nabla J_j(\theta^{(j)}) \tag{4.4}$$

for some small $\eta_j > 0,\ j \in \mathbb{N}$ and $J_j = J$. There are two problems with this approach, firstly we might end up only finding a local minimum and secondly we need to calculate ∇J in every step, with a potentially massive number of steps this can become computationally expensive. Both of these issues can be mitigated by the use of stochastic gradient descent and backpropagation. Backpropagation describes the process of first tracking all computations and then calculating the gradient by iteratively calculating the gradients of each individual computation and then putting them together via the chain rule, see [9] chapter 7.3.1. This technique is build into tensorflow, that tracks all computations done in its framework and is then able to calculate gradients very efficiently. As for the stochastic gradient descent we now replace the expectation in (4.3) with the weighted sum over a chosen sample of size N_{batch}, so that J_j is now given by

$$J_j(\theta) = \sum_{m=1}^{N_{batch}} \exp[-\lambda(-Z(\omega_m^{(j)}) + (\delta^\theta \cdot S(\omega_m^{(j)}))_T - C_T(\delta^\theta)(\omega_m^{(j)}))] \frac{1}{N_{batch}}$$

with the sample $\omega_1^{(j)}, \ldots, \omega_{N_{batch}}^{j} \in \Omega$. In our case we assume equal probability for all samples, which is reflected by the $\frac{1}{N_{batch}}$ term. This doesn't guarantee that we won't get stuck in a local minimum, however due to the randomness of the process, we might be able to escape even after reaching a local minimum in some cases.

We are now ready to discuss the structure of the neural network that we have chosen for our hedging tasks. Our approach will be an adaptation of the network structure proposed in [6], which in turn is an adaptation of [5]. Using a semi-recurrent structure we have an individual network for each period, that takes as input the output of the network of the previous period, which is the trading strategy as well as any other information available up to that point. Since the Markov property is not satisfied for the rBergomi model, we cannot just use the new information in each period as input and instead must find a sensible way to track all the information available up to that point. This is done by adding an additional output for each tradeable asset in each period, that

is used purely as input for the next period.
We are also using a more advanced version of the stochastic gradient descent, the "Adam" algorithm, see [10]. This algorithm adapts the learning rate and therefore the size of the parameter updates by calculating momentum from estimated first and second moments of the gradients. This can help avoid "zig-zagging" of paramter updates and being stuck in local minima.
Another key feature is the so called batch normalization, see [11], which is a very popular and well known technique to facilitate and accelerate the training of networks by addressing the covariate shift. Covariate shift is the phenomenon that because in a fully connected network each layer's input is the output of the previous layer, the training process, i.e. learning the correct weights and biases for our network, gets slowed down for later layers as the input distribution changes with the changed parameters of the previous layers. Therefore normalizing the batches between layers allows us to train all layers at the same time, effectively reducing the dependence on previous layers.

The techniques above help us chosing the correct weights and biases for our layers, but those aren't the only parameters in our network of networks. In fact the usage of the techniques could even be considered to be a parameter of our network. This is what is sometimes refered to as outer optimization of hyper parameters. Which optimization algorithm do we use, which activation functions do we use, do we use batch normalization and how many layers of which size should we chose? We can't use gradient based algorithms, because there are no gradients with respect to any of these parameters and without any previous knowledge the best thing we can do is to try out what works and what doesn't. A full grid-like search over hyperparamters however is also computationally unfeasible as training our network takes a lot of time before any sort of perfomance can be measured. A proper search for hyper parameters is left for future work, the parametrs chosen in this thesis are presented in the numerical experiments chapter.

5 Market Models

We will now take a look at the different market models for our hedging. The Black Scholes model will serve as an introduction and a general proof of the concept of deep hedging. The other three models, the rBergomi, the Heston and the Merton Jump Diffusion model will first be fed to our neural network individually as market data input and then all three models will be mixed together as market data and be fed to our network again to see how well our network can generalize. When referring to "our network" that of course only means the structure of the network and not the weights and biases which will be fixed only after training, i.e. we train three different networks on three different market models and then compare them to the mixed market model. The reader will also notice that parameter names show up multiple times, while this can cause confusion, the purpose of this is to be somewhat in line with the notation found in the literature and to have parameters that do similar things in different models be recognizable as such, e.g. the Brownian Motions or the starting values of the assets. All model dynamics are assumed to be under $\mathbb{P}$, which, as mentioned before, may or may not be a martingale measure. Finally, please be reminded that we have assumed the interest rates to be zero and will therefore not mention them in our formulas below.

5.1 The Black Scholes Model

The Black Scholes (BS) model is the foundation of a lot of derivative hedging and pricing, because it is analytically very well understood in the absence of trading restrictions. Its dynamics are given by

$$dS_t = \sqrt{V} S_t dB_t, \ for\ t > 0\ and\ S_0 = s_0. \tag{5.1}$$

B is a one dimensional Brownian motion and $s_0 > 0$ and $V > 0$ are constant. The asset price at time t is given by:

$$S_t = S_0 \exp(-\frac{V^2}{2}t + VB_t).$$

The risk-neutral price at time t for a call option at time T is then:

$$P_t = S_t\Phi(d_+) - K\Phi(d_-)$$
$$d_\pm = \frac{\log(\frac{S_t}{K}) \pm \frac{V^2}{2}(T-t)}{V\sqrt{T-t}},$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution. The call option, of course, having the payoff $g(S_T) = (S_T - K)^+$ at maturity for a strike price K. And the delta-hedge of that option is given by

$$\delta_t = \Phi(d_+).$$

Sampling in the BS model is also very easy, all we need is to sample i.i.d. standard normal variables $Z_i,\ i = 1, ..., N$ and calulate:

$$S(t_{k+1}) = S(t_k)exp((-\frac{1}{2}V^2)\Delta t + \sqrt{\Delta t}VZ_k),$$

where $t_{k+1} - t_k = \Delta t$ for all time-steps.

5.2 The rBergomi Model

Rough volatility models are known to reflect real market better than classical Markovian models, in the sense that log-volatility in real markets behaves like a fractional Brownian motion with Hurst parameter of around $H = 0.1$, see [12], the Hurst parameter determining the degree of roughness of the model from basically Markovian for $H = 0.5$ to the aforementioned realistic roughness of $H = 0.1$. This superiority at reflecting real markets comes at the cost of losing the assumption of a Markovian structure of our market data, which has led us to the aforementioned change in our network structure. In particular we take a look at the rBergomi model developed in [13]. The price process is given by:

$$S_t := S_0 \exp[\int_0^t \sqrt{V_u}dB_u - \frac{1}{2}\int_0^t V_u du],\ B_u := \rho W_u^1 + \sqrt{1-\rho^2}W_u^2, \tag{5.2}$$

$$V_t := v_0 \exp[\eta Y_t^a - \frac{\eta^2}{2}t^{2a+1}],\ Y_t^a := \sqrt{2a+1}\int_0^t (t-u)^a dW_u^1.$$

This time with two independent Brownian motions W^1 and W^2, $\rho \in [-1, 1]$, $v_0, \eta > 0$ and $a = H - \frac{1}{2} \in (-\frac{1}{2}, 0)$ where the notation is taken from the git repository, associated with [14], who have conveniently uploaded an implementation of the sampling procedure below, which has been adapted to fit our framework.
Assuming again we can sample normal random variables, the challenge this time lies in accurately simulating the process Y_t^a, which is a so called Riemann-Liouville process for $a \in (-\frac{1}{2}, 0)$. This is a centered, locally $(a + \frac{1}{2} - \epsilon)$-Hölder continuous, Gaussian process with $Var[Y_t^a] = t^{2a+1}$ and not a martingale. The sampling relies on a first order variant of the hybrid scheme proposed in [15], based on the approximation

$$Y_t^a \approx \sqrt{2a+1}\left(\int_{\frac{i-1}{n}}^{\frac{i}{n}} (\frac{i}{n} - s)^a dW_u^1 + \sum_{k=2}^{i} (\frac{b_k}{n})^a (W^1_{\frac{i-(k-1)}{n}} - W^1_{\frac{i-k}{n}})\right),$$

where

$$b_k := \left(\frac{k^{a+1} - (k-1)^{a+1}}{a+1} \right)^{\frac{1}{a}}.$$

With the sum above being evaluated via the fast Fourier transform we can approximate a discretization $Y_0^a, Y_{\frac{1}{n}}^a, ..., Y_{\frac{\lfloor nt \rfloor}{n}}^a$. The rest of the price and volatility processes can then be sampled using the standard methods. We will now also introduce a second asset to be traded in to complete the market. From [4] we take the definition of the forward variance with maturity T_f and dynamics

$$d\hat{\Theta}_{T_f}^t = \hat{\Theta}_{T_f}^t \sqrt{2a+1)} \eta (T_f - t)^a dW_t^1$$

and the solution

$$\hat{\Theta}_{T_f}^t = \mathbb{E}\left[\int_0^{T_f} V_s ds | \mathcal{F}_t \right] = \xi \exp\left[\Theta_{T_f}^t + \frac{1}{2}\eta^2 [(T_f - t)^{2a+1} - T_f^{2a+1}] \right],$$

with $\Theta_{T_f}^t := \sqrt{2a+1}\eta \int_0^t (T_f - r)^a dW_r$. The forward variance was sampled using the Euler scheme.

5.3 The Heston Model

Yet another take on volatility is proposed by Heston in his 1993 paper [3] with dynamics

$$dS_t = \sqrt{V_t} S_t dB_t, \ S_0 = s_0 \tag{5.3}$$

$$dV_t = \alpha(b - V_t)dt + \sigma\sqrt{V_t} dW_t, \ V_0 = v_0, \tag{5.4}$$

where $\alpha, b, \sigma, v_0, s_0$ are positive constants and B and W are two Brownian motions with a correlation coefficient $\rho \in [-1, 1]$. The price process looks suspiciously similar to that of the BS model, the only difference being the time dependence of the volatility. The volatility process is described by a Cox-Ingersoll-Ross (CIR) process. Again, because there are two risk factors, trading in S alone is not sufficient, we therefore introduce a simplified variance swap with price

$$\hat{\Theta}_t = \mathbb{E}\left[\int_0^T V_s ds | \mathcal{F}_s \right]. \tag{5.5}$$

Lemma 3. *The price of a variance swap in the Heston model defined in (5.5) is given by:*

$$\hat{\Theta}_t = \int_0^t V_s ds + L(t, V_t), \ \textit{with} \ L(t, V_t) := b(T-t) + \frac{(V_t - b)}{\alpha}(1 - e^{-\alpha(T-t)}) \tag{5.6}$$

Proof. We start with the dynamics of V:

$$\begin{aligned}
&dV_t = a(b - V_t)dt + \sigma\sqrt{V_t}dW_t \\
\Leftrightarrow\ & e^{at}(dV_t + aV_t dt) = e^{at}(abdt + \sigma\sqrt{V_t}dW_t) \\
\Leftrightarrow\ & d(e^{at}V_t) = e^{at}(abdt + e^{at}\sigma\sqrt{V_t}dW_t) \\
\Leftrightarrow\ & e^{at}V_t = v_0 + \int_0^t e^{as}abds + \sigma\int_0^t e^{as}\sqrt{V_s}dW_s \\
\Leftrightarrow\ & e^{at}V_t = v_0 + b(e^{at} - 1) + \sigma\int_0^t e^{as}\sqrt{V_s}dW_s \\
\Leftrightarrow\ & V_t = b + (v_0 - b)e^{-at} + \sigma e^{-at}\int_0^t e^{as}\sqrt{V_s}dW_s,
\end{aligned}$$

where the second step is integration by parts. We can now use this in our definition of the price of the variance swap.

$$\begin{aligned}
\hat{\Theta}_t &= \mathbb{E}\left[\int_0^T V_s ds|\mathcal{F}_t\right] = \mathbb{E}\left[\int_0^t V_s ds + \int_t^T V_s ds|\mathcal{F}_t\right] \\
&= \int_0^t V_s ds + \mathbb{E}\left[\int_t^T \left(b + (v_0 - b)e^{-as} + \sigma e^{-as}\int_0^s e^{ar}\sqrt{V_r}dW_r\right)ds|\mathcal{F}_t\right] \\
&= \int_0^t V_s ds + b(T - t) + \frac{e^{-at} - e^{-aT}}{a}(v_0 - b) + \mathbb{E}\left[\int_t^T \left(\sigma e^{-as}\int_0^s e^{ar}\sqrt{V_r}dW_r\right)ds|\mathcal{F}_t\right].
\end{aligned}$$

For the remaining expectation term we use Fubini's theorem to swap the order of integration and calculate:

$$\begin{aligned}
&\mathbb{E}\left[\int_t^T \left(\sigma e^{-as}\int_0^s e^{ar}\sqrt{V_r}dW_r\right)ds|\mathcal{F}_t\right] \\
=&\mathbb{E}\left[\int_0^t\int_t^T \sigma e^{-as}e^{ar}\sqrt{V_r}dsdW_r + \int_t^T\int_r^T \sigma e^{-as}e^{ar}\sqrt{V_r}dsdW_r|\mathcal{F}_t\right] \\
=&\mathbb{E}\left[\int_0^t \frac{e^{-at} - e^{-aT}}{a}\sigma e^{ar}\sqrt{V_r}dW_r + \int_t^T \frac{e^{-at} - e^{-aT}}{a}\sigma e^{ar}\sqrt{V_r}dW_r|\mathcal{F}_t\right] \\
=&\int_0^t \frac{e^{-at} - e^{-aT}}{a}\sigma e^{ar}\sqrt{V_r}dW_r + \mathbb{E}\left[\int_t^T \frac{e^{-at} - e^{-aT}}{a}\sigma e^{ar}\sqrt{V_r}dW_r|\mathcal{F}_t\right] \\
=&\frac{e^{-at}(1 - e^{-a(T-t)})}{a}\sigma\int_0^t e^{ar}\sqrt{V_r}dW_r
\end{aligned}$$

Putting this back into the previous calculation of (5.5) yields:

$$\begin{aligned}\hat{\Theta}_t &= \int_0^t V_s ds + b(T-t) + \frac{e^{-at} - e^{-aT}}{a}(v_0 - b) + \frac{e^{-at}(1 - e^{-a(T-t)})}{a}\sigma \int_0^t e^{ar}\sqrt{V_r} dW_r \\ &= \int_0^t V_s ds + b(T-t) + \frac{(1 - e^{-a(T-t)})}{a}\left(e^{-at}(v_0 - b) + \sigma e^{-at}\int_0^t e^{ar}\sqrt{V_r} dW_r\right) \\ &= \int_0^t V_s ds + b(T-t) + \frac{(1 - e^{-a(T-t)})}{a}(V_t - b)\end{aligned}$$

□

While the sampling can always be done with a simple Euler Scheme, it would yield very poor results even if we were to use some of the possible "fixes", see [16]. We instead use the following procedure originally proposed in [17], where we use a crucial simplification proposed in [18]:

1. Sampling V_t for a given V_u, $t > u$, whose distribution then is a noncentral chi-squared distribution times a factor.

$$V_t \overset{d}{=} \frac{\sigma^2(1 - e^{-\alpha(t-u)})}{4\alpha}\mathcal{X}_d^2\left(\frac{4\alpha e^{-\alpha(t-u)}}{\sigma^2(1 - e^{-\alpha(t-u)})}\right),$$

$$d = \frac{4\alpha b}{\sigma^2}.$$

2. Sample $\int_u^t V_s ds$ given V_t and V_u via the approximation:

$$\int_u^t V_s ds | V_s, V_t \approx V_s.$$

3. Use the quantities from the first two steps to recover $\int_u^t \sqrt{V_s} dW_s$ from (5.4).

4. Generate a sample $S_t = \exp(m(u,t) + std(u,t)Z)$,

where Z is a standard normal variable, $std(u,t) = \sqrt{1 - \rho^2 \int_u^t V_s ds}$ and

$$m(u,t) = \log(S_u) + \left[-\frac{1}{2}\int_u^t V_s ds + \rho \int_u^t \sqrt{V_s} dW_s\right].$$

The swap was then calculated with a simple interpolation.

5.4 The Merton Jump Diffusion Model

Originally proposed by Merton in [2], the Merton model adds discontinuity to the BS model. This is done by introducing a compound Poisson jump process into its dynamics:

$$dS_t = \lambda k S_t dt + V S_t dB_t + (y_t - 1)dN_t, \tag{5.7}$$

where $y_t \sim lognormal(e^{\mu+\frac{1}{2}\delta^2}, e^{2\mu+\delta^2}(e^{\delta^2}-1))$, which implies $\log(y_t) \sim \mathcal{N}(\mu, \delta^2)$ and N_t being a poisson process satifying the following conditions:

1. $N_0 = 0$
2. N has independent increments
3. $N_t - N_s \sim poiss(\lambda(t-s)),\ t > s.$

With parameters $\lambda, V, \delta > 0$, $\mu \in \mathbb{R}$ and $k = e^{\mu+\frac{1}{2}\delta^2} - 1$. The explicit solution to the above dynamics is

$$S_t = S_0 \exp\left[(\lambda k - \frac{V^2}{2})t + VB_t + \sum_{i=1}^{N_t} \log(y_i)\right]. \tag{5.8}$$

As in the BS model for the sampling we need to generate i.i.d. standard normal variables $Z_i,\ i = 1, ..., N$, a discretization of the poisson process, which can be obtained by sampling i.i.d. $N_{\Delta t} \sim poiss(\lambda \Delta t)$,because we operate under the assumption of equally sized time-steps. We also need to sample $N_{\Delta t}$ i.i.d. normal variables $ln(y_i) \sim \mathcal{N}(\mu, \delta)$. Now we simply plug this into our sampling formula:

$$S(t_{k+1}) = S(t_k) exp((-\lambda k - \frac{1}{2}V^2)\Delta t + \sqrt{\Delta t} V Z_k + \sum_{i=1}^{N_{\Delta t}} \log(y_i)).$$

Derivations and more information can be found in Merton's original paper, but also in an easier to read version in [19]. Since this model is supposed to be mixed into our market for some of our numerical experiments we also need a second tradeable asset. Similar to our previous two models we chose:

$$\hat{\Theta}_t = \mathbb{E}\left[\int_0^T V ds | \mathcal{F}_s\right] = VT,$$

because V is not stochastic.

5.5 Calibration

We wish to hedge in above market models both individually and at the same time. Realistically this can only be done, if we calibrate the parameters of these models to the

same market. To make things easier and reduce the number of models we need to calibrate, we assume the rBergomi market to be the "true" market and calibrate the other models to it. The rBergomi model is the best choice for this as it most closely reflects real markets and would be difficult to calibrate otherwise. This also has the advantage, that we don't need to gather market data and instead can simulate as much of it as we need with the procedure described in the rBergomi section. For the parameters of the rBergomi model we took the parameters of [4], who achieved a remarkable fit to SPX options data on Febuary 4 in 2010. We did however change the Hurst parameter slightly from 0.07 to 0.1 to be in line with [6], who also mention a slightly different correlation in their paper, but that is just a typo. As a quick side note, one might think that chosing the rBergomi model as the "true" market, the other models are calibrated to gives it an advantage in the hedging process later on. However this is not the case, as once the parameters are calibrated the models are used as input for our network equally and hedging errors are equally costly in all models, in fact there are no models as far as our network is concerned, only input data and a resulting hedging error.
The calibration of the Heston model is largely done with the help of QuantLib, an open-source library for quantitative finance. We use Monte-Carlo simulation to generate an implied volatility surface from the rBergomi model for different maturities and strikes of our call option and then use QuantLib's built in Levenberg-Marquardt optimization algorithm to find the best Heston parameters, fitting the Heston implied volatilities to the rBergomi ones.
The calibration of the BS model, even though it will not be included in the mixed market, was done to help with the calibration of the Merton model. We calculated the analytical price of our call option on the parameters of the Heston model using QuantLib and then calculated the BS volatility using the inversion formula.
Originally the calibration of the Merton model was done by again using a Monte-Carlo simulation in the rBergomi model to generate prices (not implied volatilities this time) of call options at our chosen maturity for different strikes and then fitting the Merton parameters to match that price surface employing the SLSQP (Sequential Least Squares Programming) optimization algorithm from scipy. This unfortunately yields poor results, not because the calibration is bad, but because a good calibration is bad in this case. We introduced the Merton model as introducing discontinuity to the BS model in the form of jumps, however the model resulting from this calibration was a discontinuity with some BS features. The average jump size and frequency simply overpowered the BS characteristics of the model and plotting sample paths next to paths of the original rBergomi model makes it clear that this is only the same market in terms of the call option price. What makes this unacceptable is the fact that, as even Merton agreed, random jumps cannot be hedged-out even in continuous time and therefore feeding our hedging network nothing but jumps creates a "garbage in garbage out" scenario. The solution was taking the BS volatility and lowering it by a certain amount. That way, because the total volatility of the Merton model is made up of both the BS volatility and its jump parameters, the resulting jump parameters can be controlled by how much we lowered the BS volatility.

Table 5.1: **Calibrated Model Parameters**

Model	Parameter	Value
rBergomi	v_0	0.0552
	a	0.1
	η	1.9
	ρ	-0.9
Heston	v_0	0.0512
	α	3.981
	b	0.0933
	σ	1.7042
	ρ	-0.7165
Merton	V	0.198
	λ	2.083
	μ	0.0
	δ	0.0489
BS	V	0.2083

While achieving a perfect calibration is not the purpose of this thesis we should at least see how well we have done in that regard. To this end we have calculated price surfaces of European call options for our three main models, the rBergomi, the Heston and the Merton model using our parameters from above. Using these prices surfaces we can then create heatmaps showing the relative difference between prices in different models for any set of strike and maturity.

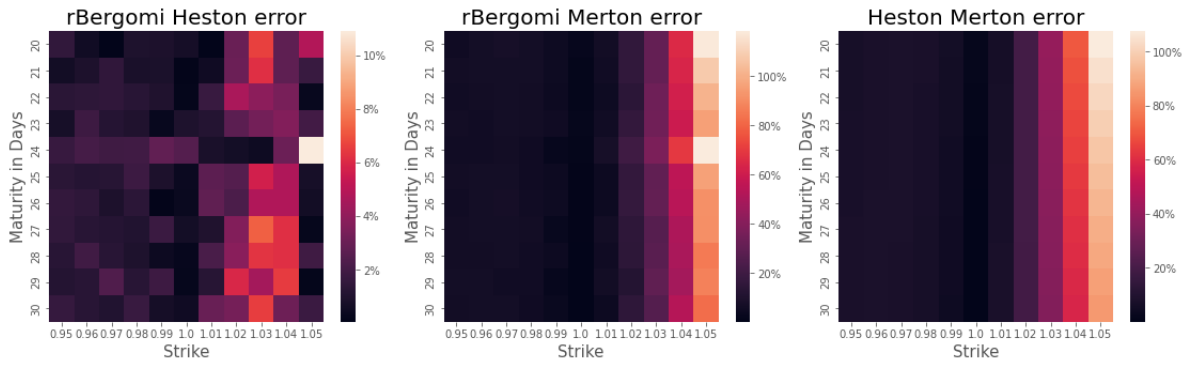

Figure 5.1: Calibration Heatmap

The leftmost figure shows that our calibration of the Heston model was quite successful, especially for the most relevant at the money strike prices. While the two figures involving the Merton model look quite bad with relative errors up to 100% we are mostly interested in the quality of our calibration at the money for a maturity of around a month, which is where we achieve a similar results to the rBergomi vs. Heston case.

In our calibration process we have ignored our variance swaps/forwards and for good reason. While these assets are also influenced by the model parameters and are technically based on the same conditional expectation formula, due to the very different nature of the volatilities in all models the resulting assets are always going to be very different. What could have been done is leaving the parameters as is and applying linear transformation to the asset paths and make all the swaps/forwards have the same expectation and variance. However this was not done, because the expectation of an asset is irrelevant as one can simply adapt the position size proportionally and changing the variance would interfere with the quality of the hedges in the one market model hedging experiments.

6 Numerical Experiments

Before we show the results let us remind ourselves of the setting we chose and fill in the quantities that were left open.
The task we chose for our experiments is hedging a standard call option with a maturity of one month $T = \frac{30}{365}$, an at the money strike $K = S_0$ and payoff $|S_T - K|^+$. The tradeable assets are as described in the previous chapter. We chose daily rebalancing, i.e. N=30, and thus $t_0 = 0, t_1 = \frac{1}{365}, ..., t_N = \frac{30}{365}$. The variance swaps have the same maturity as our option and the variance forward of the rBergomi model has maturity $T_f = \frac{45}{365}$ to avoid a singularity as t catches up the maturity in $\Theta^t_{T_f}$

As mentioned before, there was only very little "outer optimization" done for our network parameters, starting with the parameters proposed in [5] and making the structural changes proposed in [6] to effectively hedge despite not being able to rely on the Markov property of our assets, a little trial and error yielded the following network specifications: Each time-step is handled by a dense neural network of its own, with individual weights and biases, but all having the same structure. We call these one-period networks. The one-period networks have two hidden layers, and 17 nodes in each hidden layer, with batch normalization layers in between hidden layers and the last hidden layer and the output layer. We use the ReLU (Rectified Linear Unit) activation function $ReLU(x) = x^+$ for all but our output layer, where we instead use sigmoid activation function

$$sigmoid(x) = \frac{1}{(1 + \exp(-x))}.$$

All weights and biases were initialized as standard normal random variables. We use the aforementioned Adam for our optimization of weights and biases with a batch size of 256 and a starting learning rate of 0.05. As input all one-period models take the current price of the underlying, the current price of the variance swap/forward, the logarithm of the current price of the underlying and the current value of the variance process V_t for the Heston and rBergomi model and the BS and Merton model instead simply received their volatility parameter V as input. As output all one-period networks produce a hedging strategy and two additional numbers, that can contain anything any one-period network deems worthy of sharing with the next, we call them non-Markov output hinting at the purpose we hope they might serve. Additionally all one-period models, except the first one also receive the current trading strategy and the non-Markov output of the previous one-period network as input.

We have simulated 600,000 samples paths for each market model, 500,000 of which are used as training data and the remaining 100,000 are used as testing data. We do

not use any particular way of terminating our training process and instead fix a number of epochs for our training and track the improvement of our network by testing it on a small amount of our testing data after each epoch, one epoch being the process of using all 500,000 samples once for training. We find that improvement is very marginal after around 3 epochs. We have chosen 3 epochs for our experiments in the BS model, 15 for our experiments in the other models and 5 in our mixed hedging experiments, because the sample size is three times as large. This, we hope, strikes a good balance between the amount of training the one market and mixed market networks receive, so that their performance is truly a testament to how well they can hedge in their given markets and not just how much input data they have received for training.

The first two experiments will be done using the entropic risk measure (3.4) with risk parameter $\lambda = 1$ hedging in the BS market to demonstrate key properties of our hedging setup. For all the experiments after that we have chosen slightly a different approach proposed in [6], where our optimization problem becomes:

$$\pi(X) := \inf_{\delta \in \mathcal{H}} \mathbb{E}[(X + (\delta \cdot S)_T - C_T(\delta) + p_0)^2]. \tag{6.1}$$

The price of the quadratic hedging loss is not exactly a convex risk measure, however it delivers both better results in terms of PnL of our hedging and crucially offers a little more intuitive results.
Before we present the results let us explain some notation. NN stands for neural network, a model name with NN after it means that this network was trained solely on data of that model and 3NN and 2NN will refer to neural a network trained to hedge data from either all three models (rBergomi, Heston, Merton) or two models (rBergomi, Heston).

6.1 BS experiments

What we have not yet discussed is, how we are going to calculate the price p_0 we would like to charge someone for buying the call option above from us. The price of course is given as the amount of cash we need to add to our position to make the risk acceptable or in the case of using (6.1) we choose the price, such that it minimizes our expected quadratic hedging error. There are two different ways in our framework to calculate the price with our network, not including p_0 in our PnL and then calculating the appropriate price that we would have needed to charge after fact. The second approach is having our network train to also calculate the price p_0, while it is training to hedge. This can be done by adding a second much simpler network to our original network of networks that takes for example the starting price as input(the inputs don't matter here, since just just want to guess a constant number) and has its output plugged into the final calculation of the PnL. The first approach is basically a Monte-Carlo simulation and has its accuracy depend on the quality and quantity of the test data. The second approach has the immediate disadvantage that having a neural network train two things at once like this slows down both processes, but also has the advantage of not needing a testing set at all. Here the accuracy of the price depends on how well this additional small network is designed and can be trained alongside our original architecture and of course the quality and quantity of our training data.
With this in mind let us proceed to look at our first experiment, where we compare both of the approaches above to the Delta Hedge in the BS model and see that both approaches produce a very similar squared hedging error, which is also very close to the Delta Hedge. Unfortunately, and perhaps this is something that future work can improve, it seems that calculating the premium during training is ever so slightly biased. In fact this bias still exist with roughly the same size if we do this experiment with the other models. The reason for this is not entirely clear. Because of the slight bias of the NN+premium approach described above, in all the experiments below the price was calculated after hedging the testing data as in the NN simple approach.

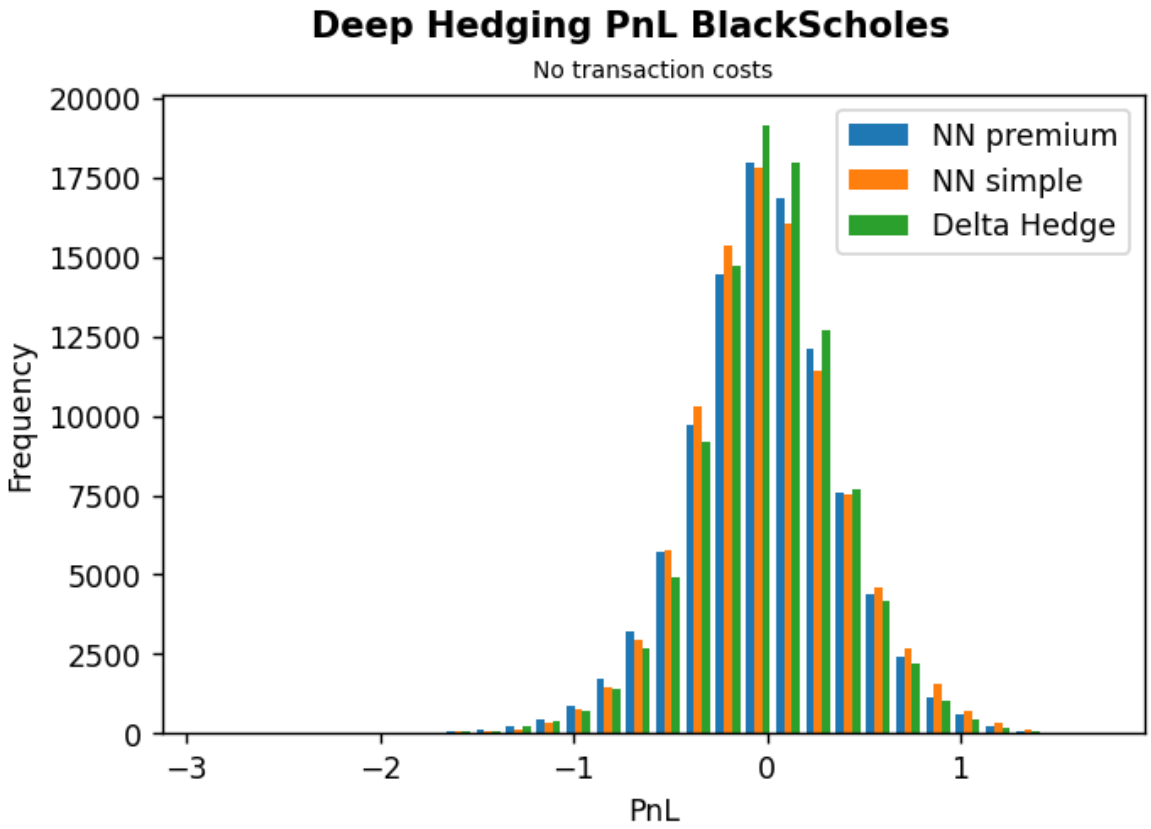

Figure 6.1: Comparison of NN hedging errors vs. Delta Hedge hedging errors on BS data.

Table 6.1: **BS hedging results**

Model	Price	Average Squared Error
Analytic	2.3826	-
Delta Hedge	2.3837	0.1388
NN+premium	2.371	0.1543
NN simple	2.3839	0.1544

To demonstrate our networks capabilities to learn to deal with trading restrictions we now introduce proportional trading costs, with cost parameters $c_k^i = \epsilon = 0.01,\ \forall i, k$. We also allow ourselves to make the almost comical comparison to the standard Delta Hedge from the non-restricted case. While the very long leftside tail of our PnL distribution for the NN hedge does not look good, it should be noted that this is literally the accepted risk when hedging under the risk preferences defined by our risk measure. Our network, as we will see later on, reduces the traded volume and accepts that we will have heavy losses some of the time.

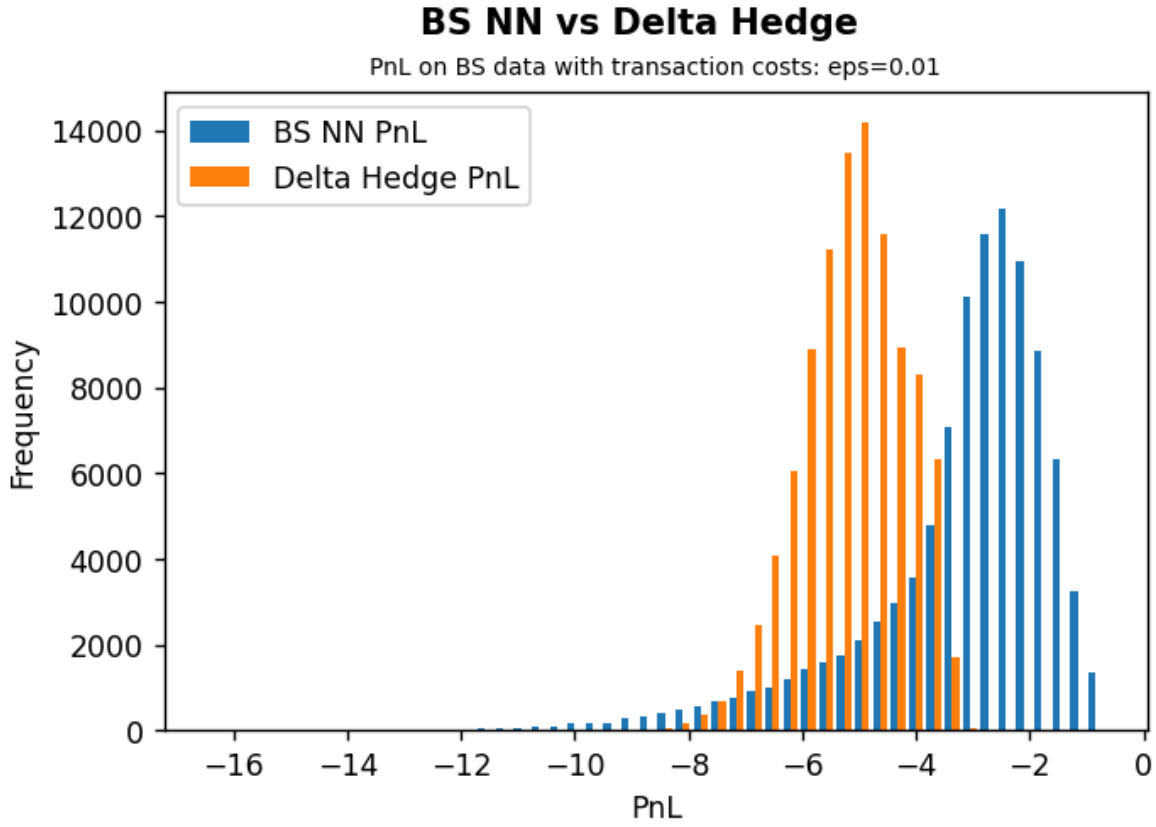

Figure 6.2: NN hedge vs. Delta Hedge with transaction costs on BS data

Table 6.2: **BS hedging results with transaction costs**

Model	Price	Average Squared Error
Delta Hedge	5.1188	0.8291
NN	3.2099	2.9328

6.2 Mixed market experiments

We now turn our eyes towards the heart of this thesis, hedging in mixed markets. We start with no transaction costs and train a network on each market model individually and then train a fourth network (3NN) on all market models at the same time and compare the 3NN in its performance to the one market models on the respective testing data.

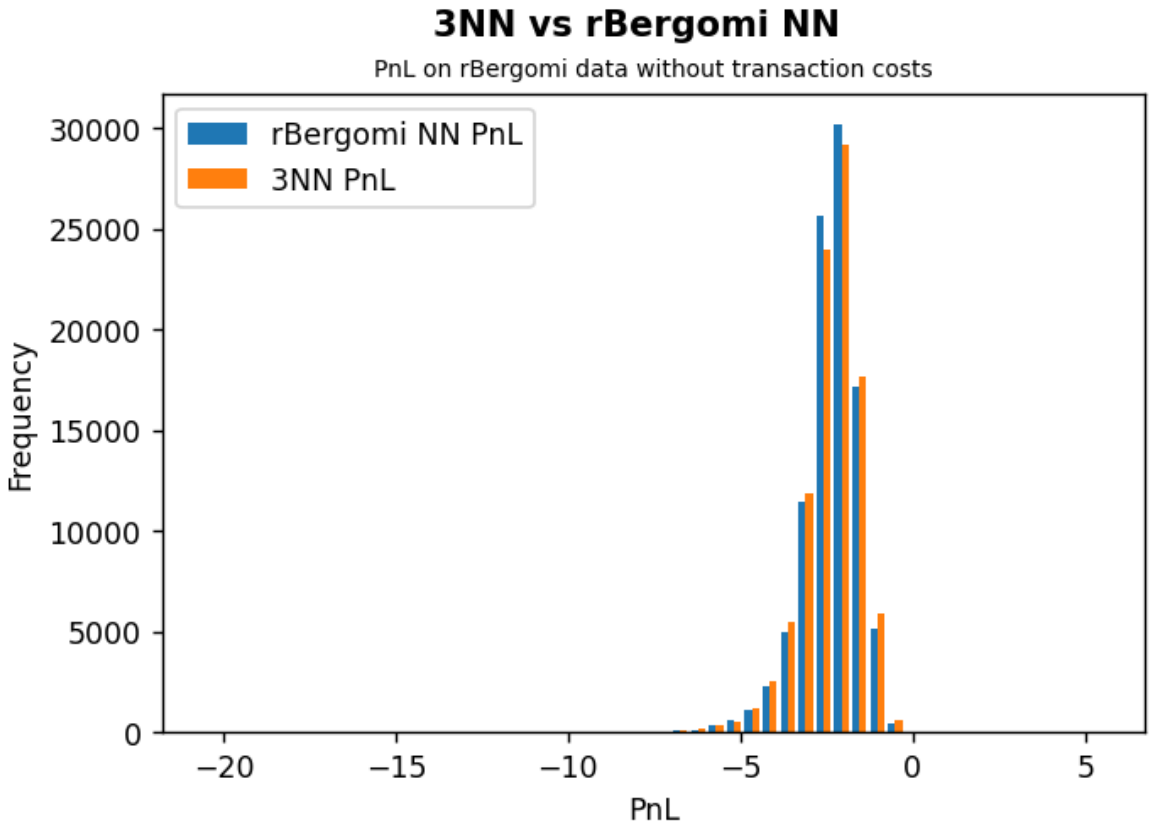

Figure 6.3: 3NN hedge vs rBergomi NN hedge on rBergomi data

Table 6.3: **rBergomi hedging results without transaction costs**

Model	Price	Average Squared Error
rBergmoni NN	2.3819	0.7293
3NN	2.3809	0.7818

We can see the 3NN can accurately calculate the price proposed by the rBergomi NN and is only very slightly worse in terms of the squared hedging error.
Moving on we take a look at the same experiment in the Heston model to find similar results.

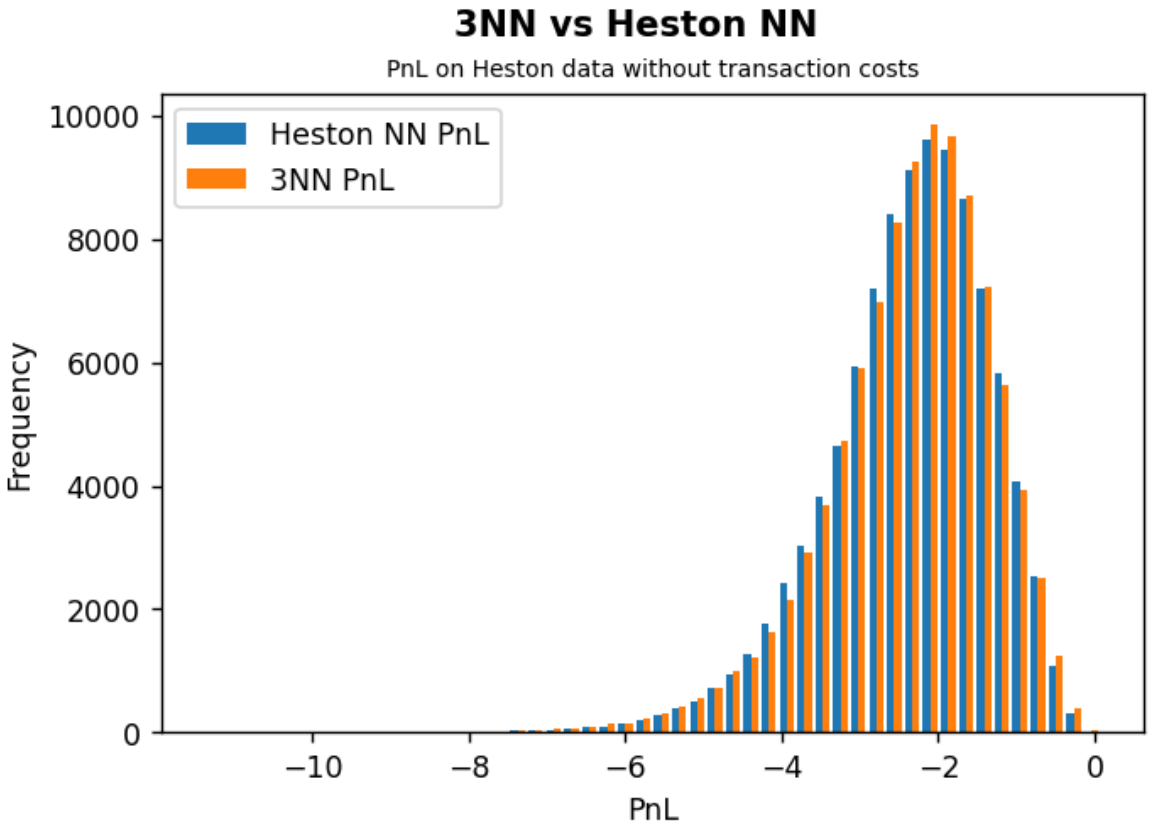

Figure 6.4: 3NN hedge vs Heston NN hedge on Heston data

Table 6.4: **Heston hedging results without transaction costs**

Model	Price	Average Squared Error
Heston NN	2.3763	1.0909
3NN	2.3748	1.1346

Finally we have our third and final market model, the Merton model. While the overall performance of our NN hedges is better than in the other market models, this is the first time we see a our 3NN hedge being significantly worse than the Merton NN hedge. The good performance is of course to some degree a result of the artificially lowered jump rate and intensity and we see just how difficult hedging such jumps is with the squared hedging error increasing threefold compared to the BS experiment with the inclusion of very minor jumps, despite the network having done five times more training.

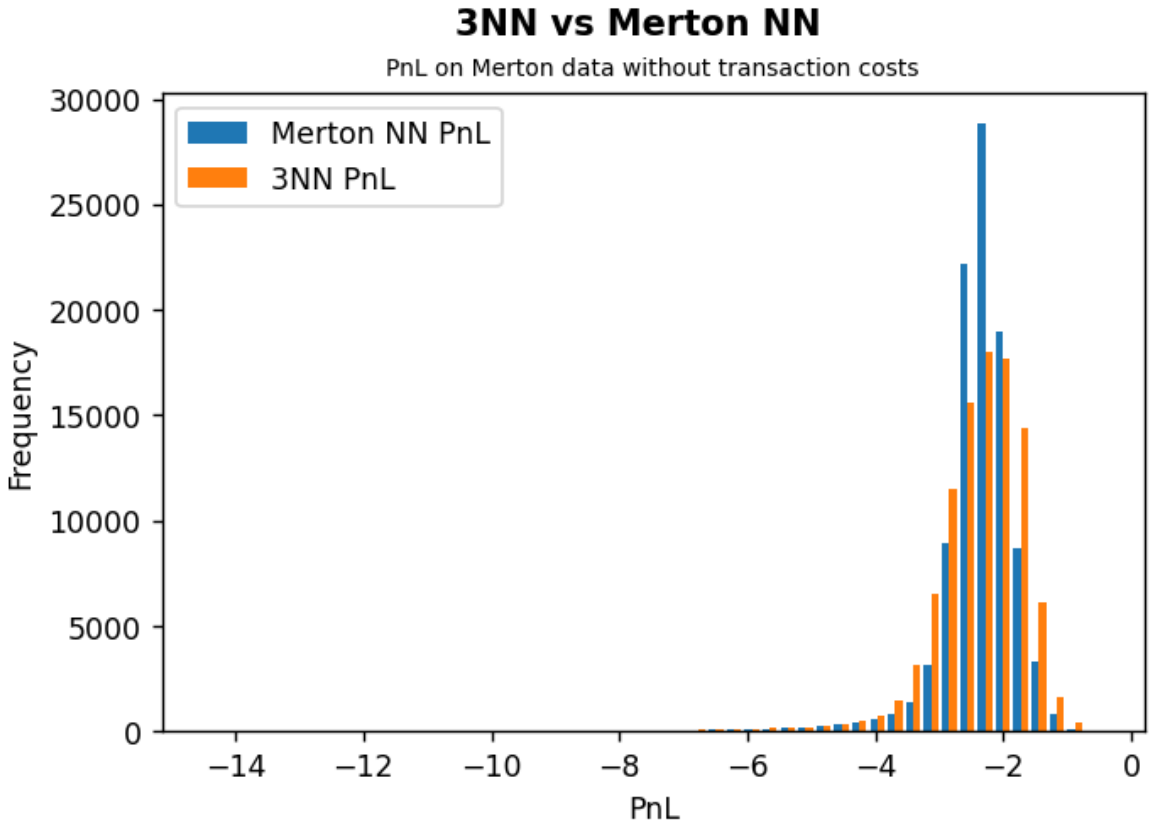

Figure 6.5: 3NN hedge vs Merton NN hedge on Merton data

Table 6.5: **Merton hedging results without transaction costs**

Model	Price	Average Squared Error
Merton NN	2.388749	0.4463
3NN	2.3912	0.617

Overall the performance of the 3NN is very impressive, losing almost none of its accuracy despite two thirds of its training data describing different markets.

It is now time to reintroduce our trading costs of $\epsilon = 0.01$ and rerun the above experiments. Starting again with the rBergomi model we can now see a bias when it comes to the price and a slightly higher squared hedging error.

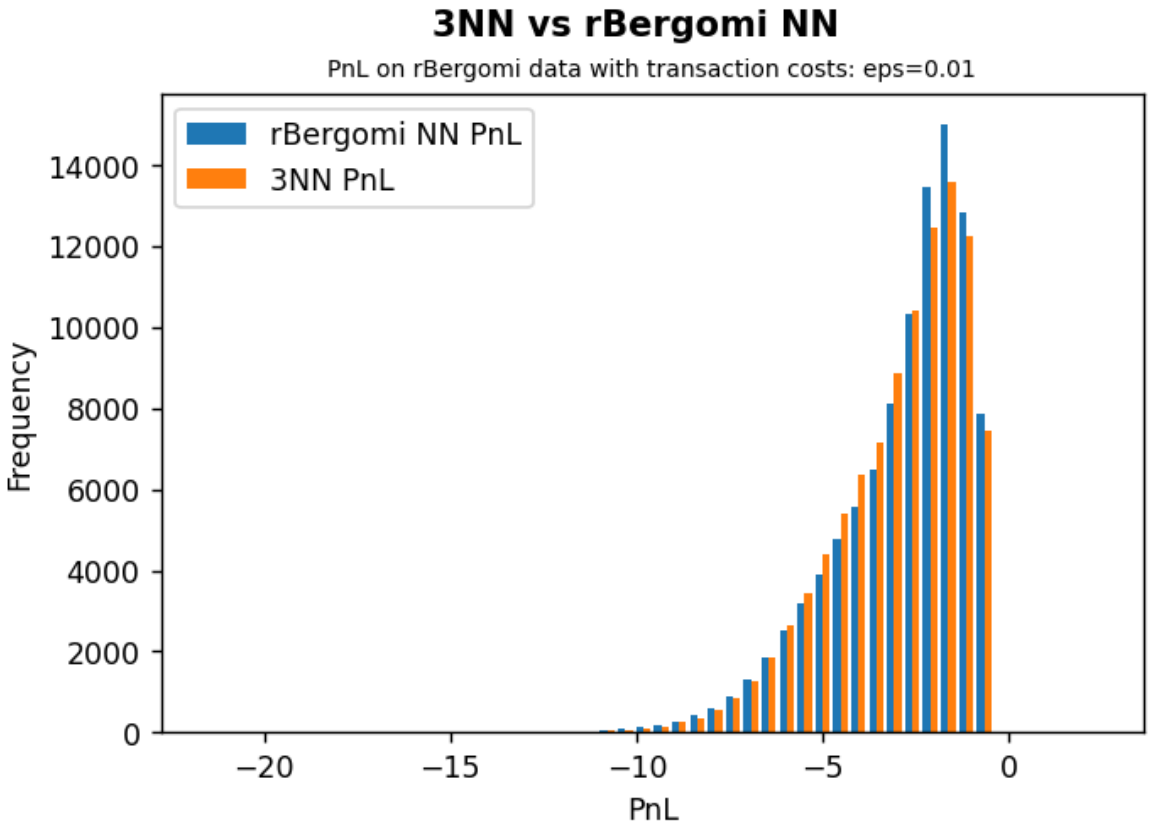

Figure 6.6: 3NN hedge vs rBergomi NN hedge with transactions costs on rBergomi data

Table 6.6: **rBergomi hedging results with transaction costs**

Model	Price	Average Squared Error
rBergmoni NN	2.8834	3.3088
3NN	2.9534	3.1759

The Heston model produces similar results, where we overestimate the price again.

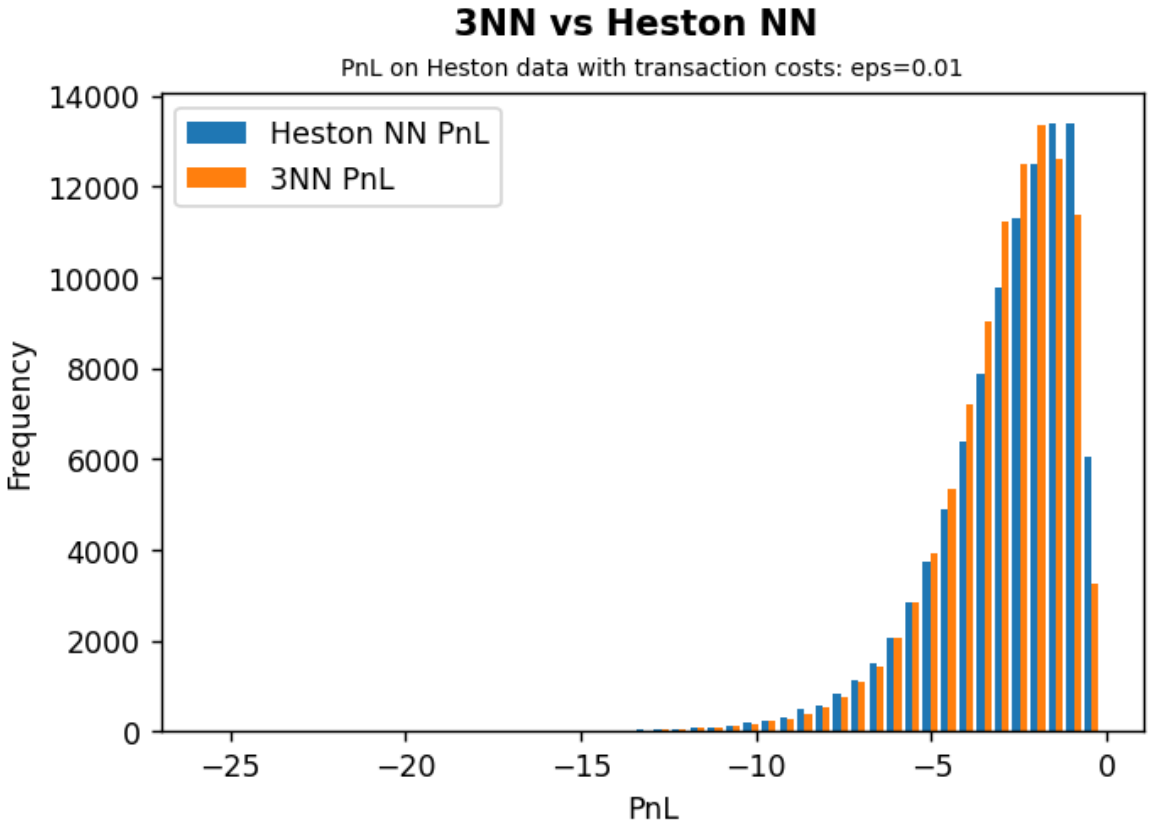

Figure 6.7: 3NN hedge vs Heston NN hedge with transactions costs on Heston data

Table 6.7: **Heston hedging results with transaction costs**

Model	Price	Average Squared Error
Heston NN	2.8542	3.7483
3NN	2.9568	3.3442

We now take a look at the Merton model to find even worse results and perhaps an understanding, where the bias is coming from. We now have a lower 3NN price than Merton NN price but a massively worse squared hedging error.

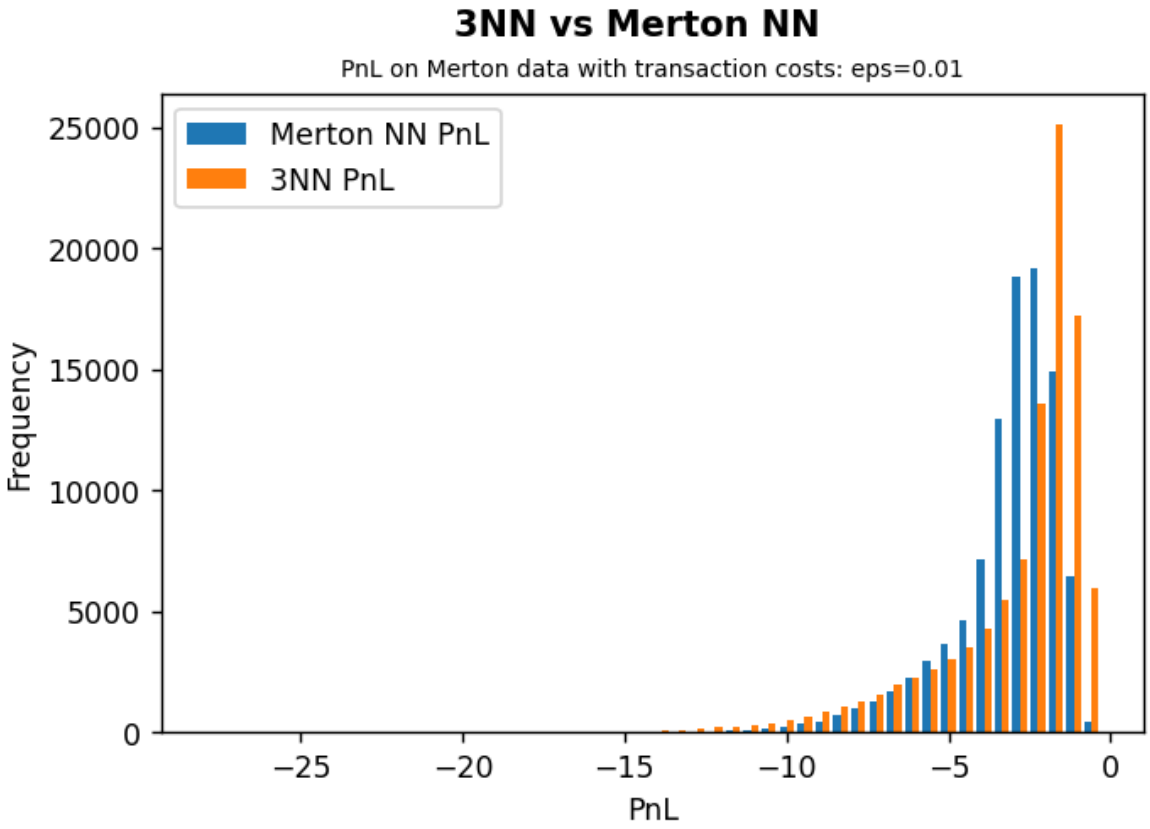

Figure 6.8: 3NN hedge vs Meeton NN hedge with transactions costs on Merton data

Table 6.8: **Merton hedging results with transaction costs**

Model	Price	Average Squared Error
Merton NN	3.2213	3.1147
3NN	2.9619	5.7211

Whereas the restriction-free experiments yielded very nice results, the introduction of trading costs into our mixed market seems to throw our network off its game. We now try to find the reason or at least some intuition as to why that is the case. We notice that the prices rose across the board once we introduced trading costs, the reason for that is simple, trading costs make our replicating portfolio of the payoff of our liability more expensive and therefore we need to charge a higher price. We should also note that the one model networks all have different prices now with the trading cost. That is because the price is now dependent on a previously irrelevant quantity, the traded

volume. The total trading volume is calculated by accumulating $|\delta_k - \delta_{k-1}|S_{t_k}$ over all periods including buying into our initial position and liquidizing our final position. Tracking the total traded volume for all of our paths and the calculating the average gives us the following table.

Table 6.9: **Average total traded volume in our hedging**

Network/Model	without transaction costs	with transaction costs
rBergomi NN	265.45	55.79
3NN	264.57	57.86
Heston NN	307.55	48.02
3NN	264.87	57.94
Merton NN	312.78	84.28
3NN	269.08	57.63

Doing the simple calculation of taking the prices in the no transaction cost cases from before and adding $0.01 \cdot \textit{total traded Volume}$ to it, brings us remarkably close the the prices in the cases with transactions costs. The different prices in the case of transaction costs are a function of the optimal trading volume, which is different in each model. It is clear now that in the presence of market frictions our calibrated market models no longer describe the same market and as such can't be hedged together very accurately. Ideally we would calibrate our market parameters again, this time in a market with frictions and then do our experiments again, however calibration in such markets is not a trivial task and will be left for future research. The best we can do for now is to throw out the Merton model, because its optimal trading volume is the most different and train a new mixed market network on the remains: 2NN. This is already much closer to the

Table 6.10: **2NN hedging results with transaction costs**

Model	Price	Average Squared Error
rBergmoni NN	2.8834	3.3088
2NN	3.0474	3.3442
Heston NN	2.8542	3.7483
3NN	3.0421	3.5876

result we were hoping for, but because the models do describe slightly different markets we should take these results with a grain of salt. In fact in a way these results are bad, because if our NN approach was truly model free and completely robust the 3NN model should deliver the same results as the 2NN model and not be deterred by misleading Merton market data and the 2NN model should produce the same results as the one market models, but of course such absolute expectations are unrealistic. Overall these are good results considering how difficult hedging with transaction costs can be.

7 Conclusion and Outlook

We have seen a remarkable ability of our hedging algorithm to perform on all four given markets individually and on the mixed market that excludes the BS model, losing almost none of its accuracy when pricing and hedging, despite two thirds of the training data being fundamentally different from the testing data. We have seen that we can also train our network to calculate the price and the hedging strategies at the same time, although this approach perhaps needs an improved implementation in future work. We have seen an impressive ability of our network to adapt to the introduction of trading costs and hedge as well as possible in the individual markets, but losing quite a bit of accuracy in its pricing and hedging when trained on the mixed market. Overall this shows great promise, that this approach and especially future versions of this approach (for example doing an exhaustive outer optimization over NN model parameters) are a great way to hedge when model assumptions are to be relaxed. This is because this approach has proven to be impressively robust with regards to the dynamics of its market data input as well as dealing with trading restrictions.
What is left to do? The aforementioned outer optimization to truly make the most out of the deep hedging approach, fixing the premium calculation while training and rerunning some of the experiments with market model parameters calibrated to markets with restrictions. Once all this is done an application to real world data might be the next thing to do. There is two ways for this to be done, the first would already work with the framework from this thesis: Calibrate a model to market data (rBergomi in this case), use that model as input to train the network and then test the networks performance on the real world market data. The second and perhaps less likely to succeed approach would be to throw out financial models altogether and do all the training and all the testing on real world data, the problem of course being that it is hard to find enough high quality training data.

8 Appendix

The code used for all numerical experiments can be found here, where some additional references are made to code that was used or adapted from online sources. We conclude with a list of Python libraries:

Table 8.1: **Installed packages**

Package	Version
tensorflow	2.5.0
Keras	2.4.3
py-vollib-vectorized	0.1.1
QuantLib	1.26
seaborn	0.11.2
sklearn	0.0

These are not all the packages that were used, but a lot of packages require other packages, for example installing tensorflow requires installing numpy and will install a compatible version automatically if the installation is done via pip. The Python version used is 3.8.8.

Bibliography

[1] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *J. Polit. Econ.*, 81(3):637–654, 1973.

[2] Robert C. Merton. Option prices when underlying stock returns are discontinuous. 1976.

[3] Steven Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of Financial Studies*, 6:327–343, 1993.

[4] Jim Gatheral Christian Bayer, Peter Friz. Pricing under rough volatility. 2016.

[5] Josef Teichmann Ben Wood Hans Bühler, Lukas Gonon. Deep hedging. 2018.

[6] Zan Zuric Blanka Horvath, Josef Teichmann. Deep hedging under rough volatility. 2018.

[7] Alexander Schied Hans Föllmer. *Stochastic Finance An Introduction in Discrete Time*, volume 4. De Gruyter, 2016.

[8] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. 1991.

[9] Stefan Richter. *Statistisches und maschinelles Lernen*. Springer, 2019.

[10] Jimmy Leo Ba Diederik Kingma. Adam: A method for stochastic optimization. 2015.

[11] Christian Szegedy Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[12] Mathieu Rosenbaum Jim Gatheral, Thibault Jaisson. Volatility is rough. 2014.

[13] Mikko S. Pakkanenl Ryan McCrickerd. Pricing under rough volatility. 2018.

[14] Jim Gatheral Christian Bayer, Peter Friz. Turbocharging monte carlo pricing for the rough bergomi mode. 2016.

[15] Mikko S. Pakkanen Mikkel Bennedsen, Asger Lunde. Hybrid scheme for brownian semistationary processes. 2017.

[16] Christian Khal Leif B.G. Andersen, Peter Jackel. Simulation of square-root processes. 2010.

[17] Özgür Kaya Mark Broadie. Exact simulation of stochastic volatility and other affine jump diffusion processes. 2006.

[18] Antoon Pelsser Alexander van Haastrecht. Efficient, almost exact simulation of the heston stochastic volatility model. 2008.

[19] Kazuhisa Matsuda. Introduction to merton jump diffusion model. 2004.