Imperial College London

MSc Statistics Thesis

Imperial College London

Department of Mathematics

Hedging by Deep Learning

Author:
Hang Lou

CID:
00945795

Supervisor:
Dr. Mikko Pakkanen

Declaration:
The work contained in this thesis is my own work unless otherwise stated

Signature:

September 5, 2018

Abstract

This thesis addresses how derivative hedging problem can be tackled by applying deep reinforcement machine learning method in both complete and incomplete market models.

I firstly trained a deep learning network with respect to quadratic loss function to reproduce the Black-Scholes hedging strategies to a high degree of accuracy. As the volatility is unknown in practice, improvements on top of the initial model were made involving an implicit estimation of the volatility with historical data during the training. I show that my results are mostly consistent with delta hedging ratio for a relatively narrow range of volatility samples.

In the presence of market friction such as transaction cost, the Black-Scholes model violates. Pricing and hedging of derivatives can be fulfilled by using some utility functions, i.e in our case exponential utility, as an alternative of standard quadratic loss. Finally, I illustrated my approach by showing the effect on pricing and hedging by both risk-averse parameter and transaction cost.

Overall, three neural network models were built progressively with different neural network architectures and loss functions to match specified market scenarios and trading requirements. I illustrated that the neural network approach is consistent with most analytical results under different parameter settings.

Acknowledgements

I would like to thank my supervisor Dr Mikko Pakkanen for his constant support and guidance on my Master thesis. I would also like to thank my personal tutor Dr Tony Bellotti for his continued assistance and study advices over the past year. .

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Options are financial contracts between two parties who will have the right but not the obligation to sell or buy a prespecified quantity of the underlying at the prespecified price, also known as the strike price. The value of the option is dependent on the price of the underlying which can be stocks, FX rates, commodity etc. Furthermore, European style call option is the common option which the buyer has a right to buy the underlying at strike price at the time of maturity. In this thesis, my study is focused on the European call option. The seller of a European call option accepts in principle unlimited liability for losses, unless the seller manages the risk by hedging. Therefore, the problem of pricing and hedging of a option is crucial for pricing risk-management in the financial industry. It has been recently suggested that the hedging problem could be tackled more systematically by applying deep learning.

The theory of option pricing and hedging is strongly based on the seminal articles by Black and Scholes[BM73] and Merton[Met73]. In an idealised, frictionless and "complete market" model, the Black-Scholes model provides a analytical closed-form solution for option pricing and hedging. In this thesis, my first goal is to build up a deep neural network which can learn the Black-Scholes hedging strategy.

In practice, though, volatility of the underlying is unknown and inconstant. This requires traders to estimate the volatility of the underlying from historical market data. I incorporate this idea with my previously built model. The hope is, therefore, the deep neural network firstly is able to extract the information of volatility from the simulated historical data, and secondly learn the Black-Scholes strategies with current prices.

Furthermore, in the real market, trading in any instrument is subject to transaction cost, permanent market impact and liquidity constraints. In incomplete markets, pricing and hedging can be solved by using convex risk measure, e.g. exponential utility in our case, according to [Xu06]. Motivated by the recent studying in [BGTW18], I model the trading decisions in our hedging strategies as neural networks under transaction costs.

Our analysis is based on out-of-sample performance. To calculate our hedging strategies numerically, I approximate them by deep neural network. This is implemented in Python using Keras[CO15] which uses TensorFlow and Theano as back ends. In our experiments, I used Adam[KB17] mini-batch gradient descent training for a reinforcement learning problem.

The rest of the article is structured as follows. Section 2 gives some finance preliminaries for pricing and hedging in both complete and incomplete market models. Section 3 describes the deep learning algorithm in details and extends to some recent neural network applications in option hedging and pricing. Section 4 outlines the data, neural network structures and performance measures are used for different market scenarios. In Section 5, I presented several experimental results to demonstrate the feasibility and accuracy of hedging by the neural network approach.

Chapter 2

Finance Preliminaries

2.1 Hedging of European options in discrete time

A European call option is a contract that give the buyer a right to buy a share in the underlying stock from the seller at time T at price K>0. Thus we see that, mathematically, the call option X can be represented by a unlinear function of S_T:

$$X = f(S_T) = \begin{cases} S_T - K & , S_T > K \\ 0 & S_T \leq K \end{cases} \tag{2.1}$$

In order to hedge the call option X at time T, we need to rebalance the portfolio periodically to adjust to the changing prices of underlying and other hedging instruments, such as bond.

Hedging. An adapted and self-financing portfolio π perfectly hedge the derivative X if:

$$\mathbf{P}[V_T^\pi = X] = 1 \tag{2.2}$$

Where $V_T^\pi = \pi_T^0 B_T + \pi_T^1 S_T$ is the terminal value of a portfolio π. In the other word, we say the derivative X is attainable.
Recall the definitions of portfolio , adaptedness and self-financing under discrete time in [Pak18].

Defition 2.1.1. A portfolio is a sequence of $\pi = (\pi_t)_{t=0}^T$ of random vectors $\pi_t = (\pi_t^0, \pi_t^1)$, where π_t^0 denotes the number of units of the bond and π_t^1 is the number of shares in the underlying at time t.

Defition 2.1.2. A portfolio π is adapted if for any $t = 0, 1, ..., T$, we have $\pi_t = f_t(S_0, S_1, ..., S_t)$ for some measurable function $f_t : (0, \infty) \to \mathbb{R}^2$

Defition 2.1.3. A portfolio π is self-financing if for any $t = 0, 1, ..., T$,

$$V_t^\pi = \pi_{t-1}^0 B_t + \pi_{t-1}^1 S_t \tag{2.3}$$

where, V_t^π is the value of the portfolio, B_t is the bond price and S_t is the underlying stock price at time t.

A portfolio π must satisfies adaptedness and self-financing condition. Adaptedness makes sure the there is no knowledge of the future prices which would essentially be insider information. Self-financing condition is a inter-temporal budget condition. It also requires that the proceed from sold shares or bonds must be reinvested in bond or shares and not used, e.g., for consumption.

Simplifications. For notational simplicity, we assume that all intermediate payments are accrued using a risk-free overnight rate. This essentially means we may assume that rate is zero.

Remark 2.1.4. An alternative way of formulating the self-financing condition to postulate that for any $t = 1, 2, ..., T$:

$$V_t^\pi - V_{t-1}^\pi = \pi_{t-1}^0 (B_t - B_{t-1}) + \pi_{t-1}^1 (S_t - S_{t-1}) \tag{2.4}$$

Under zero risk-free rate, the bond price is constant at B , the equation (2.3) can be reduced to:

$$V_t^\pi = V_{t-1}^\pi + \pi_{t-1}^1(S_t - S_{t-1}) \tag{2.5}$$

For all $t = 1, 2, ..., T$.

Combine the equation (2.4) at every time point, we can substitute the intermediate portfolio values, $V_1^\pi, V_2^\pi, ..., V_{T-1}^\pi$ and formulate to:

$$V_T^\pi = V_0^\pi + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t) \tag{2.6}$$

Where V_0^π is the initial cash injection to hedge the liability $-X$, it is also known as the price of the derivative p_0.

Remark 2.1.5: If we are particular interested in a non-zero risk-free parameter, we can consider discounted process such that:

$$\widetilde{V}_T^\pi = \tilde{V}_0^\pi + \Sigma_{t=0}^{T-1}\pi_t^1(\tilde{S}_{t+1} - \tilde{S}_t)$$

Where $\widetilde{V}_t^\pi = \frac{V_t^\pi}{B_t}$ and $\widetilde{S}_t = \frac{S_t}{B_t}$ for all $t = 1, 2, ..., T$.

The hedging error of a portfolio π is given by:

$$Hedgingerror = -X + V_0^\pi + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t) \tag{2.7}$$

This is also called the Profit&Loss of the portfolio π with call option X and initial cash V_0^π at the maturity T.

2.2 Complete market model

Before we introduce the famous Black-Scholes model, we first give a formal definition of complete market model. Recall the definition of martingale and equivalent martingale measure.

Definition 2.2.1. A stochastic process $M = (M_t)_{t=0}^T$ is a martingale under the probability measure $\mathbf{P}$ if

1. M is adapted
2. $\mathbf{E}_{t-1}^{\mathbf{P}}[M_t] := \mathbf{E}[M_t|S_{t-1}, S_{t-1}, ..., S_0] = M_{t-1} \quad \text{for} \quad \text{any} \quad t = 1, ..., T$

Then a probability measure Q is an equivalent martingale measure (EMM), under which the discounted price process $(S_t/B_t)_{t=0}^T$ is a martingale defined above.

Remark 2.2.2 An EMM is also called a *risk-neutral measure*.

The complete model is defined by the following theorem.

Theorem 2.2.3 *Second Fundamental theorem of asset pricing*, see Theorem 5.9 of [Rom12]:
A market model has a unique EMM if and only if it is complete, which means that every European derivative in the model is attainable.

2.3 Black-Scholes and Delta hedging

Black–Scholes model [BM73], motivated by the complete model, is a continuous-timed market model for the dynamics of a financial market containing derivative instruments. The Black-Scholes

molde consists of a bond and a stock, with prices $(B_t)_{t\in[0,T]}$ and $(S_t)_{t\in[0,T]}$ that evolve in continuous time according to equation:

$$B_t = B_0 e^{rt}, \qquad S_t = S_0 e^{(\mu - \frac{\sigma^2}{2})t + \sigma W_t} \tag{2.8}$$

Where r is the continuously compounded risk-free interest rate, σ is the annualised volatility on the returns of the stock price and W_t is a Brownian motion. There are number of assumptions that the Black-Scholes model makes about the market and assets:

- The rate of return on the riskless asset is constant.
- stock price is a geometric Brownian motion with constant drift and volatility.
- The stock does not pay a dividend
- There is no arbitrage opportunity
- It is possible to borrow and lend any amount, even fractional, of cash at the riskless rate
- It is possible to buy and sell any amount, even fractional, of the stock
- The above transactions do not incur any fees or costs(i.e. frictionless market)

The analytic Black-Scholes price for a European call option at time t with time to maturity $T - t$ is given by:

$$C(S_t, t) = N(d_1)S_t - N(d_2)Ke^{-r(T-t)} \tag{2.9}$$

Where

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

$$d_2 = d1 - \sigma\sqrt{T-t}$$

Where $N(\cdot)$ is the standard normal cumulative distribution function.

Delta Hedging. Under the classical Black-Scholes model, the derivative can be theoretically hedged perfectly by hold delta proportion of the underlying continuously. Delta, Δ, is the rate of change of the theoretical option value with respect to change in the underlying asset's price. In call option the delta is given by:

$$\Delta = \frac{\partial C}{\partial S} = N(d1) \tag{2.10}$$

Note that under continuous time, the self-financing condition is given by:

$$V_T^\pi = V_0^\pi + \int_{t=0}^{T} \pi_t^1 dS_t \tag{2.11}$$

And in the Black-Scholes market model, we know that the initial cash injection is given by the Black-Scholes call option price C_{BS} and the hedging strategy is according to the Black-Scholes delta. However, the continuous time hedging is not possible in practice, so the Black-Scholes delta is approximated at each time points, which can be seen in Figure 2.1. And the self-financing condition is reduced to (2.6) in discrete time. It is worthwhile to point out that the hedging error of the Black-Scholes delta hedging is always not zero in discrete time, however it converges to 0 as the discrete time step tend to 0.

Therefore, in complete market models, the replicate portfolio π in discrete time with small time increment should match the discrete delta hedging strategies. The graph below illustrates the approximated delta hedging under discrete time for European call options with Strike price $K = S_0 = 10$.

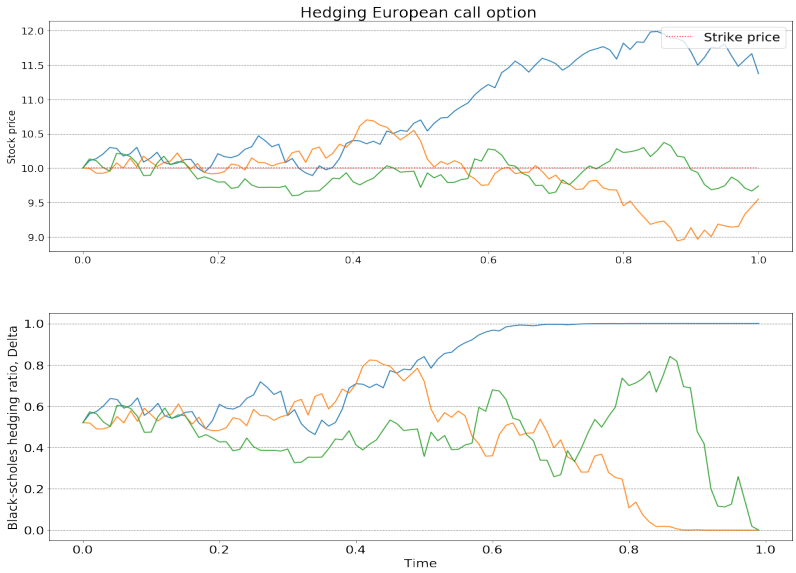

Figure 2.1: Black-Scholes hedging for at the money European call option

2.4 Hedging in incomplete market models

In an idealised complete market, we have shown in previous section that for any liabilities $-X$ there exists a unique replication strategy π and a fair price $p_0 \in \mathbb{R}$ such that $-X + p_0 + \Sigma_{t=0}^{T-1} \pi_t^1 (S_{t+1} - S_t) = 0$ holds $\mathbb{P}$-a.s. However, this is not true in the presence of trading cost.

Assume that any trading activity causes costs follows: if the agent decides to buy or sell a position $n \in \mathbb{R}$ in stock at time t, then this will incur cost $c_k(n)$. The total cost of a trading strategy π up to maturity is therefore

$$C_T(\pi) := \Sigma_{t=0}^{T} c_t(\pi_t - \pi_{t-1})$$

(recall $\pi_{-1} = \pi_T := 0$, the latter of which implies full liquidation in T). Hence, the terminal P&L in the presence of trading cost is

$$PL_t(X, p_0, \pi) := -X + p_0 + \Sigma_{t=0}^{T-1} \pi_t^1 (S_{t+1} - S_t) - C_T(\pi) \tag{2.12}$$

Here we will introduce two types of trading cost effects:

- Proportional transaction cost: for a constant $c_t > 0$, define $c_t(n) := c_t S_t |n|$
- Fixed transaction cost: for $c_t > 0$ and$\epsilon > 0$, define $c_t(n) = c_t 1_{|n| \geq \epsilon}$

In this paper, we will mainly use the proportional transaction cost in our implementation to discuss the hedging problem in an incomplete market model.

2.4.1 Exponential Utility Indifference Pricing

In an incomplete market with frictions, the writer of call option has to specify an optimality criterion which defines an acceptable price for any position. In this project, I will use exponential utility function as the optimality criterion. I used the ideas from the present framework includes exponential utility indifference pricing as studied for example in [BGTW18] and [WW97].

Definition 2.3.1. Exponential Utility

$$U(x) := -\exp(-\lambda x), \qquad x \in \mathbb{R} \tag{2.13}$$

where parameter λ is the measure of risk-aversion. Bigger λ corresponds to higher degree of risk-aversion. In particular, $\lambda = +\infty$ indicates absolute risk-aversion while $\lambda = 0$ corresponds

to risk-neutrality. Then, with the terminal wealth as defined in (2.11), the indifference price of $p(X) \in \mathbb{R}$ of X is defined by:

$$\sup_{\pi} \mathbb{E}\big[U(p(X) - X + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t) - C_T(\pi))] = \sup_{\pi} \mathbb{E}\big[U(\Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t) - C_T(\pi))\big]$$

which means that the writer is indifferent to his expected utilities between writing the option for the charge $p(X)$ and writing no option.

Due to the desirable separability of the exponential function, we have the following explicit expression for the indifference price:

$$\begin{aligned} p(X) &= \frac{1}{\lambda}\log\left(\frac{sup_{\pi}\mathbb{E}\big[U(-X + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t)) - C_T(\pi))\big]}{\sup_{\pi}\mathbb{E}\big[U(\Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t) - C_T(\pi))\big]}\right) \\ &= \frac{1}{\lambda}\log\left(\sup_{\pi}\mathbb{E}\big[U(-X + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t)) - C_T(\pi))\big]\right) - \frac{1}{\lambda}\log\left(\sup_{\pi}\mathbb{E}\big[U(\Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t) - C_T(\pi))\big]\right) \end{aligned} \tag{2.14}$$

Remark 2.3.2. It is worthwhile to point out that the indifference price is independent of the initial cash $p(X)$ for the exponential utility function.

Remark 2.3.3. Due to the separability of the exponential function again:

$$\sup_{\pi} \mathbb{E}\big[U(p(X) - X + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t)) - C_T(\pi))\big] = \exp(-\lambda p(X)) \sup_{\pi} \mathbb{E}\big[U(-X + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t)) - C_T(\pi))\big]$$

This mean that the optimal strategy π is identical for both sides. Hence, we can find the optimal hedging strategy π which maximise $\mathbb{E}\big[U(p(X) - X + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t)) - C_T(\pi))\big]$, without knowing the price $p(X)$ in advanced.

Remark 2.3.4. If we are in the Black-Scholes market model, the exponential utility indifference price reduces to the Black-Scholes price, seeing the proof in [BGTW18]. But the optimal hedging strategy which depends on the risk-aversion parameter λ is not same as the Black-Scholes delta.

Chapter 3

A Brief Overview of Neural Network

3.1 Artificial Neural Network

Inspired by biology, Artificial neural networks (ANNs) are architectures which are designed to mimic the way the human brain works by McCulloch and Pitts in 1943 [MP43]. Artificial Neural Network (ANN), sometimes referred as multilayer perceptron (MLP), have become an increasingly popular in the field of machine learning.

Artificial neural network consists of multiple layers of neurons (Figure 3.2). The first and the last layers are called the input layer and output layer respectively. Hidden layers are the layers between input and output layers. Each neuron in the hidden layers takes input of weighted sum of all inputs element and a bias term which is not included in the weighted sum, then output a real number through the activation function (Figure 3.1). Note that the commonly used activation functions are shown in Table 3.1. This outputs with another bias are passed to all neurons in the next hidden layer. The process continues similarly through all hidden layers after which the output of the whole network is produced.

Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
ReLU	$R(x) = max(0, x)$
hyperbolic tangent	$f(x) = \frac{2}{1+e^{-x}} - 1$

Table 3.1: Activation functions

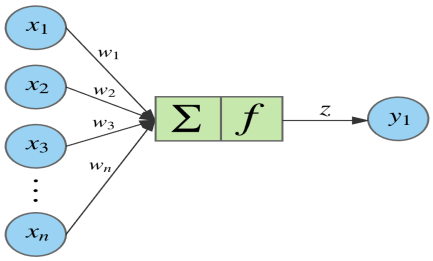

Figure 3.1: Functions in the Neuron

I have now shown the basic structure of an ANN, but it is still unclear how the network actually learns a representation (i.e. a function) from its input data. This network training process consists of two steps, forward pass and back-propagation algorithm.

3.2 The Forward Pass

The training of the ANN is a process where the weights and biases of the network are updated to minimise the loss function during iterations. In each iteration, the inputs are passed through

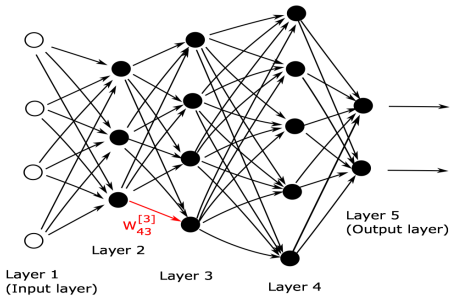

Figure 3.2: A network with five layers. The edge corresponding to the weight $w_{43}^{[3]}$ is highlighted. The output from neuron number 3 at layer 2 is weighted by the factor $w_{43}^{[3]}$ when it is fed into neuron number 4 at layer 3, an example from [HH18]

.

the network with parameter values of the network to obtain the output values. This phase of the process is called the forward pass.

The precise formulation for forward pass can be written as follows. Note that here I use the notations and an example introduced in [HH18] to illustrate ANNs. Suppose that the network has L layers, with layers 1 and L being the input and output layers, respectively. Suppose that layer l, for $l = 1, 2, 3, ..., L$, contains n_l neurons. So n_1 is the dimension of the input data. Overall, the network maps from $\mathbb{R}^{n_1}$ to $\mathbb{R}^{n_L}$. We use $W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ to denote the matrix of weights at layer l. More precisely, $w_{jk}^{[l]}$ is the weight that neuron j at layer l applies to the output from neuron k at layer $l-1$. Similarly, $b[l] \in \mathbb{R}^{n_l}$ is the vector of biases for layer l, so neuron j at layer l uses the bias $b_j^{[l]}$.

In Figure 3.2 I give an example with $L = 5$ layers. Here, $n_1 = 4$, $n_2 = 3$, $n_3 = 4$, $n_4 = 5$, $n_5 = 2$, so $W^{[2]} \in \mathbb{R}^{3\times4}$, $W^{[3]} \in \mathbb{R}^{4\times3}$, $W^{[4]} \in \mathbb{R}^{5\times4}$, $W^{[5]} \in \mathbb{R}^{2\times5}$, $b^{[2]} \in \mathbb{R}^3$, $b^{[3]} \in \mathbb{R}^4$, $b^{[4]} \in \mathbb{R}^5$, $b^{[5]} \in \mathbb{R}^2$.

Given an input $x \in \mathbb{R}^{n_1}$, we may then summarise the forward pass process by letting $a_j^{[l]}$ denote the output from neuron j at layer l and σ denote our choice of activation function. Then, we have

$$a^{[1]} = x \in \mathbb{R}^{n_1} \tag{3.1}$$

$$a^{[l]} = \sigma\big(W^{[l]}a^{[l-1]} + b^{[l]}\big) \in \mathbb{R}^{n_l}, \quad for \quad l = 2, 3, ..., L \tag{3.2}$$

It should be clear that (3.1) and (3.2) amount to an algorithm for feeding the input forward through the network in order to produce an output $a^{[L]} \in \mathbb{R}^{n_L}$.

Now suppose we have N data points, $\{x^i\}_{i=1}^N$ in $\mathbb{R}^{n_1}$, for which there are given target outputs $\{y(x^i)\}_{i=1}^N$ in $\mathbb{R}^{n_L}$. The loss function that we wish to minimise has the form

$$Loss = \frac{1}{N}\sum_{i=1}^{N} L(y(x^{\{i\}}), a^{[L]}(x^{\{i\}})) \tag{3.3}$$

Where L is a function to output the loss on a single data point. There are wide range of loss functions for different purposes, here I introduce some commonly used loss functions :

$$\text{Mean Squared Error :} \quad Loss = \frac{1}{N}\sum_{i=1}^{N}(y(x^{\{i\}}) - a^{[L]}(x^{\{i\}}))^2 \tag{3.4}$$

$$\text{Mean Absolute Error :} \quad Loss = \frac{1}{N}\sum_{i=1}^{N}|y(x^{\{i\}}) - a^{[L]}(x^{\{i\}})| \tag{3.5}$$

$$\text{Negative Logarithmic Likelihood :} \quad Loss = -\frac{1}{N}\sum_{i=1}^{N}\log(a^{[L]}(x^{\{i\}})) \tag{3.6}$$

MSE and MAE are commonly used as a performance measure in regression problems. By comparison, negative logarithmic likelihood is used to measure the accuracy of a classifier. In this project,

MSE and exponential utility function are implemented as loss function for different models.

Remark 3.2.1. Note that the Loss is a function of all the weights and biases.

3.3 Gradient Descent

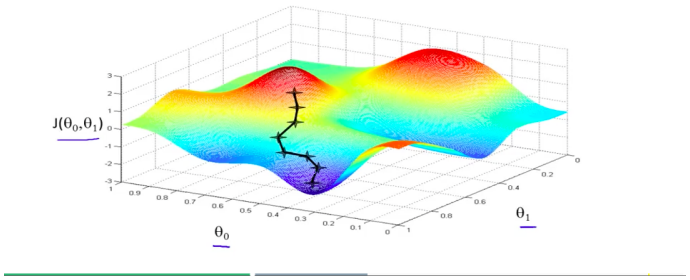

Figure 3.3: A 2D example of gradient descent from [Ng16]

As mentioned in the last section, the output error is calculated with some loss function after completing the forward pass. For simplicity, in this section we may use a single vector p to represent all the weights and biases parameters. In general, we can write the Loss function in (3.3) as $Loss(p)$ to emphasise its dependence on the parameters, such as $Loss(p) : \mathbb{R}^s \to \mathbb{R}$ for $p \in \mathbb{R}^s$.

The objective is to minimise the loss function, and this can be done by gradient descent algorithm. Gradient descent suggests that we should update the parameters p iteratively by choosing the change of parameter Δp in the direction of $-\nabla Loss(p)$, as showed in the Figure 3.3. This leads to the update

$$p \to p - \eta \nabla Loss(p) \tag{3.7}$$

Here η is a hyperparameter called the *learning rate*.

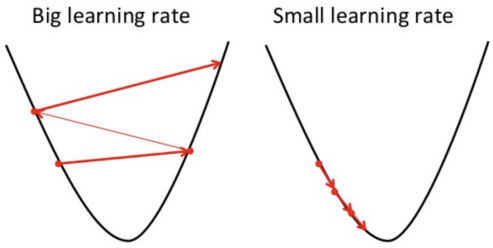

Figure 3.4: The affect of learning rate in gradient descent

Remark 3.3.1. The Figure 3.4 illustrates that the choice of learning rate can affect the network training significantly. The value of loss function may get stuck in a local minimal rather than a global minimal or may take too long to descend if the learning rate is too small. A large learning rate may cause the algorithm overshoot and may diverge eventually.

It is worthwhile to point out that, when we are using a large number of parameters and training data, computing the gradient vector $\nabla Loss(p)$ by gradient descent method (3.4) can be computationally expensive. For example, the MSE loss function (3.4) involves all training data. Then the partial derivative $\nabla Loss(p)$ is the sum of all partial derivatives at each training data. Recall quadratic loss:

$$L_{x^{\{i\}}} = (y(x^{\{i\}}) - a^{[L]}(x^{\{i\}}))^2 \tag{3.8}$$

Then, from (3.3),

$$\nabla Loss(p) = \frac{2}{N}\sum_{i=1}^{N} \nabla L_{x^{\{i\}}}(p) \tag{3.9}$$

An alternative way which is much computationally cheaper is to use the gradient at a random chosen training point instead of computing each individual partial derivative. Such optimisation algorithm is called the *stochastic gradient method*. Such that we can replace (3.7) by:

$$p \to p - \eta \nabla L_{x^{\{i\}}}(p) \tag{3.10}$$

It is also natural to consider a compromise between stochastic gradient descent and full gradient descent where we use a small sample average. Here we introduce *mini-batch stochastic gradient descent* which for some $m \ll N$ follows the steps:

1. Choose m integers, $c_1, c_2, ..., c_m$, uniformly at random from $\{1, 2, 3, ..., N\}$.

2. Update

$$p \to p - \eta \frac{1}{m}\sum_{i=1}^{m} \nabla L_{x^{\{i\}}}(p) \tag{3.11}$$

In addition, assuming $N = Cm$ for some C, we can split the training set randomly into C distinct mini-batches and cycle through, referred to as completing an *epoch*. Combining *mini-batch gradient descent* and *epoch* method can not only make the gradient computation more efficient, but it also avoids getting stuck in the local minimal, due to the randomness of gradient. The book [GBC16] gives a more concise discussion about the effect of hyperparameters, such as *learning rate* and *batch size*, on the training performance.

3.3.1 Adam Gradient Descent

Adam[KB17] is an optimization algorithm that can used instead of the classical stochastic gradient descent procedure to update network weights iterative based on the training data. The stochastic gradient descent we introduced previously uses a single constant learning rate for all parameter updates. Adam essentially computes individual adaptive learning rates for different parameters from the estimations of first and second moments of the gradients.

Adam combines the advantages of Adaptive Gradient Algorithm(AdaGrad)[DHS11] and Root Mean Square Propagation[TH12]. Specifically, AdaGrad maintains a per-parameter learning rate that improves performance on problems with sparse gradients. RMSProp use a adaptive per-parameter learning rates based on the mean of recent magnitudes of the gradients for the parameter (e.g. how quickly it is changing). This means the RMSProp algorithm does well on online and non-stationary problems (e.g. noisy). Adam realises the benefits of both AdaGrad and RMSProp.

In the original paper [KB17], the results has shown that Adam compare favourably to other stochastic optimisation methods and can achieve good results faster. This make Adam very popular as an optimisation method in the field of deep learning. Throughout my project, I use Adam gradient descent as the optimisation method. The Adam algorithm code and parameters setting can be find in the original paper [KB17].

3.4 The Back-propagation Algorithm

In order to train an artificial neural network, the parameters are updated through *back-propagation* with the stochastic gradient descent method. To update each weight and bias parameter, $w_{jk}^{[l]}$ and $b_j^{[l]}$, we need to find the partial derivative respect to each individual parameter. Note that, for a fixed data point $x^{\{i\}}$, the loss function $L_{x^{\{i\}}}$ is a function of all weights and biases.

Recall the loss function in (3.3), the dependence of Loss on weights and biases arises only through the output from the artificial neural network $a^{[L]}$.

we will introduce two useful sets of variables to express the the partial derivatives. First we write

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]} \in \mathbb{R}^{n_l} \quad for \quad l = 2, 3, ..., L. \tag{3.12}$$

We refer to $z_j^{[l]}$ as the weighted input for neuron j at layer l, so it is the number before we apply activation function σ in each neuron. Therefore

$$a^{[l]} = \sigma\big(z^{[l]}\big), \quad for \quad l = 2, 3, ..., L \tag{3.13}$$

Second, we let $\delta^{[l]} \in \mathbb{R}^{n_l}$ be defined by

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}, \qquad for \quad 1 \leq j \leq n_l \quad and \quad 2 \leq l \leq L. \tag{3.14}$$

which is often called the *error* in the neuron j at layer l.

The following lemmas are used in back-propagation algorithm.

Lemma 3.4.[HH18]

$$\delta^{[L]} = \sigma'(z^{[L]}) \circ (a^L - y) \tag{3.15}$$

$$\delta^{[l]} = \sigma'(z^{[L]}) \circ (W^{[l+1]})^T \delta^{[l+1]}, \qquad for \quad 2 \leq l \leq L-1, \tag{3.16}$$

$$\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]}, \qquad for \quad 2 \leq l \leq L, \tag{3.17}$$

$$\frac{\partial C}{\partial w_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]} \qquad for \quad 2 \leq l \leq L. \tag{3.18}$$

Where σ is the activation function, $\circ$ represents component-wise product of two vectors such that $(x \circ y)_i = x_i y_i$. The proof of this lemma is omitted here, it can be found in the chapter 5 from [HH18].

Recall that, the variables $a^{[1]}$,$z^{[2]}$,$a^{[2]}$,$z^{[3]}$,...,$a^{[L]}$ are computed in order during the forward pass through network. So $\delta^{[L]}$ can be computed immediately according to (3.14) after forward pass. Then, from equation (3.15), $\delta^{[L-1]}$,$\delta^{[L-2]}$,..., $\delta^{[1]}$ may be computed in a backward pass. We then can compute the partial derivative of parameters in each layer from equation (3.16) and (3.17). Such a way to compute gradient is called back-propagation.

3.5 Neural Networks for Option Hedging and Pricing

There are two main advantages of neural network that make it an important architecture for option hedging and pricing. Firstly, neural network is robust for non-linear modelling of data because of the network structure and activation function. This means that neural network not only allows us to replicate the Black-Scholes hedging and pricing (which is a non-linear function as shown in (2.9) and (2.10)) in a complete market model, but it can also include market frictions such as transaction cost, liquidity constraints, market impact etc, as inputs to find the hedging strategy and price in a incomplete market model. Secondly, neural network approach allows for high-dimensional data with low sample size. This property of the neural network will not be illustrate in this thesis because we only consider hedging with a stock and a risk-less bond. However, this is very useful when we consider hedging with multiple hedging instruments. And the potential of the deep learning algorithm for high-dimensional problems is demonstrated in [BGTW18].

There are several literatures in option pricing with neural networks. One of the first study in this field is [HLP94] in 1994 and they illustrated that deep learning algorithm can outperform the Black-Scholes pricing with real market data. Then several studies revisited the neural network pricing with a "deeper and wider" network architecture. However, hedging strategies by deep learning has just been studied recently in [BGTW18]. In this thesis, I will revisit some ideas in [BGTW18] with different neural network structures and experiment in various parameter settings.

Chapter 4

Data and Model Specification

In this chapter, I will discuss data simulations, neural network structures and performance measures of three different neural network models which are used to find the hedging strategies under different market assumptions:

- **Model 1**: Neural network model with known volatility parameter in an idealised, frictionless complete market model
- **Model 2**: Neural network model with unknown volatility parameter in an idealised, frictionless complete market model
- **Model 3**: Neural network model with known volatility parameter in an incomplete market model (only with trading costs in our case)

Data, neural network structures and performance measures are set differently for each case in order to meet the model requirements and financial interests. For example, the second model uses different data and neural network architecture in order to estimate the volatility parameter with historical data. For performance measures, we will compare the model delta hedging strategies with the Black-Scholes delta strategies for first two cases. By contrast, we are more interested in the pay-offs at the time of maturity with different risk-aversion parameters in an incomplete market model.

4.1 Data

Throughout the project, for simplicity, we set the time of maturity of the European call option is 1, discrete time-step is 0.01, so there are 101 time points at $t = 0, 0.01, 0.02, ..., 1$. Also the simulation of stock prices is according to the Black-Scholes formula such that the stock prices follow geometric Brownian motion, recall equation(2.8):

$$S_t = S_0 e^{(\mu - \frac{\sigma^2}{2})t + \sigma W_t}$$

Where W_t is the standard Brownian motion. Brownian motion path can be simulated by using the property such that its increments have standard normal distribution with 0 mean and 0.01 variance, such that $W_{t+0.01} - W_t \sim \mathbb{N}(0, 0.01)$ for $t = 0, 0.01, ..., 0.99$. Hence, for each Brownian path, 100 increments of Brownian motion are generated by random normal number generators and summed cumulatively to obtain the Brownian motion path. Then S_t are simulated with different parameter setting.

For **Model 1** and **Model 3**, I simulated 100000 stock paths $S_0, S_{0.01}, S_{0.02}, ..., S_1$ with different parameter setting.

For **Model 2**, at first, I randomly generated 100000 volatility parameters σ. Then I simulated 100000 historical stock price paths and 100000 current stock price paths with generated σ samples according to the geometric Brownian motion. Same as other models, current stock price paths have 101 time points from $t = 0$ to $t = 1$. Historical stock price paths has 301 time points from $t = 0$ to $t = 3$. I simulated historical stock price path longer is because historical data are normally sufficient in practice. The returns of the historical stock price paths were then computed. The returns of historical stock prices were used instead of historical stock prices to implicitly estimate the σ because it can reduce the complexity of estimation.

4.2 Neural Network Structure

4.2.1 Neural network with known σ in the Black-Scholes model

Recall that hedging error of a European call option , equation (2.7), in discrete time:

$$Hedgingerror = -X + V_0^{\pi} + \Sigma_{t=0}^{T-1} \pi_t^1 (S_{t+1} - S_t)$$

We known that the hedging error is approximated to zero if the initial value is the Black-Scholes call option price and the portfolio of stock is the Black-Scholes delta, such that $V_0^{\pi} = C_{BS}$ and $\pi = \Delta_{BS}$. In this model, essentially I aim to find whether the neural network can represent the functionality of Black-Scholes delta if we are given the initial cash C_{BS}.

Recall equations (2.9) and (2.10), the Black-Scholes delta Δ_t is a function of S_t and t for all t when the time of maturity T, strike price K, interest rate r and volatility σ are fixed known. Then, we expect the neural network can represent the a function $f(S_t, t)$, for $t = 0, 0.01, ..., 0.99$, which minimises the hedging error. We therefore can naturally define a quadratic loss function:

$$loss := (C_{BS} + \Sigma_{i=1}^{100} f(\frac{i-1}{100}, S_{\frac{i-1}{100}})(S_{\frac{i}{100}} - S_{\frac{i-1}{100}}) - (S_1 - K)^+)^2 \tag{4.1}$$

Note that the loss function above take the outputs from the neural network at all $t = 0, 0.01..., 0.99$. Essentially, we need to define a shared weight neural network architecture which contains 100 identical neural networks so that outputs at all time points can be passed to the loss function simultaneously. The shared weight neural network architecture are showed as below:

$$\left.\begin{array}{l} Network\ 1 : (0, S_0) \to f(0, S_0) \\ Network\ 2 : (\frac{1}{100}, S_{\frac{1}{100}}) \to f(\frac{1}{100}, S_{\frac{1}{100}}) \\ \vdots \\ \vdots \\ \vdots \\ Network\ 100 : (\frac{99}{100}, S_{\frac{99}{100}}) \to f(\frac{99}{100}, S_{\frac{99}{100}}) \end{array}\right\} \to (C_{BS} + \Sigma_{i=1}^{100} f(\frac{i-1}{100}, S_{\frac{i-1}{100}})(S_{\frac{i}{100}} - S_{\frac{i-1}{100}}) - (S_1 - K)^+)^2$$

Figure 4.1: Shared weight neural network architecture with known σ

Remark 4.2.1. In shared weight neural network architecture, we only defined one neural network and the network is called 100 times to pass the inputs (S_t, t) at all time points simultaneously. So the back-propagation algorithm is applied to a single neural network rather than 100 different neural network.

The neural network (Figure 4.2) takes 2 inputs (stock price and time) and output 1 number (model delta). It has 3 hidden layers with 100 neurons in each layer, and all neurons in the first 2 hidden layers use **ReLu** (Table 3.1) as activation function. The neurons in last hidden layer use **Sigmoid** (Table 3.1) as activation function. Note that **Sigmoid** outputs a real number ranging from 0 to 1. Here I used **Sigmoid** activation function to restrict the output of network bounded by 0 and 1. Because we know that $\Delta \in (0, 1)$ in theory and practice. In theory we know that the $\Delta = \frac{\partial C}{\partial S} = N(d1)$ from (2.10) and normal cumulative function is always in the range of 0 and 1. In practice, hedge ratio large than 1 means that we essentially borrow cash to invest in extra shares of stock, such movement is unnecessary because it is sufficient to hedge the call option with hedge ratio equals to 1 when stock price is so high that the buyer is guaranteed to fulfill the contract. A similar argument can be applied when the hedge ratio is below than 0.

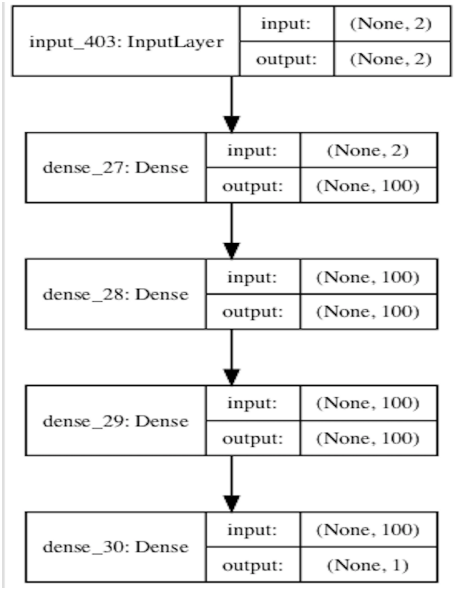

Figure 4.2: Neural network

4.2.2 Neural network with unknow σ in the Black-Scholes model

Based on the previous neural network model, I further developed a model with unknown σ which can find the Black-Scholes trading strategies base on the historical returns and current stock prices. Through experiments, I found that a single network which takes both historical data and current stock prices induces a extremely complex function to learn. So I separate the network into two parts, the first part of network takes the historical returns of a stock to estimate the volatility parameter σ implicitly, and the σ was then used in the second part of network to find the trading strategies. Same as previous model, I used shared weights neural networks architecture to meet the requirement of loss function (4.1). The network architecture is shown as below.

$$\text{Pre-network} : (R_{hist}) \to \hat{\sigma} \to \left\{ \begin{array}{l} Network\ 1 : (0, S_0, \hat{\sigma}) \to f(0, S_0, \hat{\sigma}) \\ Network\ 2 : (\frac{1}{100}, S_0, \hat{\sigma}) \to f(\frac{1}{100}, S_{\frac{1}{100}}, \hat{\sigma}) \\ \vdots \\ \vdots \\ \vdots \\ Network\ 100 : (\frac{99}{100}, S_0, \hat{\sigma}) \to f(\frac{99}{100}, S_{\frac{99}{100}}, \hat{\sigma}) \end{array} \right\} \to loss$$

Figure 4.3: Shared weight neural network architecture with unknown σ

It is worthwhile to point out that the $\hat{\sigma}$ serves as a node to connect the two parts of the network and it is unknown for us throughout the network training.

And because of the additional input vector R_{hist}, the loss function in this model is slightly different to (4.1):

$$loss := (C_{BS}(\sigma) + \Sigma_{i=1}^{100} f(R_{hist}, \frac{i-1}{100}, S_{\frac{i-1}{100}})(S_{\frac{i}{100}} - S_{\frac{i-1}{100}}) - (S_1 - K)^+)^2 \tag{4.2}$$

$f(R_{hist}, \frac{i-1}{100}, S_{\frac{i-1}{100}})$ represents the hedging strategy by neural network, which essentially is a function with inputs: all historical returns, current stock price and time. Note that the Black-Scholes call option price is dependent on σ, so $C_{BS}(\sigma)$ was calculated with Black-Scholes formula with simulated σs. This means my model actually used the parameter σ. But because we prioritised

on testing the performance of the network structure, we ignore this drawback of the model. In order to make the σ fully unknown in the model, recall **Remark 2.3.4.**, we can build another neural network to calculate the Black-Scholes call option price with exponential utility indifference pricing and use it as a input in the current model.

The first part of network takes inputs of 300 historical returns and output 1 number($\hat{\sigma}$) . It has 3 hidden layer with 2000 neurons, 1000 neurons and 200 neurons respectively. All the neurons use **ReLu** as activation function. The second part of the network is same to the network in previous model (Figure 4.2) but with 3 inputs ($\hat{\sigma}$, current stock price and time). Overall, the neural network takes 4 inputs: a vector of historical returns R_{hist}, predetermined call options prices $C_{BS}(\sigma)$, current stock price S_t and time t.

4.2.3 Neural network with known σ in an incomplete market model

In an incomplete market model with friction, Section 2.4.1 introduce how exponential utility function can be used as a optimality criterion to solve the hedging problem. Recall **Remark 2.3.3**, we can find the optimal strategy π which maximise:

$$\mathbb{E}\big[U(-X + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t)) - C_T(\pi))\big]$$

And by the definition of exponential utility, $U(x) := -exp(-\lambda x)$,we know that

$$\sup_{\pi} \mathbb{E}\big[U(-X+\Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1}-S_t))-C_T(\pi))\big] = \inf_{\pi} \mathbb{E}\big[exp(-\lambda[-X+\Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1}-S_t))-C_T(\pi)])\big]$$

Then the optimal strategy π can be obtain by a new optimal criterion:

$$J(\pi) = \inf_{\pi} \mathbb{E}\big[exp(-\lambda[-X + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t)) - C_T(\pi)])\big] \tag{4.3}$$

In a neural network, this optimal criterion can be implemented naturally with mini-batch gradient descent under a loss function defined as:

$$Loss = \exp(-\lambda[-X + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t)) - C_T(\pi)]) \tag{4.4}$$

By using mini-batch stochastic gradient descent, the expectation in (4.3) (which is in fact a weighted sum over all elements of the finite, but potentially very large sample) is replaced by an expectation over a randomly (uniformly) chosen subset of sample of size $N_{batch} \ll N$ (size of sample).

With proportional cost function, the loss of my neural network in an incomplete model then can be defined by:

$$loss = \exp\left(\lambda\big[-(S_1-K)^+ + \sum_{i=1}^{100} f(\frac{i-1}{100}, S_{\frac{i-1}{100}})(S_{\frac{i}{100}} - S_{\frac{i-1}{100}}) - c\sum_{i=0}^{100} |f(\frac{i}{100}, S_{\frac{i}{100}}) - f(\frac{i-1}{100}, S_{\frac{i-1}{100}})| S_{\frac{i}{100}}\big]\right) \tag{4.5}$$

Where $f(\frac{-1}{100}, S_{\frac{-1}{100}}) = f(1, S_1) = 0$.

This neural network model uses exact the same neural network architecture in (4.2.1), as shown in Figure 4.1 and Figure 4.2.

4.3 Performance Measures

Under idealised, frictionless and complete market. The performance of the neural network will be compared to the basic Black-Scholes model. We can therefore visualise the difference between delta hedging of neural network and delta hedging of the Black-Scholes.

Also I will follow the methodology of [HLP94] and measure the delta hedging performance of the neural network with a tracking error. We know that the Black-Scholes model assumes continuous delta hedging, which is not possible in practice and this makes the hedging error different from zero at the time of maturity. This is called the delta-hedging tracking error. There will always be some hedging error because of the discrete delta hedging, as we discussed in Section 2. So the hedging performance by deep learning algorithm will be evaluated by comparing with the Black-Scholes delta.

Recall **2.3.4**, the exponential utility indifference price reduces to Black-Scholes price in the Black-Scholes model. Hence we can measure the performance of my model by comparing the exponential utility indifference price with the Black-Scholes price when there is no transaction cost applied.

Once we know the exponential utility indifference price matches the Black-Scholes price, then we can observe how the hedging strategy varies according to risk-aversion parameter. Also we are more interested in the distribution of Profit&Loss at the time of maturity with different risk-aversion parameters. In a incomplete market with transaction cost, call option price are affected by the size of transaction cost. In order to show the neural network model is feasible in an incomplete market model, I will compare if the changing of call option price for different transaction costs coincides with some analytical results.

Chapter 5

Numerical Experiments and Results

After introducing the model specification and performance measures in Chapter 4, I now turn to numerical experiments to illustrate the feasibility of the neural network approach in several scenarios. In the experiments, I consider the out of sample performance of neural network with different parameter settings (e.g. different values of volatility σ and strike prices K).

Throughout the project, the deep learning algorithms has been implemented in Python, using Keras [CO15] which uses TensorFlow and Theano as back ends. The network parameters are initialised according to Keras default setting where weights are initialised randomly (drawn from uniformly distribution) and biases are initialised as 0. For network training, the Adam algorithm are used with learning rate of 0.0001, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and a batch size of 100 has been used. The code for the numerical implementations can be found in my GitHub repository: https://github.com/lhang39/Hedging-by-Deep-Learning.

To demonstrate the feasibility of the neural network approach, this chapter will be devoted to examining the following questions:

- Section 5.1: How does the delta hedging of neural network (using quadratic loss function) compare to the delta hedging of the Black-Scholes model without transaction cost? What are the effects of the volatility and strike prices on the neural network hedging?
- Section 5.2: With unknown volatility, how well can the special designed neural network architecture learn the delta hedging of the Black-Scholes. Does the sample of volatility parameters affect the learning of neural network? How well does the historical data estimate the volatility parameter (essentially we compare the neural network using historical data to the neural network using volatility samples directly)?
- Section 5.3: How does the delta hedging of neural network (using exponential utility function) compare to the benchmark in the Black-Scholes model? What is the effect of risk-aversion parameter on the distribution of Profite&Loss with optimal hedging strategy?
- Section 5.4: What is the effect of proportional transaction costs on the exponential utility indifference price? What is the effect of transaction costs on the distribution of Profite&Loss with optimal hedging strategy?

5.1 Neural network hedging with fixed σ in the Black-Scholes model

I will illustrate the performance of the neural network approach in the Black-Scholes model with different σs and strike prices K. Notice that the drift parameter μ is out of my interests in this project for 2 reasons:

- The drift parameter μ vanishes in the Black-Scholes formula , see (2.9). Some concise discussions about this topic can be seen in [BFK12].
- The drift parameter μ is generally very small in real market data.

For simplification, I simulated the stock price path with $\mu = 0$ in a geometric Brownian motion such that $S_t = S_0 \exp(-\frac{\sigma^2}{2}t + \sigma W_t)$, where I choose $S_0 = 10$.

The performance of delta hedging of neural network model is firstly discussed with ATM (at the money), OTM (out the money) and ITM (in the money) conditions with fixed $\sigma = 0.1$. The strike prices are then set $K = 10$, 12 and 8 corresponding to ATM, OTM and ITM respectively. The whole 100,000 stock prices paths was split with ratio 9:1 into training data and test data. The hedging errors of strategies were calculated based on the out-of-sample test data.

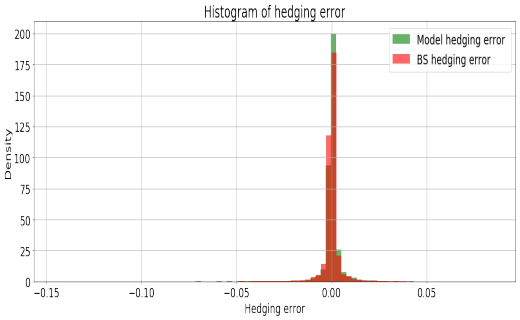

Figure 5.1: Histogram of hedging errors when K=8, σ=0.1

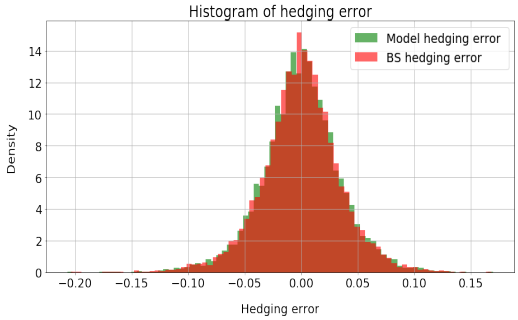

Figure 5.2: Histogram of hedging errors when K=10, σ=0.1

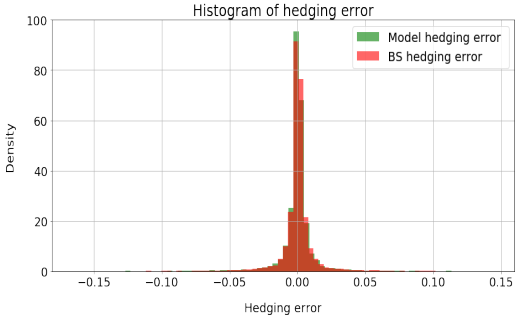

Figure 5.3: Histogram of hedging errors when K=12, σ=0.1

	BS 5% Quantile	BS 95% Quantile	Model 5% Quantile	Model 95% Quantile
K=8	-0.0052	0.0053	-0.0047	0.0058
K=10	-0.057	0.056	-0.056	0.056
K=12	-0.014	0.014	-0.015	0.014

Table 5.1: Hedging error quantiles on out-of-sample test data with fixed $\sigma = 0.1$

As shown from the histogram of hedging errors and table of hedging error quantiles. It is clear that both Black-Scholes discrete delta hedging and neural network delta hedging do better at ITM condition. More importantly, these results illustrate that the hedging errors of neural network model coincide with the hedging errors of the Black-Scholes in all three types of European call options when $\sigma = 0.1$. Hence we can assume that the hedging errors of neural network model come from the discrete setting rather than the model error. This means that our neural network model may nearly perfect-hedge a European call option with a much smaller time-step setting when $\sigma = 0.1$.

Furthermore, I have done a similar statistical analysis on the hedging errors with different values of volatility parameter σ, where volatility parameters are chosen at values $\sigma = 0.01, 0.1, 0.5$ and strike price is fixed at $K = 10$.

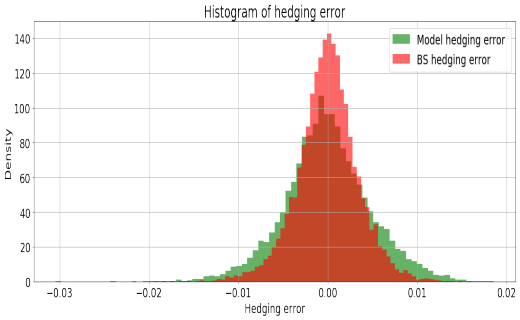

Figure 5.4: Histogram of Hedging error when K=10, σ=0.01

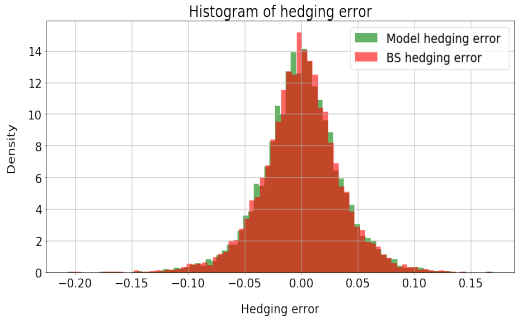

Figure 5.5: Histogram of Hedging error when K=10, σ=0.1

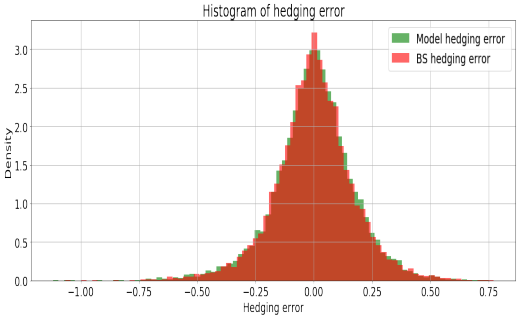

Figure 5.6: Histogram of Hedging error when K=10, σ=0.5

	BS 5% Quantile	BS 95% Quantile	Model 5% Quantile	Model 95% Quantile
σ=0.01	-0.0057	0.0055	-0.0079	0.0082
σ=0.1	-0.057	0.056	-0.056	0.056
σ=0.5	-0.28	0.27	-0.29	0.27

Table 5.2: Hedging error quantile on out-of-sample test data with fixed strike price$K = 10$

Two main features can be drawn from the results shown above are:

- For both Black-shcoles hedging and model hedging, we observe that the hedging error quantiles increase as the value of σ increases. This is mainly due to the discrete friction. Recall the hedging error formula:$-X + V_0^\pi + \Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t)$. Large σ induces a large stock change increment $S_{t+1} - S_t$, so the difference between $\Sigma_{t=0}^{T-1}\pi_t^1(S_{t+1} - S_t)$ and $\int_{t=0}^{T}\pi_t^1 dS_t$(it is a integral in continuous case which has zero hedging error) is larger.

- For large σ, the neural network delta hedging perform as well as the Black-Shcoles delta hedging. In a contrast, one can see that the model hedging error is much larger than the Black-Scholes hedging error when $\sigma = 0.01$ and it took more computational effort than large σ to reduce the hedging error in experiments. This is because the variability of the dataset is very low with very small σ. Hence the deep learning algorithm need to learn the functionality of hedging strategy from the data with little difference which is more computational expensive and may cause some model error. Nevertheless, this special case is not concerned because the volatility is normally very high in the real market.

We know that with the fixed volatility σ, time of maturity T, strike price K, the delta strategies is a function of stock price and time (Recall (2.9) and (2.10)). To illustrate the neural network strategies is able to approximate well the Black-Scholes strategies, I also visualise the neural network strategies and Black-Scholes strategies as a function of stock prices and time for $\sigma = 0.1, K = 10$.

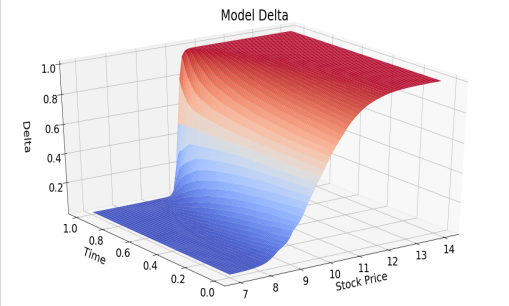

Figure 5.7: Neural network delta approximates as a function of (S_t,t)

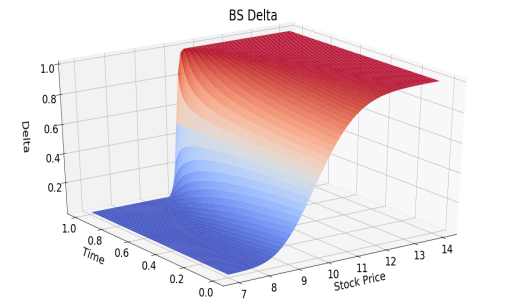

Figure 5.8: Black-Scholes delta approximates as a function of (S_t,t)

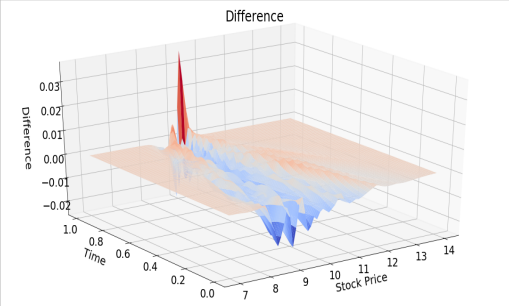

Figure 5.9: Difference between neural network delta and the Black-Scholes delta

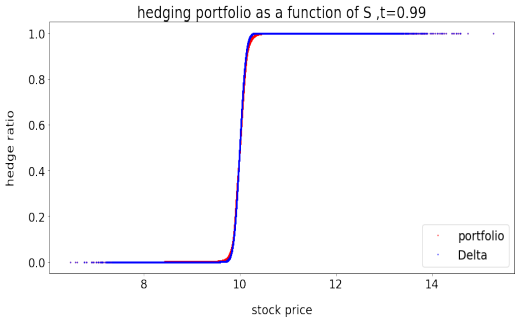

Figure 5.10: neural network delta and the Black-Scholes delta as a function of $S_{0.99}$ at $t = 0.99$

From the Figures (5.7), (5.8) and (5.9), the neural network learns the Black-Scholes delta strategy very well with the difference in the range of 0.04. However, one can see that the difference is quite large at the beginning of the call option, this is because most of the stock prices scatter near $S_0 = 10$ at the beginning so that the algorithm fails to find the delta hedging at the stock prices which have very thin data. Also it is clear that there is a large spike of the difference located at time close to the maturity and stock price at strike price $S_T = K = 10$, this large difference may because the shape of the function of delta hedging is so steep at the time of maturity (shown in Figure (5.10)) that the data around strike price are too thin to accurately explore the feature of the function.

With convincing delta hedging performance, my results was consistent with previous literature [AKS96], which shows that the network deltas were consistent with the Black-Scholes. However [AKS96] prioritise on option pricing with neural network, so the network model were used to output the call option price C_t at all t and delta hedging ratio was then calculated by $\frac{\partial C_t}{\partial S_t}$. Recent paper[Sta17] revisited this method, however it failed to reproduce a consistent Black-Scholes delta hedging strategy with network. My neural network model prioritise on the hedging strategy which produce a consistent results with the Black-Scholes delta hedging in wide range of parameter setting.

5.2 Neural network hedging with unknown σ in the Black-Scholes model

As introduced in section 4.2.2, this neural network model is to find the delta strategy with historical returns, current stock prices and time with special designed neural network architecture. Such neural network application has not been studied previously, so the neural network delta strategies was not only compared to the Black-Scholes hedging strategies, but also compared to the hedging strategies from another neural network model which uses σ samples as inputs. Here, I let Δ_{BS}, Δ_H and Δ_S denote the delta hedging strategies of the Black-Scholes, the neural network using historical data and the neural network using σ samples respectively. Then $\epsilon_{BS}, \epsilon_H, \epsilon_S$ denote

the hedging error from using Δ_{BS}, Δ_H and Δ_S as hedging strategies respectively.

In this section, we prioritise our research on the performance of neural network delta hedging with different range of σ sample space. Ideally, we hope that the neural network delta hedging can match the Black-Scholes hedging for a large range of σ sample space. However, as we seen in the previous section, both the discrete Black-Scholes delta and the neural network delta varies with the values of σ. So the hedging by neural network may suffer from the variation of σ when we use a range of random σ samples instead of fixed σ. Therefore, we choose 4 different ranges of σ sample space, σs were randomly generated within the range $(0, 0.5)$,$(0.1, 0.4)$ and $(0.3, 0.4)$. Then the stock prices path was generated according to geometric Brownian motion with simulated σ samples. And we choose the strike price $K = 10$ and $S_0 = 10$.

As a first example, we consider the performance of my approach with the $\sigma \in (0, 0.5)$.

	Hedging error 5% quantile	Hedging error 95% quantile
Model using historical data	-0.18	0.19
Black-Scholes	-0.14	0.16
Model using sigma sample	-0.18	0.18

Table 5.3: Quantils of hedging errors in different models, using random $\sigma \in (0, 0.5)$

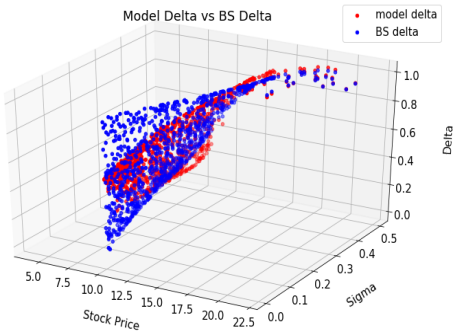

Figure 5.11: Neural network delta and the Black-Scholes delta as a function of (S_t,σ) for $\sigma \in (0, 0.5)$,K=10, t=0.5

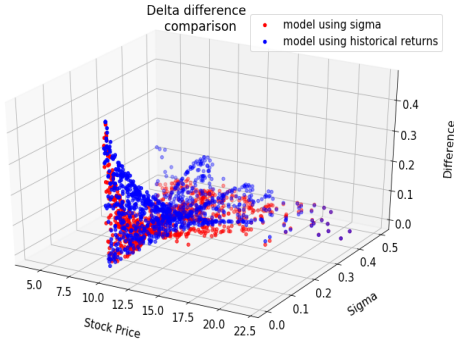

Figure 5.12: The absolute difference between neural network delta using historical data and the BS delta is compared to the absolute difference between the neural network delta using σ sample and the BS delta for $\sigma \in (0, 0.5)$

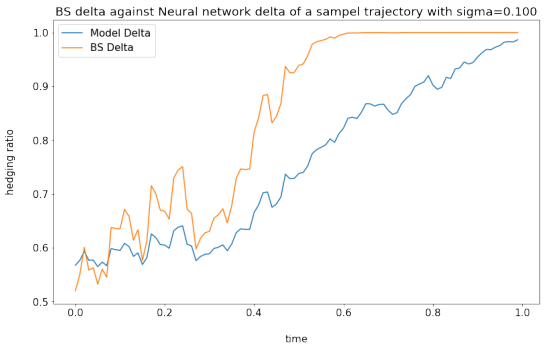

Figure 5.13: Neural network delta and the Black-Scholes delta of a stock price path with $\sigma = 0.001$

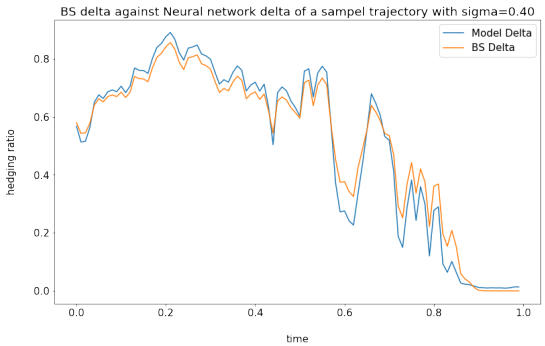

Figure 5.14: Neural network delta and the Black-Scholes delta of a stock price path with $\sigma = 0.50$

Table 5.3 shows that, for the data generated with $\sigma \in (0, 0.5)$, both ϵ_H and ϵ_S are larger than $t\epsilon_{BS}$ but ϵ_H is quite similar to ϵ_S . This means most of the model errors are not incurred from the implicit estimation of σ from historical data. It is also worthwhile to point out that the ϵ_S is

significantly larger than the ϵ_{BS} too, which means some of the model errors is due to additional variability from implicit input parameter σ.

Figure 5.11 plots the hedging strategies as a function of stock price S and volatility σ at a fixed time $t = 0.5$, and it compares the Δ_H and Δ_{BS} for the test data with $\sigma \in (0, 0.5)$. It shows that the hedging strategies from our neural network model is not consistent to the Black-Scholes delta. Figure 5.12 essentially compares $|\Delta_H - \Delta_{BS}|$ to $|\Delta_S - \Delta_{BS}|$, we can see both neural network model can not find the Black-Scholes delta strategy for $\sigma \in (0, 0.5)$. Also both we can visualise that both neural network model perform more inconsistently in the range of small σ values from both figures. To understand this problem occurred in both neural network models, I made a comparison between a sample trajectory with a high σ and another one with a low σ.

As shown in Figure 5.13 and 5.14, the neural network can learn the Black-Scholes hedging strategies on the sample with high σ but not on the sample with low σ. This coincides our finding from Figure 5.12. However, if we look at the hedging errors on these two specific sample trajectory, I find that the hedging error on the sample with $\sigma = 0.001$ is 0.062 but the hedging error on the sample with $\sigma = 0.5$ is -0.26. The question then arises, why a sample with lower hedging error has a more inconsistent hedging strategy from the Black-Scholes delta hedging strategy?

Recall the hedging error formula: $-X + V_0^\pi + \Sigma_{t=0}^{T-1} \pi_t^1 (S_{t+1} - S_t)$. As I pointed out in previous section, the $(S_{t+1} - S_t)$ is small for small σ so that the hedging error stay quite relatively small even the hedging strategy is far from Black-Scholes delta. In the other words, the back propagation algorithm will update the parameter in favour of large σ value data samples because they have large losses in general.

Now, I narrow the range of σ sample to (0.1,0.4) to observe if the model generates a similar result as before.

	Hedging error 5% quantile	Hedging error 95% quantile
Model using historical data	-0.18	0.16
Black-Scholes	-0.14	0.14
Model using sigma sample	-0.16	0.15

Table 5.4: Quantiles of hedging errors in different models, using random $\sigma \in (0.1, 0.4)$

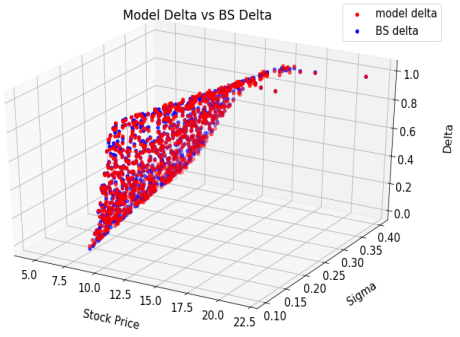

Figure 5.15: Neural network delta and the Black-Scholes delta as a function of (S_t,σ) for $\sigma \in (0.1, 0.4)$,K=10, t=0.5

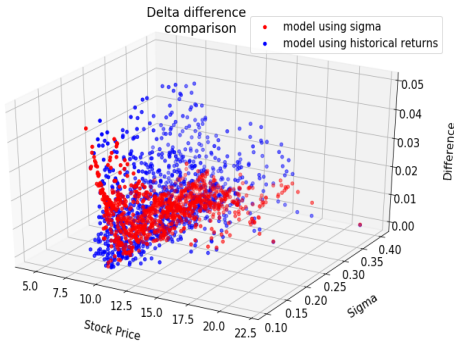

Figure 5.16: The absolute difference between neural network delta using historical data and the BS delta is compared to the absolute difference between the neural network delta using σ sample and the BS delta for $\sigma \in (0.1, 0.4)$

Table 5.4, for $\sigma \in (0.1, 0.4)$, we can observe that the quantiles of ϵ_H decrease a moderate amount and it is much closer to ϵ_S and ϵ_{BS}. Figure 5.15 also illustrates that the neural network model using historical returns is able to learn the Black-Schole delta very well with out-of sample test data. We can visualise that both $|\Delta_H - \Delta_{BS}|$ and $|\Delta_S - \Delta_{BS}|$ improved, which are both lower than 0.05. Surprisingly, the model using historical returns shows the evenly distributed difference across all $\sigma \in (0.1, 0.4)$. However the neural network model using σ sample still produce less consistent delta hedging strategy at low σ value.

	Hedging error 5% quantile	Hedging error 95% quantile
Model using historical data	-0.21	0.20
Black-Scholes	-0.20	0.19
Model using sigma sample	-0.20	0.20

Table 5.5: Quantiles of hedging errors in different models, using random $\sigma \in (0.3, 0.4)$

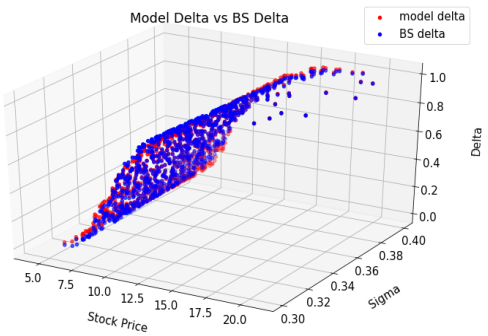

Figure 5.17: Neural network delta and the Black-Scholes delta as a function of (S_t,σ) for $\sigma \in (0.3, 0.4)$,K=10, t=0.5

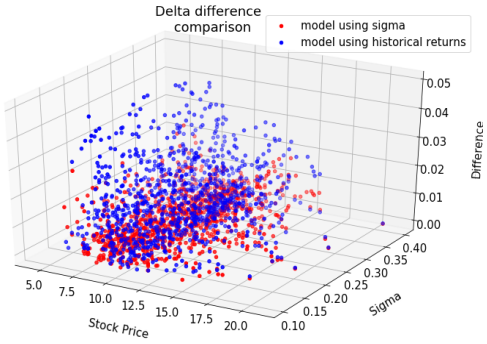

Figure 5.18: The absolute difference between neural network delta using historical data and the BS delta is compared to the absolute difference between the neural network delta using σ sample and the BS delta, for $\sigma \in (0.3, 0.4)$

Table 5.5 shows that ϵ_H , ϵ_S and ϵ_{BS} are almost same for $\sigma \in (0.3, 0.4)$, which means the neural network model friction are significantly reduced as the range of σ samples narrows. Figure 5.17 and 5.18 also illustrate that the delta hedging of my approach is consistent with the Black-Scholes delta for the σ samples are in a smaller range. Furthermore, it is worthwhile to point out that the performance of neural network using historical data is close to the performance of neural network using σ samples under this condition, which essentially means that the special designed neural network is able implicitly estimate σ better in a smaller range of σ samples too. This results in a better overall performance of my approach.

In addition, I will illustrate my deep learning algorithm can find the Black-Scholes deltas of all ITM, ATM and OTM European call options accurately for $\sigma \in (0.3, 0.4)$. Here I choose the strike prices $K = 8$, $K = 10 = S_0$ and $K = 12$ corresponding to ITM, ATM and OTM respectively.

	ITM		ATM		OTM	
	5% quantile	95% quantile	5% quantile	95% quantile	5% quantile	95% quantile
Model using historical data	-0.16	0.17	-0.21	0.20	-0.21	0.23
Black-Scholes	-0.15	0.15	-0.20	0.19	-.20	-0.20
Model using sigma sample	-0.17	0.15	-0.20	0.20	-0.22	0.20

Table 5.6: Quantiles of hedging errors in different models in ITM, ATM and OTM conditions respectively, using random $\sigma \in (0.3, 0.4)$

Table 5.6 demonstrate that, for small range of σ sample, the neural network delta is able to perform, in terms of hedging errors, similar to the Black-Scholes delta for all ITM, ATM and OTM conditions. Figure 5.19 and 5.20 shows that the neural network delta matches with Black-Scholes delta on out-of-sample for both ITM and OTM call options. Combine with Figure 5.17, I illustrated the feasibility of my approach on all three type of European call options.

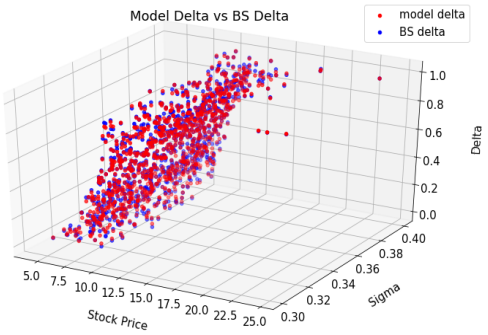

Figure 5.19: Neural network delta and the Black-Scholes delta as a function of (S_t,σ) for $\sigma \in (0.3, 0.4)$,K=8, t=0.5

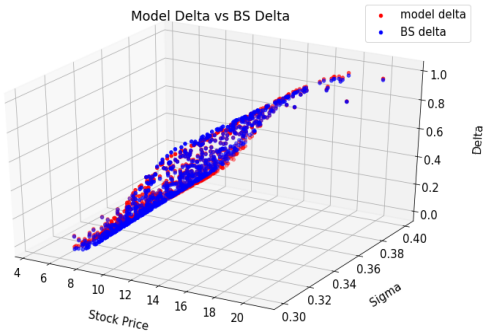

Figure 5.20: Neural network delta and the Black-Scholes delta as a function of (S_t,σ) for $\sigma \in (0.3, 0.4)$,K=12, t=0.5

My results have shown that my neural network architecture is able to learn the Black-Scholes delta hedging strategy when σ samples is within certain small range. For all range of σ samples, my approach is able to implicitly estimate the σ accurately with historical returns. However, the neural network using σ sample perform slightly better in general. Then two potential improvements can be done on top this deep learning algorithm for future research:

- Use a more sophisticated neural network structure in the first part of my network architecture to estimate the σ more accurately.
- The imbalanced loss due to the mixed values of σ causes that the neural network delta hedging is very inconsistent with the Black-Scholes delta at the sample trajectories with small σ. This problem might be tackled by applying some normalising factor which is dependent on the σ values to the loss function such that the losses are almost equivalent for all sample trajectories.

5.3 Neural network hedging with exponential utility function in the Black-Scholes model

As a benchmark, we first consider hedging with exponential utility without transaction cost in the Black-Scholes model. Here I consider a ATM European call option with strike price $K = S_0 = 10$.

In this part, we focus on the performances of neural network model with different values of risk-aversion parameter. Theoretically, in the Black-Scholes model, the exponential utility indifference price (2.14) reduce to the Black-Schole call option price C_{BS}. I will check if the exponential utility indifference price from the neural network satisfies this theory. Also, I used $\sigma = 0.1$ and $\sigma = 0.3$ to generate two stock sample trajectories in order to demonstrate our approach is feasible in for different data samples.

	BS call price	indifference price, $\lambda = 0.01$	indifference price, $\lambda = 0.5$	indifference price, $\lambda = 1$
$\sigma = 0.1$	0.399	0.401	0.403	0.400
$\sigma = 0.3$	1.192	1.194	1.200	1.198

Table 5.7: comparison between the Black-Scholes call option price and the exponential utility indifference prices from the neural network model at different λ.

The numerical results in Table 5.6 illustrates that the neural network accurately reproduces the Black-Scholes call option prices with exponential utility indifference prices for all chosen λ and σ.

The theoretical Black-Scholes price is found for continuous-time trading which is not achievable in real market. Hence I recalculate the Black-Scholes prices with discrete-time delta hedging strategies.

$$p_0 = -X + \Sigma_{t=0}^{T-1} \pi_t^1 (S_{t+1} - S_t) \tag{5.1}$$

The $p_0 = 1.193$ and 0.400 for $\sigma = 0.3$ and 0.1 respectively, which means our indifference is very accurate for all λ. Also note that Remark 2.3.4 only holds in continuous-time too, so the exponential indifference price and the risk-neural price do not coincide in discrete-time trading , which explains why my numerical results are slightly overpriced compared to the theoretical Black-Scholes call option price.

In order to compare the hedging performances and optimal strategies of different risk-preferences. I choose risk-aversion parameters $\lambda = 0.01, 0.5, 1$ for my neural network model. Recall that the different level of λ correspond to different levels of risk-aversion, ranging from risk-neural for λ close to 0 to very risk-averse for $\lambda \to \infty$.

In practice, traders are interested in the Profit&Loss (is also called the hedging error) at the time of maturity for different risk-preferences optimal strategies. If all optimal strategies with $\lambda = 0.01, 0.5$ and 1 are used and a common BS call price (1.192 when $\sigma = 0.3$) is charged, one can clearly see the risk-preference from Figure 5.21 and Table 5.8 (which is base on the out-of-sample test data): the P&L of the optimal strategy with $\lambda = 1$ has quantiles are closer to 0, but the the optimal strategy with $\lambda = 0.01$ yields larger extreme profits and losses. This pattern correctly corresponds to the functionality of risk-aversion parameter in a trading strategy.

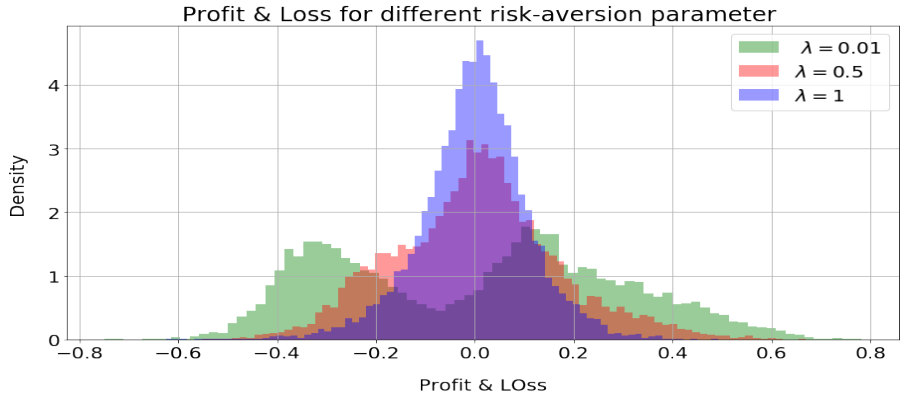

Figure 5.21: Profit&Loss of the optimal strategies with different values of λ when $\sigma = 0.3$

	5% Quantile of P&L	Median of P&L	95% Quantile of P&L
$\lambda = 0.01$	-0.411	0.0417	0.465
$\lambda = 0.5$	-0.267	0.0264	0.299
$\lambda = 1$	-0.186	0.00301	0.174

Table 5.8: Quantiles of Profit&Loss of the optimal strategies with different values of λ

To further illustrate the implications of risk-preferences on hedging, I plot the neural network optimal strategies as a function of time and stock prices for two different risk-aversion parameters. The optimal strategy with $\lambda = 0.5$ is compared to a extreme risk-taking optimal strategy with $\lambda = 0.01$ as showed in Figure 5.22, 5.23 and 5.24. One can see that the risk-taking strategy with $\lambda = 0.01$ is more flattened, essentially the risk-averse strategy would like to hold less position in stock even stock is below the strike price $K = 10$. Figure 5.25 demonstrates again, on a specific decreasing stock path, a extremely risk-taking strategy holds more positions of stock than other optimal strategies for all time.

My results in this section has firstly shown that the neural network with exponential utility function is able to reproduce the Black-Sholes call option price in the Black-Scholes model. Then we further explore the implications of risk-aversion parameter λ by looking at the optimal strategies and their Profit&Losse. One can see that our deep learning algorithm is feasible to produce a sensible optimal strategies under different risk-aversions.

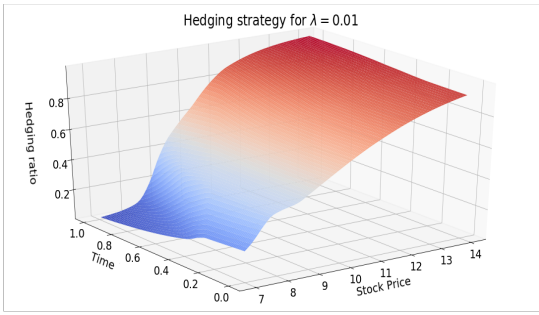

Figure 5.22: Optimal hedging strategy approximates as a function of (S_t,t) for $\lambda = 0.01$ and $\sigma = 0.3$

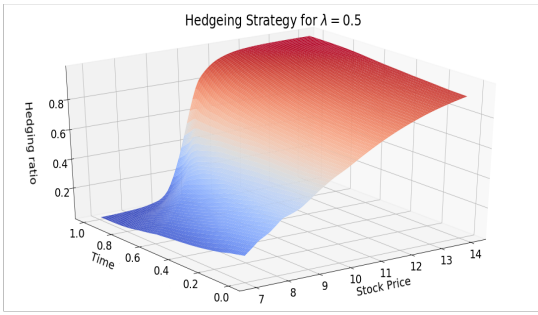

Figure 5.23: Optimal hedging strategy approximates as a function of (S_t,t) for $\lambda = 0.5$ and $\sigma = 0.3$

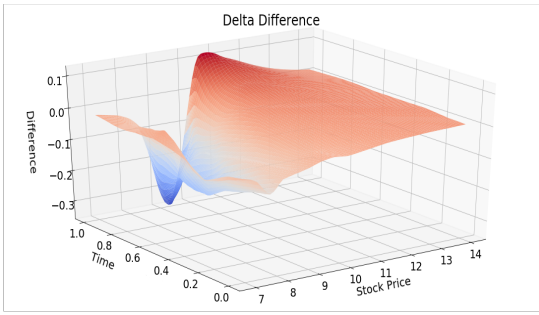

Figure 5.24: The difference between the optimal hedging strategies with $\lambda = 0.01$ and $\lambda = 0.5$

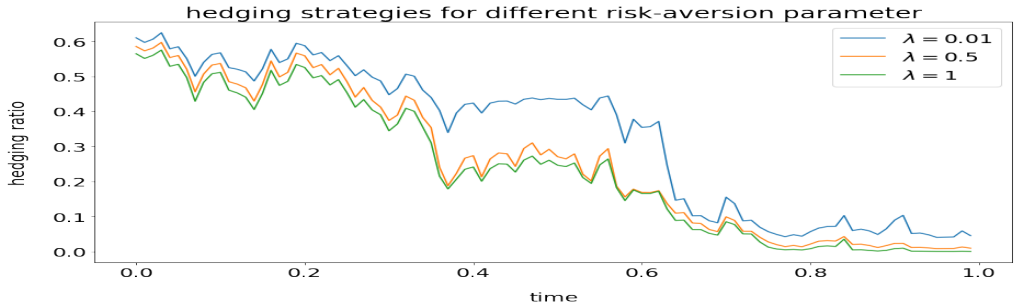

Figure 5.25: Neural network delta for different values of λ on a specific stock trajectory

5.4 Neural network hedging with transaction cost

In Section 5.3 we have seen that in the Black-Schole market model without transaction costs, deep learning is able to recover the Black-Schole call option price with exponential utility indifference pricing and can be used to represent optimal hedging strategies with different risk-aversion parameters.

In this section, I will firstly illustrate the power of the methodology by numerically calculate the exponential utility indifference price with transaction cost. Here, I revisited some numerically results in [BGTW18] with my neural network model. Essentially, motivated by the price asymptotic behaviour for small proportional transaction costs in [WW97], i.e. for

$$c_t(n) := \epsilon |n| S_t \tag{5.2}$$

and as $\epsilon \downarrow 0$. One of the results in the asymptotic analysis is that

$$p_\epsilon - p_0 = O(\epsilon^{2/3}), \qquad \epsilon \downarrow 0 \tag{5.3}$$

where $p_\epsilon = p_\epsilon(X)$ is the exponential utility indifference price of derivative X associated to transaction costs of size ϵ. So p_0 is the exponential utility indifference price without transaction cost

which we derived in Section 5.3.

Here I numerically verify (5.3) using the deep hedging algorithm for the Black-Scholes model. I choose $\sigma = 0.1$, $S_0 = 10$ and then the stock sample trajectories S_t were generated by geometric Brownian motion with $\mu = 0$, such as $S_t = S_0 \exp(t\sigma^2/2 + \sigma W_t)$. Then I used a neural network architecture defined in Section 4.2.3 to calculate the exponential utility indifference price at different values of ϵ. Recall that we used the proportional transaction cost in (5.2). In this example, I consider a European call option $X = (S_T - K)^+$ with strike price $K = S_0 = 10$. And I choose the risk-aversion parameter $\lambda = 1$ and the exponential utility indifference prices p_ϵ were calculated for $\epsilon = 0, 2^{-8}, 2^{-9}, 2^{-10}, 2^{-11}$.

I plotted $(\log(\epsilon), \log(p_\epsilon - p_0))$ at 4 different values of ϵ, since we know if (5.3) holds true, we will the obtain relation $\log(p_\epsilon - p_0) = 2/3 \log(\epsilon) + C$ for some $C \in \mathbb{R}$. Firgure 5.13 shows 4 $(\log(\epsilon), \log(p_\epsilon - p_0))$ data pairs in red points, and the closest (in squared distance) straight line in blue dot line. The blue dot line has a slope 0.657 which illustrates the asymptotic behaviour of price given by (5.3) holds with our numerical experiments.

Note that (5.2) is a result for continuous-time trading, where the exponential utility indifference price coincides with the Black-Scholes call option price. However this does not hold in discrete-time trading as I showed in Section 5.3, also there is some discrete-time friction in p_ϵ for $\epsilon > 0$. All these discrete-time friction may cause the rate of convergence in numerical experiment is slightly different to 2/3.

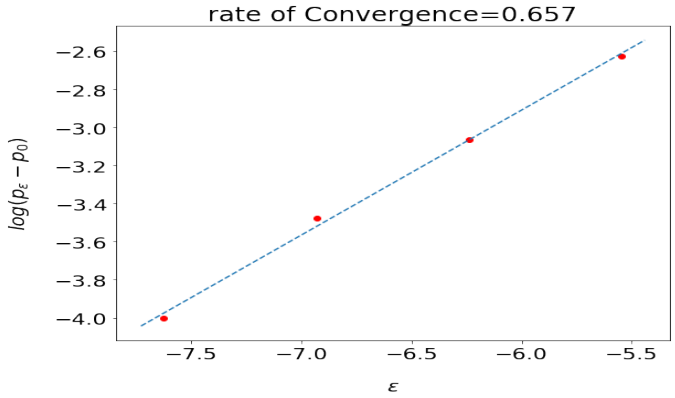

Figure 5.26: exponential utility indifference price asymptotics with transaction cost

	$\epsilon = 0.02$	$\epsilon = 0.002$	$\epsilon = 0$
$\lambda = 0.1$	1.34	1.24	1.19
$\lambda = 0.5$	1.53	1.26	
$\lambda = 1$	1.63	1.27	

Table 5.9: exponential utility difference prices for different λ under some proportional transaction costs

In order to further understand the effect of transaction cost on pricing and hedging, I experimented with several different risk-aversion parameters $\lambda = 0.1, 0.5, 1$ with different proportional transaction costs $\epsilon = 0.02$ and 0.002. I choose the volatility $\sigma = 0.3$ to simulate the stock trajectories for this example. Under incomplete market, Table 5.9 shows that the risk-adjusted price is always higher than risk-neutral price. And the higher risk-averse optimal strategy yields a higher risk-adjusted price with same transaction cost. Also, as we expected, a higher transaction cost yields a higher risk-adjusted price for the same λ.

I also evaluated the Profit&Loss at the time of maturity associated to different λ, charging the risk-adjusted prices correspondingly for a proportional transaction cost $\epsilon = 0.02$. This is shown in a histogram in Figure 5.27 for $\lambda = 0.1, 0.5, 1$ with risk-adjusted prices 1.34, 1.53 and 1.63 respectively. Compare Figure 5.27 to Figure 5.21 which is the histogram of Profit & Loss of optimal strategies with no transaction cost, one can see:

- Two figure have shown similar pattern that more risk-taking optimal strategy yields larger

extreme profit and loss.

- The optimal strategies with same risk-aversion parameter has larger extreme profit and loss when there is transaction costs.
- When this is transaction costs, the Profit&Loss distribution is no longer symmetric and has a longer left tail (larger extreme loss).

Figure 5.28 and 5.29 plot the optimal hedging strategies as a function of (S_t, t) with transaction cost for $\lambda = 0.1$ and 1 respectively. As same as in the complete market model, optimal hedging strategy with lower λ is more flattened compare to higher λ. It is worthwhile to point out that the optimal strategy is much more aggressive when there is some transaction cost, essentially the optimal strategy with λ holds stock at least approximately 40% of the portfolio when $\lambda = 0.1$.

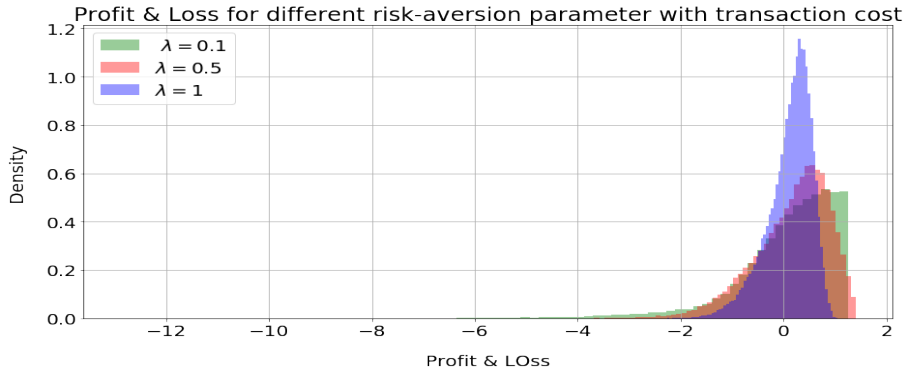

Figure 5.27: Histogram of Profit & Loss for $\lambda = 0.1, 0.5, 1$ with transaction costs $\epsilon = 0.02$

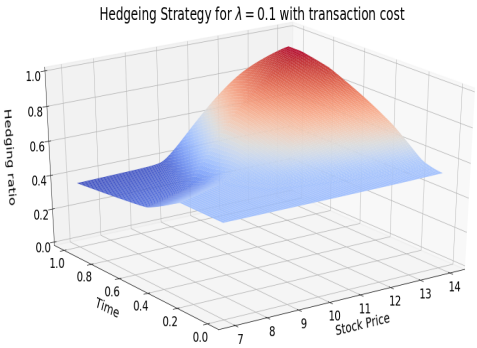

Figure 5.28: Optimal hedging strategy approximates as a function of (S_t,t) for $\lambda = 0.1$ with transaction cost $\epsilon = 0.002$, $\sigma = 0.3$

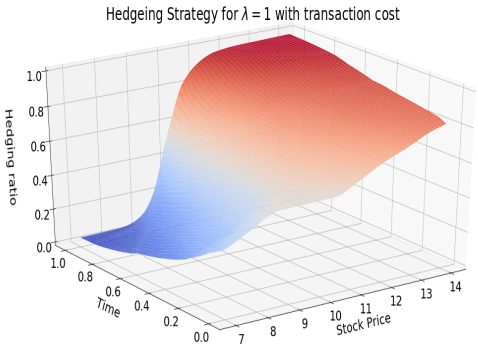

Figure 5.29: Optimal hedging strategy approximates for $\lambda = 1$ with transaction cost $\epsilon = 0.002$,$\sigma = 0.3$

Overall, my results in this section has illustrated that hedging by neural network is also feasible and accurate in an incomplete market model. Furthermore, I was able to analysis the effect of transaction cost and risk-preference on pricing and hedging with my deep learning algorithm.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

In this thesis, I analysed whether deep learning algorithm is feasible to produce accurate hedging strategies on a European call option. I trained three different neural network models for various market scenarios and trading requirements. The neural network models were tested with out-of-sample stock trajectories under different parameter settings in order to demonstrate the viability of deep learning hedging in full scope.

Based on the results, I found that the hedging strategies from my first neural network model are consistent with the Black-Scholes delta hedging strategies in all types of call options (in-the-money, at-the-money and out-the-money) and with most of volatility parameters. By plotting the hedging strategy as a function of (S_t, t), we found that the deep learning algorithm learn the Black-Scholes delta less accurately at the beginning and end of the call option due to different reasons as we discussed in Section 5.1. Overall, this neural network model has been shown that it can learn the Black-Scholes hedging strategy more accurately than the models in previous literature.

The neural network structure which first implicitly estimate σ with historical data and then output the hedging strategy has shown surprisingly good performance when the range of σ samples is small. My results illustrated that this neural network model is mostly consistent with the neural network model using σ samples directly, which infers that the poor performance of hedging strategy on large range of σ samples is because of large variation of σ samples. Furthermore, My results provided some guidelines on further improvements of the model.

As a benchmark, the neural network which uses exponential utility function was first tested in the Black-Scholes model. My results have illustrated that it is viable to reproduce Black-Scholes call option price with exponential utility indifference pricing. Then I further demonstrated that the asymptotic behaviour of option price for small proportional transaction cost in a incomplete model with deep hedging approach. Once we have shown the feasibility of neural network hedging in both complete and incomplete market model, I was then able to analyse the Profit&Loss and optimal strategies under different proportional transaction costs and risk-aversion parameters.

Overall, we have seen that hedging by deep learning algorithm allows us to provide a accurate approximation of delta hedging strategy in discrete time. Furthermore, the neural network approach can produce a reasonable optimal strategy when there is no closed form solution of hedging in a incomplete market model.

6.2 Future Work

In our consideration of the neural network model which uses historical data, there is still more we could do to improve the model in various aspect:

- As I discussed in Section 5.2, the model suffer heavily from the large variation of σ samples. Some normalising techniques can be applied to loss function so that losses are approximately equal for the sample trajectories with different σs.

- In order to make this model use no information of σ, we can apply a similar neural network model in Section 5.3 to learn the Black-Scholes call option prices.

- In practice, the strike price is set according to σ such that :

$$K \in (S_0 - \epsilon\sigma, S_0 + \epsilon\sigma)$$

  for some constant ϵ. So in theory, we could simulate the strike prices which is dependent on the σ samples, then use them as the inputs of neural network model. Such model is more efficient to use because one can avoid model training on specific strike price. However, we need study if the additional variability of inputs cause the neural network produce inconsistent results.

Furthermore, we can extend hedging by deep learning approach into some more realistic hedging model with more sophisticated neural network architecture. Inspired by [BGTW18], we can further develop the deep hedging methodology in the following areas:

- Implement our neural network model in a stochastic volatility model, such as the Heston model [Hes93].
- Replace the simple neural network by recurrent neural network model, the recurrent neural network has been shown that it can obtain more accurate results in [BGTW18].
- Implement the neural network approach on a high-dimensional hedging problem where we have several hedging instruments.
- Study the out-of-sample performance of the hedging strategies with real market data.

Bibliography

[AKS96] U. Anders, O. Korn, and C. Schmitt. Improving the pricing of options: a neural network approach. *ZEW Discussion Papers*, 96(4), 1996.

[BFK12] Kevin D. Brewer, Yi Feng, and Clarence C. Y. Kwan. Geometric brownian motion, option pricing, and simulation: Some spreadsheet-based exercises in financial modeling. *Spreadsheets in Education (eJSiE)*, 5(3), 2012.

[BGTW18] H. Buehler, L. Gonon, J. Teichmann, and B. Wood. Deep hedging. *ArXiv e-prints*, February 2018. arXiv:1802.03042v1 [q-fin.CP].

[BM73] F. Black and M.Scholes. The pricing of options and corporate liabilities. *The Journal of Political Economy*, 81(3):637–654, 1973.

[CO15] Francois Chollet and Others. Keras. https://keras.io, 2015.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. arXiv:1412.6980v9[cs.LG].

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[Hes93] Steven L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies*, 6(2):327–343, 1993.

[HH18] Catherine F. Higham and Desmond J. Higham. Deep learning: A introduction for applied mathematics. *ArXiv e-prints*, January 2018. arXiv:1801.05894v1 [math.HO].

[HLP94] James M. Hutchinson, Andrew W. Lo, and Tomaso Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, 49(3):851–889, 1994.

[KB17] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *ArXiv e-prints*, January 2017. arXiv:1412.6980v9[cs.LG].

[Met73] Robert C. Metron. Theory of rational option pricing. *The Bell Journal of Economics and Management Science*, 4(1):141–183, 1973.

[MP43] Warren S. Mcculloch and Walter H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[Ng16] Andrew Ng. Optimization gradient descent and cost function trouble, 2016.

[Pak18] M. Pakkanen. Pricing and hedging in financial markets. Lecture Notes, February 2018.

[Rom12] S. Roman. *Introduction to the mathematics of finance, 2nd edition*. New York, 2 edition, 2012.

[Sta17] L. Stark. Machine learning and options pricing: A comparison of black-scholes and a deep neural network in pricing and hedging dax 30 index options. December 2017.

[TH12] T. Tieleman and G. Hinton. Lecture 6.5 - rmsprop. COURSERA: Neural Networks for Machine Learning. Technical report, 2012.

[WW97] A. E. Whalley and P. Wilmott. An asymptotic analysis of an optimal hedging model for option pricing with transaction cost. *Mathematical Finance*, 7(3):307–324, 1997.

[Xu06] M. Xu. Risk measure pricing and hedging in incomplete markets. *Annals of Finance*, 2(1):51–71, 2006.