Deep_Hedging_with_Transaction_Costs_and_Risk_Preference

by Lauren Luo

Imperial College London

Department of Mathematics

Deep Hedging with Transaction Costs and Risk Preferences

Author: Tianyu Luo (CID: 02163578)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2021-2022

Declaration

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

Acknowledgements

First and foremost, I would like to express my deepest thanks to my academic supervisor, Dr Mikko Pakkanen, for his patience and continuous support throughout the project. His expertise in deep learning and valuable insights guide me through the challenges encountered in the dissertation. The advice always provides me with direction for my further development in the thesis.

Secondly, I would also like to give my warmest thanks and gratitude to my industry supervisors from TD Securities, Darryl Copsey and Nan Wu, who made this opportunity open to me, as well as Gabriele Pompa and Akshay Parmar, whose sophisticated industrial knowledge, wholehearted support and enthusiasm always inspire and encourage me to achieve my best. It is an honour to work in a respectful, open and cohesive team at TD Securities. At the same time, I would like to thank all the team members of the Quantitative Modelling Analytics team for being friendly and welcoming.

Finally, I would like to thank my Mom for the unconditional support and love.

Abstract

This dissertation introduces and implements deep hedging methods to solve the traditional pricing and hedging problem in the financial industry. The deep hedging method uses deep learning models to proceed with the market information and output the predicted hedging strategies. As a result, the method is efficient in processing large scales of the data set and the predictions are data-driven, depending only on the input and the choice of network structures. This allows us to incorporate the practical conditions such as the transaction costs and risk preference effortless compared to the traditional analytical approaches that are either computationally inefficient or impractical.

However, the 'black-box' behaviour of the deep learning models makes the results less trustworthy. In this dissertation, we are going to demonstrate and interpret the outputs, the deep hedging strategies, in the context of finance, reasoning the behaviours impacted by the changes in the cost rates and risk preferences. Moreover, the paper will also demonstrate and compare the deep hedging strategies and the indifference prices trained by different deep learning models, feedforward neural networks (FNN), long short-term memory (LSTM) and gated recurrent unit (GRU). The results from the experiments show reasonable behaviours on how strategies and indifference prices vary under different cost rates and risk preferences.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Derivatives are crucial financial instruments for hedgers to mitigate the risk of volatile underlying assets. Especially for the sell-side traders, hedging and pricing the derivatives products are two major responsibilities for them. The famous Black-Scholes delta hedge [2] allows traders to hedge the portfolios continuously under the complete market assumptions and obtain the risk-neutral prices. In practice, traders are traded with restrictions such as transaction costs, bid-ask spreads, the temporary and permanent price impacts and even the timing to execute the trade in the discrete-market world. All these factors violate the Black-Scholes assumption. There is also literature studying the models under market frictions. For example, in [3], it treats illiquidity as a transaction cost that has a temporary impact on the price, which follows the black-shcoles model. By computing the system of partial differential equations (PDEs), optimal hedging strategies under the liquidity constraints can be obtained. However, in reality, the numerical methods such as the finite difference approximation for solving the PDEs is inefficient for computation and time-consuming. Moreover, the Black-Scholes model does not generalize well on multi-dimensional space.

Meanwhile, deep learning is known for its ability to self-learn the pattern of the existing data and predict results using new data sets through optimizing a customized rule, namely the objective unctions. Not surprisingly, machine learning training frameworks can be applied to the financial industry. The *unsupervised* deep learning methods that apply to the derivatives products were firstly conducted by JP Morgan and ETH Zurich in the paper [4]. It is more practical to solve the hedging and pricing problems since they can easily handle market frictions and trading constraints. According to the paper, the deep hedging method is especially suitable for pricing and managing the risk of the *over-the-counter* derivatives, which do not have listed comparable market prices, or the product may be too complicated to price and hedge using the traditional analytical methods. Since it uses the deep learning method to solve the problem, it is data-driven and model-free in nature, and also efficient in processing the large data set. The methods presented in this dissertation are all based on the work in [5], [4] and [6].

Intuitively, the deep hedging methods obtain the optimal hedge ratios through minimizing the objective functions. It is an unsupervised learning deep learning algorithm, so the objective functions is not a function between the labels and the predictions, instead, the objective functions $\mathcal{L}$ will be a measure ρ of the profit and loss (PnL), where

$$PnL = \text{hedge portfolios - payoff of a contingent claim},$$

i.e., $\mathcal{L} = \rho(PnL)$. The deep hedging strategies produce the hedge portfolios that minimize the objective functions. In other words, deep hedging strategies is the minimizer of the objective function $\mathcal{L}$. This is where we can add the market frictions and risk preferences.

For example, if we would like to obtain the risk-neutral hedging strategies, we can choose $\rho(x) = \mathbb{E}(PnL^2)$. Similarly, we can choose $\rho(x) = \mathbb{E}(\exp(-\lambda x))$, where λ is the risk-averse level, to incorporate the risk preferences. Transaction costs can also be taken into the considerations by subtracting the cost from the PnL.

This dissertation focuses on finding the optimized hedging strategies, i.e., the deep hedging strategies, and the fair prices under different training models with different levels of cost rates and risk preferences for the European call options. The underlying market simulator follows the Black-Scholes model. The goal of the study is to illustrate how the variation in the cost rates and risk preferences impact the trading strategies, interpret the results in the context of finance and also demonstrate the difference in the training results under different deep learning models. The objectives are as follows.

- implements deep hedging strategies under three different deep learning models, Feedforward Neutral Network (FNN), Long Short-term Memory (LSTM) and Gated Recurrent Unit (GRU);
- applies different levels of proportional cost rates and risk preferences to the loss function of each model and trains the model to get the deep hedging strategies;
- conducts statistical tests among predicted strategies under different learning models to determine the differences and performances.

outline

The dissertation is organized into four chapters. Chapter 1 is going to introduce the overall deep learning methods, where the three deep learning models, FNN, LSTM and GRU are discussed in detail, as well as the three gradient-based optimization algorithms. Chapter 2 explains the deep hedging methods, linking the deep learning framework to the financial pricing and hedging problem. Chapter 3 will describe the training procedures and methodologies such as the market simulator for the data input, the structures for the model implementation and the training algorithms. The last chapter will demonstrate all the predicted hedging strategies, as well as the indifference prices.

Chapter 2

Deep Learning Models

2.1 Overview of Deep Learning

The deep learning models are good for capturing non-linearity relationships among data. It is essentially a mapping from one data set, $\boldsymbol{x} \in \mathbb{R}^m$, to another, $\hat{\boldsymbol{y}} \in \mathbb{R}^n$, i.e., $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^n$

$$\hat{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}) \tag{2.1.1}$$

where $\boldsymbol{\theta}$ are parameters that can be repeatedly utilized, $m, n \in \mathbb{N}+$.

For supervised learning, it provides true values, $\boldsymbol{y}_i \in \mathbb{R}^n$ which is known as 'labels', to compare to the predicted values $\hat{\boldsymbol{y}}_i$, while unsupervised learning does not have 'labels', only a referenced value $\boldsymbol{y} \in \mathbb{R}^n$ is given. Then for each data set $\boldsymbol{x}_i \in \mathbb{R}^m$, where $i = 1, 2, \dots, N$, we have a loss function that calculate a 'score' to measure the performance of the prediction, i.e.,

$$loss_i = l(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i) \tag{2.1.2}$$

Then the network learns the data pattern by optimizing the objective function which is the empirical expectation of the loss function, denoted $\mathcal{L}$, i.e.,

$$\begin{aligned} \mathcal{L} &= \hat{\mathbb{E}}(l(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i)) \\ &= \frac{1}{N}\sum_{i=1}^{N} l(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i) \\ \Leftrightarrow \quad \mathcal{L}(\boldsymbol{\theta}) &= \frac{1}{N}\sum_{i=1}^{N} l(\boldsymbol{f}(\boldsymbol{\theta}|\boldsymbol{x} = \boldsymbol{x}_i), \hat{\boldsymbol{y}}_i), \end{aligned} \tag{2.1.3}$$

A commonly used optimization methodology for objective function is Stochastic Gradient Decent (SGD), which will be introduced in section 2.4.2. As mentioned in [7, chap 3.2], by separating training data into multiple subsets, SGD becomes more efficient than the traditional Gradient Descent (GD) optimization, which makes large data sets processing possible.

The capacity to proceed with a large amount of data and to capture the complexity of the functions enable deep learning to do complex tasks such as image classification [8], speech recognition [9], and sequential data prediction, as mentioned in [7, p 5]. The deep hedging method also inherits these advantages. More details about deep hedging are in the chapter 3.

Deep learning is being introduced in two parts in this dissertation. The first part is the place where data input has been received and reformed, which includes model structures such as FNN (2.2), LSTM (2.3.2) and GRU (2.3.3). The second part is about how data is learned under certain gradient-based optimization methods (2.4), in which the parameters of models are altered through the feedback from the objective function.

2.2 Feedforward Neutral Network

It is inevitable to introduce the feedforward neural networks(FNN) at the beginning since it is a typical representation of the deep learning model. As mentioned in [1, Chapter 6], the data are passed into the FNN in a forward direction and do not reuse the output information, and this is where the name feedforward comes from. The FNN is composed of multiple linear transformations between layers, combined with non-linear component-wise transformations (done by the activation functions) at each layer, eventually constructing an output with a non-linear relationship to the input.

2.2.1 Architecture

The illustration of the FNN structure is shown in the figure 2.1

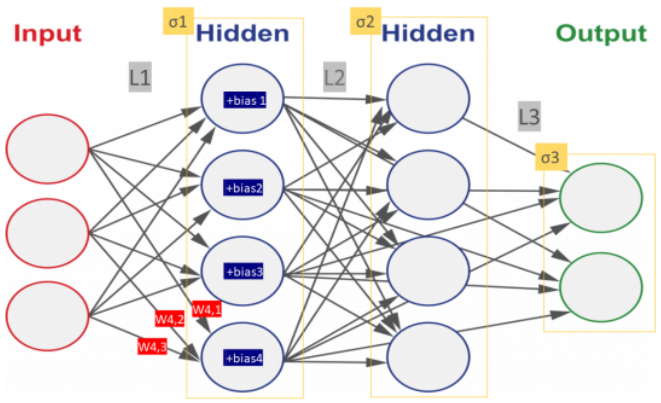


Figure 2.1: Illustration of FNN Structure

The model has three parts, which are the input layer, the hidden layer(s) and the output layer. Each layer is composed of one or multiple units. The number of units of the input layer is the dimension of the input $\boldsymbol{x} \in \mathbb{R}^m$, while the number of units of the output layer depends on the $\boldsymbol{y} \in \mathbb{R}^n$. Hidden layers are all the layers between input and output layers, of which the number of layers, $r \in \mathbb{N}+$, and the length of each hidden layer, $d_i \in \mathbb{N}+$, $i \in \mathbb{N}$, are the choice of our own.

Each hidden layer is a linear transformation of the previous layer. In other words, each unit of the hidden layer is the linear combination of all units of the previously hidden layer plus a bias term. As presented in the last unit of the second column in the figure 2.1, it is the weighted average of the units in the first column with a bias added to it. The procedure is the same with all units in all layers after the input layer.

After the linear transformation is done, which may alter the dimension of the data, we regularize the transformation, always a non-linear function, by applying a component-wise activation function to it. This procedure is presented as the yellow rectangle in the figure 2.1. Some useful activation functions will be introduced in section 2.2.2. The information go through each layer following the same procedure and finally is reconstructed as a non-linear transformation of the input.

Knowing how data proceeds within the FNN, we could represent the network that roughly define in (2.1.1) in a more details way, the definition for FNN is presented as follows.

Definition 2.2.1 (feedforward neutral networks layer)**.** Let $d_0, d_r, r \in \mathbb{N}+$, where d_0, d_r are the input size and output size respectively, and $r-1$ hidden layers and $d_i \in \mathbb{N}+$ is the number of units of the i^{th} layer for $i \in \{1, \ldots, r-1\}$. Then we define each layer other than the input layer as $\mathcal{H}_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$, where $i = 1, 2, \ldots, r$,

$$\begin{aligned} \mathcal{H}_i &= \boldsymbol{\sigma}_i(\boldsymbol{L}_i(\boldsymbol{x}_{i-1})) \\ &= \boldsymbol{\sigma}_i(\boldsymbol{L}_i(\mathcal{H}_{i-1})) \\ &= \boldsymbol{\sigma}_i(\boldsymbol{L}_i(\ldots \boldsymbol{\sigma}_1(\boldsymbol{L}_1))), \end{aligned} \tag{2.2.1}$$

where $\boldsymbol{x}_{i-1} \in \mathbb{R}^{d_{i-1}}$ is the output of the last layer, $\boldsymbol{\sigma}_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$ is the activation function that applies at the layer and $\boldsymbol{L}_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ is an affine function that is a linear transformation of the previous layer such that

$$\boldsymbol{L}_i(\boldsymbol{x}_{i-1}) = \boldsymbol{W}_i \boldsymbol{x}_{i-1} + \boldsymbol{b}_i, \tag{2.2.2}$$

where $\boldsymbol{W}_i \in \mathbb{R}^{d_{i-1} \times d_i}$ is the weight matrix, and $\boldsymbol{b}_i \in \mathbb{R}^{d_i}$ is the bias vector.

Definition 2.2.2 (feedforward neural networks)**.** Let $d_0, d_r, r \in \mathbb{N}+$. Define the feedforward neural(FNN) as $f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_r}$ with $r-1$ hidden layers and $d_i \in \mathbb{N}+$ is the number of units of the i^{th} layer for $i \in \{1, \ldots, r-1\}$.

Then the feedforward neural network is a composition of all layers $\mathcal{H}_i$ such that

$$f = \mathcal{H}_r \circ \mathcal{H}_{r-1} \circ \ldots \mathcal{H}_1 \tag{2.2.3}$$

We denote the class of such functions f by

$$\mathcal{N}_r(\overbrace{d_0, d_1, \ldots, d_r}^{\text{depth of the network}} ; \boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_r) \tag{2.2.4}$$

We can see clearly the structure in Definition 2.2.2 that the FNN is multiple layers stack together. The number of layers is also called the depth of the networks, while the number of units is the width. The word 'deep' learning is referred to the depth of the network, which is able to learn the complexity of the function at an exponential learning speed due to the composition structure, as mentioned in [7, p18].

2.2.2 Activation Functions

The activation functions, as mentioned in Definition 2.2.2 and Definition 2.2.1, are the keys to the non-linearity fitting of FNN networks. Normally, activation functions can be characterized in two categories, which are one-dimensional and two-dimensional. Some common 1-dimensional activation functions are as follows.

- ***Identity***:

$$g(x) = x, \quad g \in \mathbb{R}. \tag{2.2.5}$$

The identity activation function usually used in the output layer if we expect an unbounded output. Note that it is not appropriate to use the identity function for all layers at a time, otherwise it will make the FNN to be a linear approximation, or an affine function with respect to the input function.

- ***Sigmoid***:

$$g(x) = \tfrac{1}{1+e^{-x}}, \quad g \in (0,1). \tag{2.2.6}$$

- ***Hyperbolic Tangent (tanh)***:

$$g(x) = \tfrac{e^x - e^{-x}}{e^x + e^{-x}}, \quad g \in (-1,1). \tag{2.2.7}$$

Sigmoid and Hyperbolic Tangent have been widely used in the output layers when we want a bounded result. Traditionally they have been used in the hidden layers, however, the unboundness in this case is not compatible to the gradient-based optimization [10, section 3].

- ***Rectified Linear Unit (ReLU)***:

$$g(x) = \max\{x, 0\}, g \in (0, \infty] \tag{2.2.8}$$

ReLU function is one of the most popular activation functions for the hidden layers nowadays. It is efficient due to its simple-to-calculate derivatives which is a crucial step in the gradient-based optimization. Although when $x = 0$, the gradient is undefined, we can practically replace it by 1 with limited impact, as mentioned in [7]. It is also the activation function that our dissertation uses. For other problem such as freezing gradient when $x < 0$, [11] and [12] provide solutions using modified version of ReLU functions.

2.2.3 Universal Approximation

The Universal Approximation Property is the source of power behind the FNN. While firstly the single hidden layer FNN network has been proved to be a universal approximator under certain restrictions on the activation function in articles [13] and [14], the universal approximation property is also proved to be true in article [15] for multilayers FNN with a nonpolynomial activation function. In other words, almost any reasonable relationship between input and output of the FNN can be established under the certain assumption of activation functions.

Here, we follows the theorem that is more general by [15, Theorem 1 and Proposition 1] and reformulated in [7, Theorem 2.22].

Theorem 2.2.3 (Universal approximation property)**.** *Let $g : \mathbb{R} \to \mathbb{R}$ be a measurable function such that*

- *g is not a polynomial function,*
- *g is bounded on any finite interval,*
- *the closure of the set of all discountinuity points of g in $\mathbb{R}$ has zero Lebesgue measure.*

Let $K \subset \mathbb{R}^{d_0}$ be compact and $\epsilon > 0$, where d_0 is the network input size.

1. *For any $u \in C(K, \mathbb{R})$, there exist $d_1 \in \mathbb{N}+$ and $f \in \mathcal{N}_2(d_0, d_1, 1; g, Id)$ such that*
$$||u - f||_{sup,K} < \epsilon$$
2. *Let $p \geq 1$. For any $v \in \boldsymbol{L}^p(K, \mathbb{R})$, there exist $d_1' \in \mathbb{N}+$ and $h \in \mathcal{N}_2(d_0, d_1', 1; g, Id)$ such that*
$$||v - h||_{\boldsymbol{L}^p(K)} < \epsilon$$

where $\mathcal{N}_2$ is the FNN as in Definition 2.2.2 and Id is the identity sigmoid function as in (2.2.5)

This property also motivates the deep hedging methods as said in [6, section 4.2], and deep hedging method will be introduced in chapter 3.

2.3 Recurrent Neural Network

2.3.1 Standard Recurrent Neural Network

While FNN is not capturing the information of previous data, the recurrent neural network (RNN) is constantly taking advantage of the output at each time stamp of the input data. For example, if we have a vector of sequential data $\boldsymbol{x} \in \mathbb{R}^n$ input into the RNN, it will process $x_i \in \boldsymbol{x}$ one at a time and learn from the output generated by previous data, $x_1, \ldots, x_{i-1}$. In other words, it can capture the correlation between the current and previous data input, which is beneficial for processing the time series data.

We can interpret the RNN in two ways, folded and unfolded, as shown on the left-hand and right-hand sides of the the figure 2.2 respectively where h_t represents the hidden state at time stamp t.

For folded RNN (left-hand side of the figure 2.2), it represents that the current hidden state depends on the previous hidden state and the data at the current time. In other words, RNN depends on the previous information.

By unfolding the RNN, i.e., keep expanding the previous hidden states until reaching the first time stamp, the networks can be represented as multiple copies of the same networks with different input values, as shown in the right-hand side of the figure 2.2. Now, the unfolded RNN has

1. a fixed input size depends on the choice of the time stamp and
2. parameters shared by the repetitive networks at each time stamp.

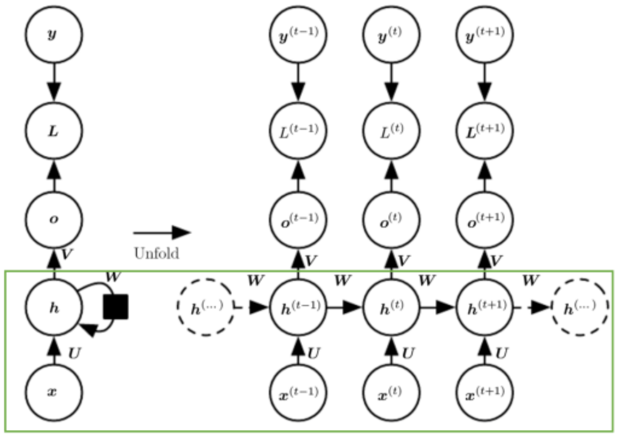


Figure 2.2: Unfolding RNN [1, p373]

These two features allow RNN to determine the relationships among data history, save time and space training different models at each time stamp and also illustrate how information proceeds forward (information flows into the system and generates output) and backward (calculate gradient using backward-propagation algorithm).

In this dissertation, we refer to the RNN model as in the figure 2.2, i.e., at each time stamp $t = 0, \dots, T$ and $T \in \mathbb{N}+$, for each $\boldsymbol{x}^{(t)} \in \mathbb{R}^n$, the network produces an output $\boldsymbol{o}^{(t)} \in \mathbb{R}^m$, and the hidden states $\{\boldsymbol{h}^{(i)}\}_t$ receive, store and pass past information, where $m, n \in \mathbb{N}+$. The output at each time stamp is then inputted into the loss function $L^{(t)}$ together with a corresponding reference value $\boldsymbol{y}^{(t)} \in \mathbb{R}^m$. Note that we usually set the initial hidden state $\boldsymbol{h}_0 = \boldsymbol{0}$.

There is also RNN design pattern that only outputs values at the last time stamp, or that passes the information depending on the transformed output instead of the hidden states. Illustrations of other types of RNNs are in the appendix A.

Let us define an RNN forward propagation as shown in the figure 2.2 to clarify how data proceed within the network.

Definition 2.3.1 (Recurrent Neural Networks Forward Propagation)**.** Let $i \in \mathbb{N}$, and $n, m, k, T \in \mathbb{N}+$, define the hidden state at each time stamp $t = 1, \dots T$ as $\boldsymbol{h}^{(t)} : \mathbb{R}^m \to \mathbb{R}^m$, weight matrices for hidden-to-hidden connection as $\boldsymbol{W} \in \mathbb{R}^{m \times m}$, for input-to-hidden connection as $\boldsymbol{U} \in \mathbb{R}^{m \times n}$ and hidden-to-output connection as $\boldsymbol{V} \in \mathbb{R}^{m \times m}$, and bias vector $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^k$. Denote component-wise activation functions as $\boldsymbol{\sigma}_h, \boldsymbol{\sigma}_o : \mathbb{R}^m \to \mathbb{R}^m$, we can represent the hidden state and output at each time stamp t as,

$$\boldsymbol{h}^{(t)} = \boldsymbol{\sigma}_h(\boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{b}), \tag{2.3.1}$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{\sigma}_o(\boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}) \tag{2.3.2}$$

Then, for output at each time stamp $\boldsymbol{o}_t$, we have $loss_t = l(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{o}^{(t)})$, as introduced in (2.1.2), to measure its performance, and use gradient-based optimization algorithm to update the parameters as introduced in chapter 2.4.

Vanishing and Exploding Gradient Problem

Suppose that there exists an activation function $\boldsymbol{\sigma}_h$ that lets the hidden-to-hidden connection matrix $\boldsymbol{W}$ multiply by themselves repeatedly such that

$$\begin{aligned} \boldsymbol{h}^{(t)} &= \boldsymbol{W}^t \boldsymbol{g}(\boldsymbol{h}_0) \\ &\overset{\text{eigendecomposition}}{=} \boldsymbol{Q} diag(\boldsymbol{\lambda})^t \boldsymbol{Q}^T \boldsymbol{g}(\boldsymbol{h}_0), \end{aligned} \tag{2.3.3}$$

for some $\boldsymbol{g}$.

We can see from (2.3.3) that if there exists a $|\lambda_i| > 1$ and $\lambda_i \in \boldsymbol{\lambda}$, then $\boldsymbol{W}^t \to \infty$ as $t \to \infty$. Similarly, $\boldsymbol{W}^t \to 0$ as $t \to \infty$ if any $|\lambda_i| < 1$. As introduced in chapter 2.4, the gradient with respect to the model parameters tells us the direction to improve our training results. A large magnitude of $\boldsymbol{W}^t$ will cause the explosion of the gradient and makes the learning process unstable, while if it becomes too close to 0, the parameters will stop learning since no direction for the update is provided. This is the vanishing and exploding gradient problem [1, section 8.5.2].

Moreover, even if the gradients are not exploding or vanishing, due to the composition over the same parameters, the weights still *decrease exponentially* when t becomes larger and larger, which makes RNN loses information over long-term dependencies over data set.[1, chapter 10.7]

In conclusion, RNN is designed for capturing the dependencies of the sequential data, however, the standard RNN has some drawbacks.

1. The gradient is unstable in some cases, either too large or too small and interrupts training,
2. may not catch the long-term dependencies of the sequential data.

2.3.2 Long Short-Term Memory

Fortunately, the **Long Short-Term Memory** (LSTM) [16] solves the vanishing or exploding gradient problem by altering the connection weight matrices between hidden states at each time stamp, avoiding a situation that the same weight matrix repeatedly multiplied by itself. The crucial step is to let the network *forget* some of the older states. In other words, LSTM can learn when and where to remove irrelevant information from the past.

The overall structure of LSTM is similar to the regular RNN regarding unfolded structure. It is constructed by connected cell states. What is different from the RNN is computation within the cell state. The special setting in LSTM is the *memory cell states* $\boldsymbol{C}_t$ which is the stable information flow carrying the relevant historical and current data *at all time stamps* and will not fade away as t goes further. The LSTM cell structure is illustrated in the figure 2.3. It is updated at every time stamp by three control gates.

The **forget gate** $\boldsymbol{F}_t$ takes the previous hidden states $\boldsymbol{H}_{t-1}$ and current data $\boldsymbol{x}_t$ as inputs, determines the relevance of the information and *forget* the *previous* irrelevant information in $\boldsymbol{C}_{t-1}$ by applying a sigmoid activation function indicating the level of importance.

The **input gate** $\boldsymbol{i}_t$ is going to decide the *update* of the current cell states, determining how much current information should be obtained based on the new candidate memory cell state $\tilde{\boldsymbol{C}}_t$ also by applying a sigmoid activation function, determining the level of importance. The final output from this gate will add to $\boldsymbol{C}_{t-1}$.

At this stage, we have our new cell state $\boldsymbol{C}_{t-1}$ updated to $\boldsymbol{C}_t$ by forgetting irrelevant information from the past and adding new information based on current data. A similar procedure is then followed by the **output gate** $\boldsymbol{O}_t$, which also determines the level of importance by applying a sigmoid activation function to $\boldsymbol{x}_t$ and $\boldsymbol{H}_{t-1}$. Combining the indicator with the current cell state $\boldsymbol{C}_t$, we have our output $\boldsymbol{H}_t$ which is also the hidden states for the next cell.

Here is the definition of the procedures that happened within the cells.

Definition 2.3.2 (LSTM forward propagation)**.** Let $\boldsymbol{W}_f, \boldsymbol{W}_i, \boldsymbol{W}_c, \boldsymbol{W}_o \in \mathbb{R}^{m \times m}$, $\boldsymbol{U}_f, \boldsymbol{U}_i, \boldsymbol{U}_c, \boldsymbol{U}_o \in \mathbb{R}^{m \times n}$, $\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_c, \boldsymbol{b}_o \in \mathbb{R}^m$, $m, n, T \in \mathbb{N}+$, $\boldsymbol{x}_t \in \mathbb{R}^n, \boldsymbol{H}_t \in \mathbb{R}^{m \times 1}$, where $t = 1, \dots, T$, and a component-wise sigmoid ($\boldsymbol{\sigma}$) and hyperbolic tangent (tanh), as defined in (2.2.6) and (2.2.7).

Define $\boldsymbol{F}_t, \boldsymbol{I}_t, \tilde{\boldsymbol{C}}_t, \boldsymbol{O}_t : \mathbb{R}^{m \times 1} \to \mathbb{R}^{m \times 1}$ such that

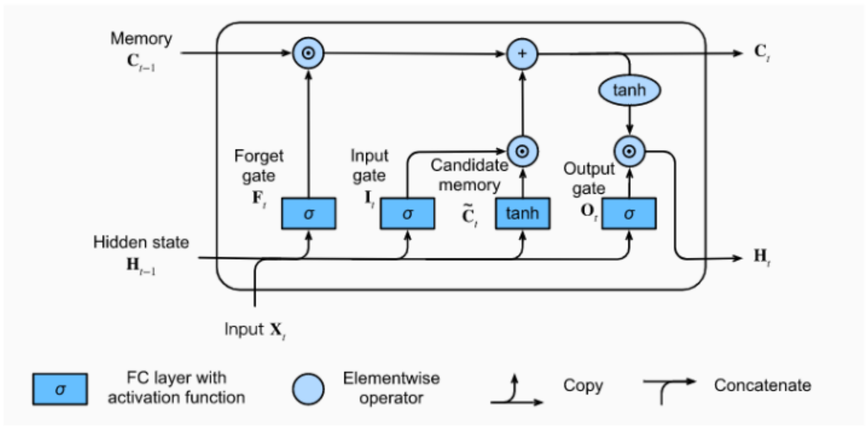


Figure 2.3: LSTM Cell Structure

$$\boldsymbol{F}_t = \boldsymbol{\sigma}(\boldsymbol{W}_f \boldsymbol{H}_{t-1} + \boldsymbol{U}_f \boldsymbol{x}_t + \boldsymbol{b}_f), \tag{2.3.4}$$

$$\boldsymbol{I}_t = \boldsymbol{\sigma}(\boldsymbol{W}_i \boldsymbol{H}_{t-1} + \boldsymbol{U}_i \boldsymbol{x}_t + \boldsymbol{b}_i), \tag{2.3.5}$$

$$\tilde{\boldsymbol{C}}_t = \mathbf{tanh}(\boldsymbol{W}_c \boldsymbol{H}_{t-1} + \boldsymbol{U}_c \boldsymbol{x}_t + \boldsymbol{b}_c), \tag{2.3.6}$$

$$\boldsymbol{O}_t = \boldsymbol{\sigma}(\boldsymbol{W}_o \boldsymbol{H}_{t-1} + \boldsymbol{U}_o \boldsymbol{x}_t + \boldsymbol{b}_o), \tag{2.3.7}$$

where $\boldsymbol{F}_t$ is the **forget gate**, $\boldsymbol{I}_t$ is the **input gate**, $\tilde{\boldsymbol{C}}_t$ is the *candidate memory* and $\boldsymbol{O}_t$ is the **output gate**.

Then the updated cell will be

$$\boldsymbol{C}_t = \boldsymbol{F}_t \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \tilde{\boldsymbol{C}}_t, \tag{2.3.8}$$

and the final output at t is

$$\boldsymbol{H}_t = \boldsymbol{O}_t \tanh(\boldsymbol{C}_t) \tag{2.3.9}$$

2.3.3 Gated Recurrent Unit

Another common and effective gated RNN is the **Gated Recurrent Unit (GRU)**. Similar to LSTM, it also has a mechanism to *update* and *optionally* pass the necessary past information, which solve the *vanishing and exploding gradient* problem. However, **GRU** only contains 2 gates, which are **update gate** and **reset gate**. The cell structure is in the figure 2.4.

The GRU network first determines the level of importance of the previous and the current information in the *reset gate* $\boldsymbol{R}_t$ and *update gate* $\boldsymbol{Z}_t$, respectively, in the scale of 0 to 1 by applying the component-wise sigmoid activation function as defined in (2.2.6). In other words, the outputs of the $\boldsymbol{R}_t$ and $\boldsymbol{Z}_t$ are the proportions of how much the past information we would like to preserve and how much new information is going to add to the states.

Apply the *reset gate* output $\boldsymbol{R}_t$ to the previous hidden state $\boldsymbol{H}_{t-1}$ then combine it with the current input data $\boldsymbol{x}_t$ to generate a *candidate hidden state* $\tilde{\boldsymbol{H}}_t$. For example, if $\boldsymbol{R}_t = \mathbf{1}$, it means that we keep all of the previous information and the complete hidden states $\boldsymbol{H}_{t-1}$ from the past will be input into the *candidate hidden state* $\tilde{\boldsymbol{H}}_t$, becoming a standard RNN state as described in (2.3.1), i.e., the candidate hidden states $\tilde{\boldsymbol{H}}_t = \tanh(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{W}\boldsymbol{H}_t + \boldsymbol{b})$.

The last step is to make a convex combination between the *candidate hidden state* $\tilde{\boldsymbol{H}}_t$ and the previous hidden state $\boldsymbol{H}_{t-1}$ through the *update gate* $\boldsymbol{Z}_t$. The final output will be our new hidden state $\boldsymbol{H}_t$.

Here we introduce the process in detail.

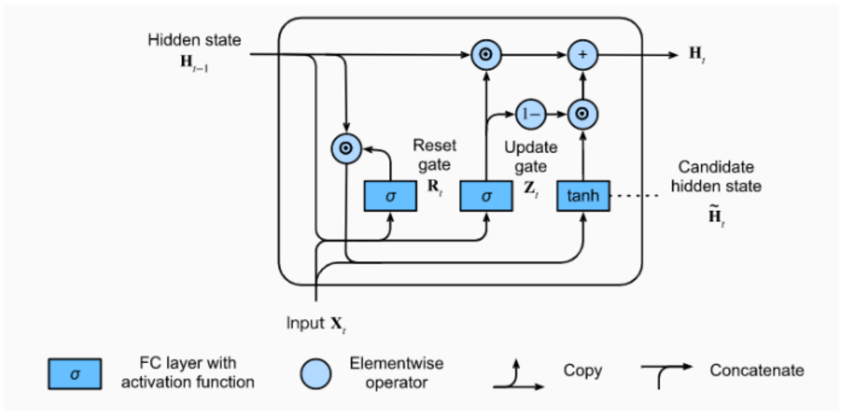

Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Reset gate $\mathbf{R}_t$
Update gate $\mathbf{Z}_t$
1−
σ
σ
tanh
Candidate hidden state $\tilde{\mathbf{H}}_t$
Input $\mathbf{X}_t$
FC layer with activation function
Elementwise operator
Copy
Concatenate

Figure 2.4: GRU cell structure

Definition 2.3.3 (Gated Recurrent Unit forward propagation)**.** Let $\boldsymbol{W}_r, \boldsymbol{W}_z, \boldsymbol{W}_h \in \mathbb{R}^{m\times m}$, $\boldsymbol{U}_r, \boldsymbol{U}_z, \boldsymbol{U}_h \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b}_r, \boldsymbol{b}_z, \boldsymbol{b}_h \in \mathbb{R}^m$, $m, n \in \mathbb{R}$, and $\boldsymbol{x}_t \in \mathbb{R}^n$, $\boldsymbol{H}_t \in \mathbb{R}^{m\times 1}$, where $t = 1, \dots, T$. Denote the $\odot$ as the *Hadamard Product*(element-wise vector multiplication)

Define $\boldsymbol{R}_t, \boldsymbol{Z}_t, \tilde{\boldsymbol{H}} : \mathbb{R}^{m\times 1} \to \mathbb{R}^{m\times 1}$ such that

$$\boldsymbol{R}_t = \boldsymbol{\sigma}(\boldsymbol{W}_r\boldsymbol{H}_{t-1} + \boldsymbol{U}_r\boldsymbol{x}_t + \boldsymbol{b}_r) \tag{2.3.10}$$

$$\boldsymbol{Z}_t = \boldsymbol{\sigma}(\boldsymbol{W}_z\boldsymbol{H}_{t-1} + \boldsymbol{U}_z\boldsymbol{x}_t + \boldsymbol{b}_z) \tag{2.3.11}$$

$$\tilde{\boldsymbol{H}}_t = \tanh(\boldsymbol{W}_r(\boldsymbol{H}_{t-1} \odot \boldsymbol{R}_t) + \boldsymbol{U}_h\boldsymbol{x}_t + \boldsymbol{b}_h) \tag{2.3.12}$$

Then the output hidden layer $\boldsymbol{H}_t$ is

$$\boldsymbol{H}_t = \boldsymbol{Z}_t \odot \boldsymbol{H}_{t-1} + (1 - \boldsymbol{Z}_t) \odot \tilde{\boldsymbol{H}}_t. \tag{2.3.13}$$

2.4 Gradient-Based Optimization Problems

In the section 2.3.1, we introduced the procedure that after the output generated from the deep learning network, they will then be measured by the *objective function*, or the empirical mean of the loss functions (2.1.2). We want the objection function output to be as small as possible. This chapter is going to introduce the method to minimize an *objective function*.

From [1, chapter 4], we know that optimization mostly refers to locate a minimum or maximum of a customised objective function, i.e., we want to find $\boldsymbol{x}^*$ such that $\boldsymbol{x}^* = \arg\min \mathcal{G}(\boldsymbol{x})$ for some function $\mathcal{G}$. There are many types of optimizations, we are going to use introduce the algorithms that calculate the first-order derivatives of the objective function. More information for second-order optimization can be found in [1, chapter 4.3.1] and for convex optimization in [17].

Usually, We find the minimum by setting the gradient $\nabla\mathcal{G}(\boldsymbol{x}) = 0$, and solve for the critical value. However, we cannot tell if this is a global or local extremum, or a saddle point of a function. This motivate scientists to develop other methods to approximate a minimized value. The goal for minimization in deep learning is to find a close enough minimized value instead of a true global minimum, says [1, Chapter 4.3, p81].

2.4.1 Gradient Descent

Gradient tells us the direction to improve the optimized value. Intuitively, for example, in a 1-dimension world as shown in the figure 2.5, a positive derivative of a function $y = f(x)$ at some point $x = 3$ means that there exists a point $x = a < 3$ such that $f(a) < f(3)$. Similarly, a negative derivative at $x = -1$ indicates that there exists a point $x = c > -1$ such that $f(c) < f(-1)$. In order to find a smaller value, we need to go to the direction that is opposite from the sign of the derivative.

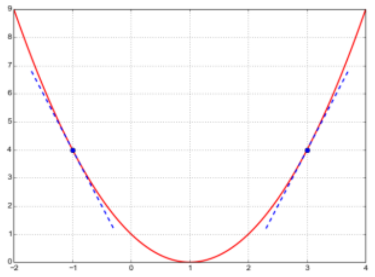


Figure 2.5: Illustration of Gradient on the Convex Curve

The following mathematics proof behind it is shown in [1, chapter4] as follow.

This time we generalize the condition to $f : \mathbb{R}^n \to \mathbb{R}$. First we define a directional derivative of f in direction $\boldsymbol{d}$ of our input data $\boldsymbol{x}$, where $||\boldsymbol{d}|| = 1$, $\boldsymbol{d} \in \mathbb{R}^n$ and $\boldsymbol{x} \in \mathbb{R}^n$, as below.

Theorem 2.4.1. *[1, chapter4]*

$$\frac{\partial}{\partial a}|_{a=0} f(\boldsymbol{x} + a\boldsymbol{d}) = \boldsymbol{d}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{2.4.1}$$

The optimal direction $\boldsymbol{d}^*$ *which provides the most efficient way to reduce* f *to its minimum is the one that can minimize the directional derivative* (2.4.1), *i.e.,*

$$\begin{aligned} \min_{\boldsymbol{d}, \boldsymbol{d}^T\boldsymbol{d}=1} \boldsymbol{d}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) &= \min_{\boldsymbol{d}, \boldsymbol{d}^T\boldsymbol{d}=1} ||\boldsymbol{d}||_2 ||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})||_2 \cos\theta \\ &= \min_{\boldsymbol{d}, \boldsymbol{d}^T\boldsymbol{d}=1} \cos\theta \end{aligned} \tag{2.4.2}$$

where

$$\cos\theta = \frac{\boldsymbol{d} \nabla_{\boldsymbol{x}} f(\boldsymbol{x})}{||\boldsymbol{d}||_2 ||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})||_2} \tag{2.4.3}$$

To minimized (2.4.2), $\boldsymbol{d}$ *needs to have a opposite direction to* $\nabla f(\boldsymbol{x})$

Then we obtain a differential equation of the function f as follows.

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = -\nabla_{\boldsymbol{x}} f(\boldsymbol{x}(t)) \quad t > 0 \tag{2.4.4}$$

In the paper [18], it defines the solution to the above differential equation (2.4.4) as gradient flow. This differential equation is helpful since we can approximate it by the Euler approximation, i.e.,

$$\boldsymbol{x}(t + \epsilon) = \boldsymbol{x}(t) - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}(t)), \tag{2.4.5}$$

where ϵ is the learning rate. The choice of the learning rates indeed have a big impact on the rate of convergence of the training, as illustrated in the figure 2.6. It is an important tuning factor when it comes to the actual implementation and training of the deep learning models.

This approximation then can be updated repeatedly, in the other words, we obtain an iterative function as mentioned in [7, p35] as follow.

$$\boldsymbol{x}_{new} = \boldsymbol{x}_{old} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_{old}), \tag{2.4.6}$$

given an initial condition x_0.

In conclusion, (2.4.6) is the algorithm of gradient descent. Firstly, you have your objective function f. Then you input all the data sets in the algorithm and the result is the last iteration. In the context of the deep learning, the algorithm updates the parameters of the networks to achieve a minimized objective function, referring to the empirical mean of loss function (2.1.3), we can write the corresponding learning algorithm as

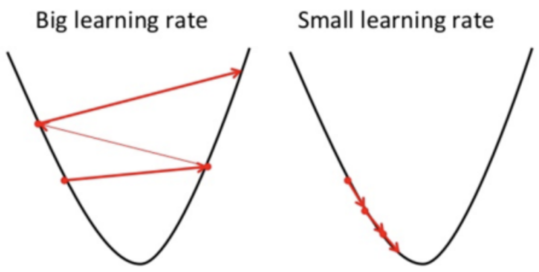


Figure 2.6: Comparison of Convergence for Large Learning Rate and Small Learning Rate

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \epsilon \nabla \mathcal{L}(\boldsymbol{\theta}_i), \tag{2.4.7}$$

where i = 0, 1, . . . , N. Given some initial condition θ_0.

2.4.2 Stochastic Gradient Descent

The SGD is almost the same as the GD, except that we separate the N data sets into q subsets, where $q \ll N$, $N = qk$ and $q, k \in \mathbb{N}+$. Each subset is called a minibatch, and the batch size is k. SGD starts reforming a minibatch B_i by randomly pick up k out of N data sets non-repeatedly, where $i = 1, \ldots, q$. In other words, the samples in the each minibatch are disjoint, i.e.,

$$B_i = \begin{bmatrix} \boldsymbol{x}^1 \\ \boldsymbol{x}^2 \\ \ldots \\ \boldsymbol{x}^k \end{bmatrix},$$

where $\boldsymbol{x}^j \in \mathbb{R}^m$ represents a j^{th} randomly and non-replacement picked up data set from all N data sets, i.e.,

$$\bigcup_{i=1}^{q} B_i = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$$

This random selection procedures is the reason why it is called 'stochastic' gradient descent. SGD will then start to train each minibatch that includes $k \ll N$ data sets instead of the whole N data sets. The objective function for each minibatch becomes as follows.

$$\mathcal{L}_{B_i}(\boldsymbol{\theta}) = \frac{1}{k} \sum_{\boldsymbol{x}^{i,j} \in B_i} l(\boldsymbol{f}(\boldsymbol{\theta} | \boldsymbol{x} = \boldsymbol{x}^{i,j})), \tag{2.4.8}$$

where $i \in \{1, 2, \ldots, q\}$, $j \in \{1, 2, \ldots, k\}$, $\boldsymbol{x}^{i,j} \in B_i$ is the j^{th} data set in B_i that randomly and non-repeatedly drawn from the whole data sets and k is the size of a minibatch.

The k data sets will be trained following the same learning algorithm as (2.4.7) except that the objective function is change to (2.4.8), so the new algorithm for SGD is

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \epsilon \nabla \mathcal{L}_{B_i}(\boldsymbol{\theta}), \tag{2.4.9}$$

Once training of one minibatch B_i is over, the up-to-date trained parameters $\boldsymbol{\theta}_k^i$ are going to passed to the next minibatch, B_{i+1}, as the initial values $\boldsymbol{\theta}_0^{i+1}$ of training. We call it a training epoch when training for all minibatches is done, or when all N data sets have been processed.

Moreover, as mentioned in [7, p36],

$$\nabla \mathcal{L}_{B_i}(\boldsymbol{\theta}) = \frac{1}{k} \sum_{j=1}^{k} \nabla l(\boldsymbol{f}(\boldsymbol{\theta} | \boldsymbol{x} = \boldsymbol{x}^{i,j})) \tag{2.4.10}$$

will be a unbiased estimate of the $\nabla\mathcal{L}(\boldsymbol{\theta})$. This means that the cost for training the data mainly depends on your choice of batch size k, instead of the size N of the whole data sets which consumes large memory space and may cause overfitting.

This can be presented as in Algorithm 1, shown in [7, section3.2, p36].

Algorithm 1: Stochastic Gradient Descent

```
Input  : # epochs: $n_e$
Input  : # batches: $n_b$
Input  : batch size: $b$
Input  : Initial Parameters: $\boldsymbol{\theta}_0$
Input  : learning rate: $\epsilon$
Output: trained network parameters $\boldsymbol{\theta}_{n_b}^{n^e}$
/* loop through all the epochs                                    */
1  for $i$ in {1, 2, ..., $n_e$}: do
2      randomly select $n_b$ samples with $b$ data sets without replacement
       /* set up an initial condition for each epoch                */
3      if $i == 1$ then
4          $\boldsymbol{\theta}_0^i = \boldsymbol{\theta}_0$ // initialize the starting weight matrix
5      else
6          $\boldsymbol{\theta}_0^i = \boldsymbol{\theta}_{n_b}^{i-1}$ // take the last training result as initial condition
7      end
       /* loop through all minibatches                              */
8      for $j$ in {1, 2, ..., $n_b$}: do
9          $\boldsymbol{\theta}_{j+1}^i = \boldsymbol{\theta}_j^i - \epsilon\nabla\mathcal{L}_{B_j}(\boldsymbol{\theta}_j^i)$ // gradient update for each minibatch j
10     end
11 end
12 return $\boldsymbol{\theta}_{n_b}^{n_e}$
```

2.4.3 Adam Algorithm

Proposed by [19], **adam** is an efficient optimization algorithm similar to SGD but applies different learning rates for different model at each time stamp by approximating the first and second moment of the gradient, and the name of 'adam' is coming from the 'adaptive moment estimation' procedure.

The **adam** algorithm presented in [19] is shown in Algorithm 2. We can see from the algorithm that it updates the first and second moments of gradient by computing the *exponential-weighted moving average (EWMA)* of previous gradient, where the first moment estimate represents the direction of update and the second moment estimate is the adjustment to the learning rate at each time stamp. Noted that the direct estimation through the EWMA always generates bias towards zero at the initial time stamp of training, so the algorithm also embeds more steps to utilize the bias, and more details in bias correction can be found in [19, section 3].

Adam also combines the advantages of other popular optimization algorithms, AdaGrad[20] and RMSProp [21]. The benefits of adams are as follows.

1. efficiency in computation since only the first-order derivatives of objective is calculated,
2. ability in dealing with the sparse gradient (when most of the gradient values are zero), which is also the advantage of *AdaGrad*,
3. ability in approximating gradient on noisy and non-stationary objective functions, which *RMSprop* can also perform well on as well.

Nowadays **adam** is popular among the deep learning training tasks and it is also the algorithms that we are going to adapt in this dissertation.

Algorithm 2: Adam algorithm, where all operations on vectors are element-wise computations, and default input $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$

```
Input  : step size: $\alpha$
Input  : exponential decay rates for moment estimates: $\beta_1, \beta_2 \in [0, 1)$
Input  : loss function with respect to parameters $\boldsymbol{\theta}$: $\boldsymbol{l}_t(\boldsymbol{\theta})$
Input  : Initial parameter vector: $\boldsymbol{\theta}_0$
Input  : intial $1^{st}$ moment vector: $\boldsymbol{m}_0 = \mathbf{0}$
Input  : inital $2^{nd}$ moment vector: $\boldsymbol{v}_0 = \mathbf{0}$
Output: updated model parameter vector: $\boldsymbol{\theta}_t$
1  t = 0 // loop from the beginning of the time stamp
2  while $\boldsymbol{\theta}_t$ not converge do
3      t = t + 1;
4      $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\nabla \boldsymbol{l}_t(\boldsymbol{\theta}_{t-1})$ // update biased first moment estimate
5      $\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)(\nabla \boldsymbol{l}_t(\boldsymbol{\theta}_{t-1}))^2$ // update biased second moment estimate
6      $\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1-\beta_1^t}$ // bias-corrected first order estimate
7      $\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1-\beta_2^t}$ // bias-corrected second order estimate
8      $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \frac{\alpha}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon}\hat{\boldsymbol{m}}_t$
9  end
10 return $\boldsymbol{\theta}_t$
```

2.4.4 Back-Propagation

(put algorithems) We have introduced how data flow into the network structure and give out loss values and optimize them. We know from the previous sections that the parameters is updated through determining the direction of change by the gradient. It is essentially a series of derivatives composite together using chain rules. We first write down the *forward path* towards the step at the objective output, and then calculate the derivative of loss function with respect to parameters. [7, section3.3, p39] introduces the back-propagation formula for **FNN** and [1, chapter10, p379] has details in back-propagation in **RNN**.

2.5 Summary

In this chapter, we talked about the general procedures of how a deep learning network is working in 2.1. Then we move to have more details introduction on the three types of deep learning models, **FNN** in 2.2, **LSTM** in 2.3.2 and **GRU** in 2.3.1, and the extra features of capturing time dependencies in the latter two algorithms. The **LSTM** and **GRU** are two models similar in many aspects.

In section 2.4, we talked about how the learning and updating is working under the gradient-based optimization algorithms, including the traditional **gradient descent** method in 2.4.1, the more efficient **stochastic gradient descent** in 2.4.2 and the state-of-the-art algorithm **adam** in 2.4.3. Finally, we talked about the general rule on calculate the gradient of the objective function, namely **back-propagation**, in 2.4.4.

Here we also summarize some features between the **LSTM** and **GRU** network structure as follows since they are both the extension of the standard recurrent neural networks and both have the mechanisms to optionally select information.

As discussed in [22, section3.3], they both have the advantages as shown below.

- remembering important information along the longer time span and keeping the uniqueness of information at each time stamp by the *forget gate* of **LSTM** and the *update gate* of **GRU**,
- the repeatedly multiplication of the non-linear and bounded activation functions allow back-probagation to perform easily without diminishing in gradient.

There are also differences between **GRU** and **LSTM**, as mentioned in [22, section3.3].

- **LSTM** has its own *memory cell* that can only be accessed by the *output gate* while **GRU** exposed the complete information flow without control.
- They also differs in the way of adding/updating new information into the memory. **LSTM** updates the memory based on uncontrolled data from previous time stamp and current data while **GRU** added the new information that depends on the selected data from previous hidden state.

All the network structures introduced in this dissertation are proven to be effective and powerful in many fields, which also motivates this research since we are interested in how deep hedging performs differently under different structures.

Chapter 3

Deep Hedging Method

3.1 Overview

The deep hedging method treats the optimal strategies as a function of the price, i.e.,

$$\phi_t = f_t(\boldsymbol{S_0}, \ldots, \boldsymbol{S_t}; \boldsymbol{\theta}),$$

referred to the procedure introduced in (2.1.1). In this dissertation, $\{\boldsymbol{\phi}_t\}_{0 \leq t \leq T}$ have been represented by **FNN**, **LSTM** and **GRU**. They optimized $\{\hat{\phi}_t\}_{0 \leq t \leq T}$ through minimizing the objective function regarding the *profit and loss*. Since it is the unsupervised learning, it does not require 'labels', or reference strategies to compare to, instead, it needs a function that can measure the *PnL*. In this dissertation, we choose the *exponential utility function* to see how *PnL* perform under certain risk-averse level. In other words, the performance of *PnL* will be the loss value in the context of deep learning, referred to (2.1.2), i.e.,

$$loss_i = U(PnL),$$

where U is the measure of performance for *PnL*. Consequently, our objective function will become

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= \tfrac{1}{N} \textstyle\sum_{i=1}^{N} U(PnL_i), \\ &= \tfrac{1}{N} \textstyle\sum_{i=1}^{N} U(PnL_i(\boldsymbol{\theta} | \boldsymbol{S_{0,i}}, \ldots, \boldsymbol{S_{T,i}})), \end{aligned}$$

where N is the number of price paths.

Another advantage of the deep hedging method is that we can easily incorporate the market frictions by editing the *PnL* function. We use the *proportional transaction cost* to capture the cost of execution at each time stamp, so all we need to do is to subtract the cost from the profit at the maturity when calculate the *PnL*, i.e.,

$$PnL = \text{hedge portfolio} - \text{contingent claim} - \text{transaction cost}$$

3.2 Delta Hedge under Black-Scholes Model

Hedging is a strategy to offset the risk of the price movement of the underlying asset of a financial instrument. Before looking into the deep hedging method, it is necessary to have a brief review on the traditional delta hedge under the *Black-Scholes* model.

3.2.1 Black-Scholes Model

In Black-Scholes model [2], the underlying stock prices $(S_t)_{t \geq 0}$ can be presented as a stochastics differential equation as follows.

$$\mathrm{d}S_t = S_t(r\mathrm{d}t + \sigma \mathrm{d}W_t), \qquad S_0 > 0, \tag{3.2.1}$$

where $(W_t)_{t \geq 0}$ is the Brownian motion on $(\Omega, (\mathcal{F}_t), \mathbb{P})$, $r \geq 0$ is the risk-free interest rate, $\sigma > 0$ is the volatility of the underlying. Then a european call option can be priced as

$$C(S,t) = S\mathcal{N}(d_+) - K\mathrm{e}^{-rt}\mathcal{N}(d_-), \tag{3.2.2}$$

where $\mathcal{N}$ is the cdf of the standard normal distribution and

$$d_\pm := \frac{\log(S_0\mathrm{e}^{rt}/K)}{\sigma\sqrt{t}} \pm \frac{\sigma\sqrt{t}}{2}.$$

It is a model under the assumptions of

- no transaction costs,
- stocks not paying dividends,
- constant and known risk-free rate and volatility,
- returns are normally distributed.

The assumptions make any strategies rely on it unrealistic. However, it provides analytical delta hedging strategies under the cost-free and risk-neutral environment, which are useful to be a benchmark for the deep hedging strategies when λ is large and cost rate equals 0.

3.2.2 Delta Hedge

Suppose that a the *value process* for the underlying is

$$V_t \quad = \phi_t^s S_t + \phi_t^b B_t, \tag{3.2.3}$$

where ϕ_t^b, ϕ_t^s on $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{0\le t\le T}, \mathbb{P})$ are locally bounded and $\mathcal{F}_t-$adapted, representing the amount that held in stock and bond, respectively.

Let $C(S,t)$ be the price of a *attainable* [1] *contingent claim* [2] Z. Then by the *non-arbitrage principle* and *law of one price*, we have the equation

$$\begin{aligned} C(S,t) \quad &= \quad V_t \\ &\overset{\text{feynman-kac}}{=} \mathbb{E}^{\mathbb{Q}}[e^{-(T-t)}V_T|\mathcal{F}_t] \\ &\overset{V_T=Z}{=} \mathbb{E}^{\mathbb{Q}}[e^{-(T-t)}Z|\mathcal{F}_t] \end{aligned} \tag{3.2.4}$$

$$\Leftrightarrow \mathrm{d}e^{-rt}C(S,t) \quad = \quad \mathrm{d}e^{-rt}V_t. \tag{3.2.5}$$

By simplifying the equation in (3.2.5), we will get the delta hedging strategy,

$$\phi_t^s = \frac{\partial C}{\partial S}(S,t), \tag{3.2.6}$$

not only representing the sensitivity to the underlying price change but also the amount of individual's holding of the stock at time t.

We are going to hedge the *european call option* in this dissertation. In our case, the *Black-Scholes* delta-dedge strategy becomes,

$$\phi_t^{BS} = \mathrm{d}e^{-rt}V_t, \tag{3.2.7}$$

where d_+ is defined in (3.2.2).

This means that we know the exact amount of holdings at every time stamp. As long as we follow this strategies all the time, it guarantees that we will receive same amount of values as the contingent claim at maturity. In other words, if we are the selling the contingent claim Y to the clients and receive a premium of $C(S, t = 0)$ in the beginning, we expect to pay $Y = f(S_0, \dots, S_T)$, the agreed contract value, back at the maturity to clients. In order to avoid the risk of volatile

[1] A contingent claim Y is attainable if there exists a self-financing strategy ϕ such that $V_T(\phi) = Y$ [23, p23]

[2] A contingent claim $Y = f(S_0, \dots, S_T)$, where $f : \mathbb{R}^{(T+1)\times d} \to \mathbb{R}$, where d is the number of assets, for the maturity T is any square-integrable and positive random variable in $(\Omega, \mathcal{F}_t, \mathbb{P})$ [23, p19]

underlying price, we reinvest our premium to buy the underlying and the risk-free assets. The above strategies will instruct us how much should we hold before maturity. Theoretically, under the *Black-Scholes* assumption mentioned in 3.2.1, by following the strategies, our re-investment will pay us back exactly the same amount as the agreed payment in the contingent claim Z.

Note that if we are under the discrete-time assumptions, i.e.,

$$S_{t\,i} = S_{t_{i-1}} + S_{t\,i}(r\Delta t_i + \sigma \Delta W_{t\,i}), S_0 > 0,$$

where $i = 1, \dots, T$ represents the index of the time stamp. the delta hedging would not be continuous either, as a result, it would be the approximations of the corresponding continuous-time delta-hedging strategies, i.e.,

$$\phi_t^s = \frac{\Delta C(S, t_i)}{\Delta S_{t\,i}}(S, t),$$

where the perfect hedge can not be achieved.

Now, we have the analytical Black-Scholes delta hedging strategies for the european call option. Later, we will use this formula to make comparison to the deep hedging strategies.

3.3 Deep Hedging Setting

For deep hedging, we consider a discrete-time market with finite number of time stamps T with d risky assets, where $T, d \in \mathbb{N}+$. The underlying prices will then be $\{\boldsymbol{S}_t\}, t = 0, \dots, T$ and $\boldsymbol{S}_t \in \mathbb{R}^d$. They form an adapted, non-negative stochastic process on the finite probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}, \mathbb{P})$, where $\Omega = \{\omega_1, \dots, N\}$ for some $N \in \mathbb{N}$. The corresponding self-financing and adapted trading strategies, or the amount of portfolio held at t, is denoted as ϕ_t. For simplicity, assume the risk-free rate $r = 0$ and the number of underlying $d = 1$, then our value process in discrete time becomes

$$V_t = V_0 + \sum_{i=0}^{t-1} \phi_i \cdot (S_{i+1} - S_i), \tag{3.3.1}$$

where V_0 is our initial wealth.

This can be seen as the same setting as the discrete Black-Scholes model. We are now going to add the market friction to it.

3.3.1 Proportional Transaction Cost

We assume that for execution at timestamp $t = 0, \dots, T$, it charges the transaction cost proportionally at a constant rate k to the transaction amount. We define the transaction cost at maturity as follows.

$$c_T(\phi; S, k) = \sum_{i=1}^{d} k_i(|\pi_{0,i}|S_{0,i} + \sum_{t=2}^{T} |\pi_{t-1,i} - \pi_{t-2,i}|S_{t-1,i} + |\pi_{T-1,i}|S_{T,i}), \tag{3.3.2}$$

where k_i is the cost rate for the i^{th} asset and $d = 1$ in our settings.

Apply the transaction cost $c(\pi; S)$ to our value process in (3.3.1), then it becomes

$$V_T = V_0 + \sum_{i=0}^{T-1} \phi_i \cdot (S_{i+1} - S_i) - c_T(\phi; S, k), \tag{3.3.3}$$

where $t = 1, \dots, T$ and $V_0 \in \mathbb{R}$.

3.3.2 Profit and Loss

The *profit and loss* at maturity is the difference between the current wealth V_T and the payoff of the contingent claim $Z = g(\boldsymbol{S_0, \dots, S_T})$, where $g : \mathbb{R}^{(T+1)\times d} \to \mathbb{R}$, i.e.,

$$\begin{aligned} PnL_Z(V_0, \phi; S, k) &= V_T - Z \\ &= V_0 + \textstyle\sum_{i=0}^{T-1} \phi_i \cdot (S_{i+1} - S_i) - c_T(\phi; S, k) - Z \\ &= V_0 + V(\phi; S) - c_T(\phi; S, k) - Z, \end{aligned} \tag{3.3.4}$$

where

$$V(\phi; S) = \sum_{i=0}^{T-1} \phi_i \cdot (S_{i+1} - S_i) - c_T(\phi; S, k), \tag{3.3.5}$$

is the change in portfolio holding.

3.3.3 Convex Risk Measure Deep Hedging Strategies

As mentioned in section 3.1, we need a measure to quantify the performance of PnL that takes into account the risk arising from the position. The average of the aggregate performance is the objective $\mathcal{L}(\boldsymbol{\theta})$ of our model. The network will then train the network through minimizing the objective.

In the original work of the deep hedging from [6, section 3], it provides a *convex risk measures* definition as follows.

Definition 3.3.1. Assume $X, X_1, X_2 \in \mathcal{X}$, representing the current positions, where $\mathcal{X} : \Omega \to \mathbb{R}$, is set of all real-value random variable. We define a convex risk measure $\rho : \mathcal{X} \to \mathbb{R}$ if:

- **Monotone Decreasing**: if $X_1 \geq X_2$, then $\rho(X_1) \leq \rho(X_2)$
  This means that the position with smaller loss and larger profit are favorable.
- **Convex**: $\rho(\alpha X_1 + (1-\alpha)X_2) \leq \alpha\rho(X_1) + (1-\alpha)\rho(X_2)$, where $\alpha \in [0, 1]$
  it indicates that the diversified position gives a better performance.
- **Cash-Invariant**: $\rho(X + c) = \rho(X) - c$, where $c \in \mathbb{R}$
  It implies that the performance will of the strategies will improve the same amount as the cash we add into the position, and $c = \rho(X)$ is the least amount that we need to add into the position if we want $\rho(X + c) \leq 0$ since $\rho(X + \rho(X)) = 0$.

If $\rho(0) = 0$, we call the measure is *normalized*.

We would like to measure our risks related to the predicted hedge strategies following the above properties. In other words, the convex risk measure provides a standard for us to find strategies such that the risks towards our final gain, PnL, would be minimized under such standard. This leads to an optimization problem.

Definition 3.3.2. [6, p7] Let T be number of time stamps, and $T \in \mathbb{N}+, t = 0, \dots, T-1$. $\{\phi_t\}_t$ is set of self-financing and adapted hedging strategies and $\phi_t \in \mathbb{R}$. $c_T(\cdot)$ is the accumulated proportional transaction cost at the maturity defined in (3.3.2). Under a *convex risk measure* $\rho : \mathcal{X} \to \mathbb{R}$, and for $X \in \mathcal{X}$ where $\mathcal{X} : \Omega \to \mathbb{R}$, define the optimization problem as

$$\pi(X) := \inf_{\phi} \rho(X + \sum_{t=0}^{T-1} \phi_t \cdot (S_{t+1} - S_t) - c_T(\phi; S, k)),$$

which has also been proved in [6, proposition 3.2] that it is *monotone decreasing* and *cash-invariant*.

In the meanwhile, if we think of the objective functions $\mathcal{L}(\boldsymbol{\theta})$ (2.1.3) as the risk measure of the PnL (3.3.4), i.e.,

$$\mathcal{L}(\boldsymbol{\theta}) = \rho(PnL_Z(V_0, \phi(\boldsymbol{\theta}))),$$

then the optimization problem π becomes exactly the deep learning optimization problem. By tuning the deep learning models parameters, $\boldsymbol{\theta}$, we find a set of strategies, $\{\phi_t^*\}_{t=0,\dots,T-1}$ such that

$\phi_t^* = f(\boldsymbol{S_0}, \ldots, \boldsymbol{S_t}; \boldsymbol{\theta^*})$ and f as presented in (2.1.1) stands for a deep learning network, minimizing the objective function $\mathcal{L}(\boldsymbol{\theta})$.

$\{\phi_t^*\}_{t=0,\ldots,T-1}$ are the so-call *deep hedging strategies*. It is the minimizer of the objective function under a convex measure. We can re-written the optimization problem in the context of deep hedging as follows.

$$\begin{aligned} \pi^{DH}(X) &:= \inf_{\phi(\boldsymbol{\theta})} \mathcal{L}(\boldsymbol{\theta}) \\ &= \inf_{\phi(\boldsymbol{\theta})} \rho(PnL_Z(X+Z;S,k), \phi(\boldsymbol{\theta}); S, k)), \end{aligned} \tag{3.3.6}$$

Definition 3.3.3. (deep hedging strategies) Let π^{DH} be a optimization problem as shown in (3.3.6). Define the deep hedging strategies $\{\phi_t^*\}_{t=0,\ldots,T-1}$ such that they are the solutions to $\pi^{DH}(V_0 - Z)$, i.e,

$$\begin{aligned} \pi^{DH}(V_0 - Z) &= \inf_{\phi(\boldsymbol{\theta})} \mathcal{L}(\boldsymbol{\theta}) \\ &= \rho(PnL_Z(V_0; S, k), \phi^*; S, k)) \\ &= \rho(V_0 - Z + \textstyle\sum_{t=0}^{T-1} \phi_t^* \cdot (S_{t+1} - S_t) - c_T(\phi; S, k)), \end{aligned}$$

where V_0 is the initial wealth, Z is a contingent claim and PnL is referred to (3.3.4).

In this dissertation, we choose the loss function l (2.1.2) which measuring the risk deriving from the predicted strategies to be the negative of the exponential utility function U, i.e., $l(x) = -U(x)$, as shown in (3.3.11), and the objective is to minimize the empirical mean of the loss functions, i.e,

$$l\left(PnL_Z(V_0, \phi(\boldsymbol{\theta}); S, k)\right) := -U\left(PnL_Z(V_0, \phi(\boldsymbol{\theta}); S, k)\right),$$

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}[l(PnL_Z(V_0, \phi(\boldsymbol{\theta}); S^{(i)}, k))].$$

The optimization problem in the context of deep hedging method applying in our dissertation will be

$$\pi^{DH}(X) = \inf_{\phi(\boldsymbol{\theta})} \frac{1}{N} \sum_{i=1}^{N} l(PnL_Z(X+Z;k), \phi(\boldsymbol{\theta}); S^{(i)}, k)), \tag{3.3.7}$$

where $PnL_Z(\cdot)$ is introduced in (3.3.4), Z is the attainable contingent claim (see footnote 1) and $\phi(\boldsymbol{\theta})$ represents the outputs from deep learning models introduced in chapter 2.

3.3.4 Indifference Pricing

Under the *Black-Scholes* assumption, i.e., the complete market without market frictions, the price of the call option is simply the premium of the contingent claim Z as shown in (3.2.4) by setting $t = 0$, i.e.,

$$\begin{aligned} V_0 &= C(S, 0) \\ &= \mathbb{E}^{\mathbb{Q}}[e^{-(T-t)} Z], \end{aligned} \tag{3.3.8}$$

and we have

$$\begin{aligned} PnL_Z(V_0, \phi; S) &= V_T - Z \\ &= V_0 + V(\phi; S) - Z \\ &= 0 \end{aligned} \tag{3.3.9}$$

by combining (3.3.4) and (3.2.4) and substituting the proportional transaction cost rate $k = 0$.

However, after applying the transaction cost to PnL, the market becomes incomplete and the fair price of the contingent claim is no longer $C(S, 0)$. Pricing is difficult under the analytical methods in this case, especially for some more complicated exotic products. Fortunately, [6] provides a solution to the incomplete market pricing problem under a convex measure.

Definition 3.3.4 (indifference price). [6, p8]

Let π be a measure defined in Definition 3.3.2 and Z be a attainable contingent claim mentioned in footnote 1. The indifference price for a contingent claim Z is $p(Z) = p_0$ such that it satisfied the equation $\pi(-Z + p_0) = \pi(0)$. As a result,

$$p(Z) := \pi(-Z) - \pi(0),$$

due to the *cash-invariant* property of π mentioned in the Definition 3.3.2.

It means that if we sell the contingent claim Z at $p = p(Z)$, applying the corresponding optimal strategies, our expected loss will be the same as optimally executing the trade under U without the contingent claim Z, i.e.,

$$\inf_{\phi} \mathbb{E}[l(V(\phi; S))] = \inf_{\phi} \mathbb{E}[l(p + V(\phi; S) - Z)], \tag{3.3.10}$$

where $V(\phi; S)$ is defined in (3.3.5).

In other words, the indifference price $p(Z) = p$ of the contingent claim Z should be the *minimal* amount of cash that can cover the potential loss from Z at a given risk-preference, i.e., a convex risk measure by our choice, and we call this *minimal* amount of compensation the *indifference price*.

Exponential Utility indifference Pricing

In this dissertation, we follow the same setting as [5]. Choose our loss function as the negative *exponential utility function*, i.e.,

$$l(x) = -U_{\lambda}(x) = \frac{1}{\lambda} e^{-\lambda x} \tag{3.3.11}$$

which is a convex function. Plug it into (3.3.10), we get

$$\begin{aligned} \inf_{\phi} \mathbb{E}[l(V(\phi; S))] &= \inf_{\phi} \mathbb{E}[\tfrac{1}{\lambda} e^{-\lambda(p+V(\phi;S)-Z)}] \\ &= e^{-\lambda p} \inf_{\phi} \mathbb{E}[l(V(\phi; S) - Z)]. \end{aligned}$$

Extracting the indifference price p, then

$$p = -\frac{1}{\lambda} \log \left(\frac{\inf_{\phi} \mathbb{E}[l(V(\phi; S))]}{\inf_{\phi} \mathbb{E}[l(V(\phi; S) - Z)]} \right). \tag{3.3.12}$$

The denominator

$$\inf_{\phi} \mathbb{E}[l(V(\phi; S) - Z)] = \inf_{\phi} \mathbb{E}[l(PnL_Z(0, \phi; S, k))]$$

and numerator

$$\inf_{\phi} \mathbb{E}[l(V(\phi; S))] = \inf_{\phi} \mathbb{E}[l(PnL_0(0, \phi; S, k))]$$

that inside the log function in (3.3.12) can be fitted into the deep learning optimization problem framework as shown in (3.3.7). In other words, the indifference price p calculated from the exponential utility function can be obtained by training the two independent deep learning networks with different objective functions.

Suppose that the ϕ_d^* and ϕ_n^* are the optimal deep hedging strategies for the denominator and numerator respectively. We can approximate the exponential utility indifference price as

$$\hat{p} = -\frac{1}{\lambda} \log \left(\frac{\frac{1}{N} \sum_{i=1}^{N} [l(PnL_0(0, \phi_d^*; S, k))]}{\frac{1}{N} \sum_{i=1}^{N} [l(PnL_Z(p, \phi_n^*; S^{(i)}, k))]} \right). \tag{3.3.13}$$

Moreover, if the underlying follows a martingale process, the initial wealth p in the exponential utility function is essentially a scaling factor of the optimization process, as a result, the choice of p does not affect the training.

3.4 Summary

In this chapter, we introduced the deep hedging methodology. In the section 3.1, we present the overall procedure of the deep hedging, linking the financial concepts to the context of deep learning. For example, the hedging strategies are represented by the network, the exponential utility function that measure the PnL is the loss function for each price path, and the emprical mean of the loss functions is our objective function that will be minimized by the gradient-based optimization algorithm.

We also have a brief review on the delta hedging under the Black-Scholes assumption in section 3.2. This will be our benchmark to compare to the deep hedging strategies later.

Then we move on to introduce the details of the deep hedging in section 3.3. Including the value function on the discrete market setting, and how we define the PnL functions (section 3.3.2) after incorporate the proportional transaction cost (section 3.3.1). After that, we also introduce the convex measure to examine the performance of the PnL under certain level of risk preference, and it is the measures that used in the loss and objective functions. With these background knowledge prepared, we can understand how the deep learning network obtain the optimal stategies. The last part introduce a useful method to price the over-the-counter derivatives, indifference pricing (section 3.3.4).

Chapter 4

Training Methedology

Now, let us introduce how we implemented the deep hedging strategies in our dissertation. The overall procedures are shown as follows.

1. *Data Simulation*:

   Create a Black-Scholes market simulator, and generate two sets of data, the training underlying set $\boldsymbol{S}_{train}$ and the validation set $\boldsymbol{S}_{val}$. For each data set, it has $N = 100,000$ simulated paths and $T = 100$ time steps;

   More details are in the section 4.1.

2. *Tailor Input*:

   Tailor the data set $\boldsymbol{S}_{train}$ and $\boldsymbol{S}_{val}$ into $\{\boldsymbol{X}_{train,model}\}_{\text{model}}$ and $\{\boldsymbol{X}_{val,model}\}_{\text{model}}$, where model $\in$ {LSTM, FNN, GRU}, to satisfy the input requirement for different model structures;

   More details are in the section 4.2.

3. *Model Implementation*:

   (a) *Deep Learning Model Structures Implementation*:
   Implement structures $\boldsymbol{f}_{FNN}$, $\boldsymbol{f}_{LSTM}$and $\boldsymbol{f}_{GRU}$, setting inputs, activation functions, layers, outputs,and units. The output of $\boldsymbol{f}_{model}$ will be the predicted hedging strategies such that $\boldsymbol{\phi}_{model} = \boldsymbol{f}_{model}(\boldsymbol{X}_{train,model}; \boldsymbol{\theta})$.
   More details of the model structures are in section 4.2.

   (b) *General Loss Functions Implementation*:
   Implement the general loss functions $l(Z, p, \boldsymbol{\phi}_{model}, k)) = \frac{1}{\lambda} e^{\lambda \cdot PnL(Z,p,\boldsymbol{\phi}_{model},\lambda,k)}$, where $PnL(Z, p, \boldsymbol{\phi}_{model}, k)$ is defined in 3.3.4. Note that the parameters $\boldsymbol{\phi}_{model}$ are the outputs of the network, it is not editable while training taking place.

   More details in the implementation of the loss functions are in the section 4.2.3.

   (c) *Optimizer selection*

   We chose the adam optimizer, as introduced in section 2.4.3, for all models. However, the default learning rates were set to different values for different model structures.

4. *Data Training*:

   Once the model structures have been built, we are able to input our data and start training. There are two sets of results that we are interested in, the optimal hedging strategies, as defined in Definition 3.3.3, and the fair price of the contingent claim Z, as defined in Definition 3.3.4. To obtain the two sets of results, it requires different loss functions. Moreover, different models response differently regarding the data, so different training strategies are applied to different deep learning model structures. As a result, the following steps are adapted to solve the problems.

   - *set loss function for optimal strategies training* as in section 4.2.3;
   - *set loss function for indifference pricing training* as in section 4.2.3;
   - *set FNN training strategies*;

- *set LSTM and GRU training strategies*;
- *set explanatory variables for each model and start training*;

5. *Results Visualization*

   We obtain the deep hedging strategies and indifference prices under various conditions. There are 4 different cost rates, 4 different risk-averse coefficients λ and 3 deep learning models, resulting in 48 scenarios in total. We are going to present them regarding the models, i.e., for each deep learning model, we illustrate the results under different cost rates and indifference prices. More details on the results are shown in chapter 5.

4.1 Data Generation

We use the Black-Scholes data simulator to generate the underlying assets path. The stochastic differential equation (SDE) of the price process is shown in (3.2.1), by solving the SDE, we obtain the pricing process

$$S_t = S_0 \exp\left[(r - \frac{\sigma^2}{2})t + \sigma W_t\right],$$

where $\sigma > 0, r > 0, S_0 > 0$ and $t > 0$.

Let $\xi_1, \ldots, \xi_T$ be mutually independent standard normal distributed random variables, $N(0,1)$. Let T be the number of time steps. We have the discrete-time Black-Scholes price process as

$$S_t = S_0 \exp\left[(r - \frac{\sigma^2}{2})\frac{t}{T} + \frac{\sigma}{\sqrt{T}}\sum_{i=0}^{t}\xi_i\right], t = 0, \ldots, T. \tag{4.1.1}$$

This is our market simulator.

in the experiments, we set the number of paths as $N = 100,00$ and the number of time steps as $T = 100$, risk-free rate $r = 0$, initial price $S_0 = 1$, price volatility $\sigma = 0.5$. By generating $\boldsymbol{\xi}_{train} = (\xi_{i,j})_{N\times T}$, where $\xi_{i,j}$ follows $N(0,1)$, we created our simulated underlying $\boldsymbol{S}_{train} = (S_{i,j})_{N\times(T+1)}$. Following the exact same setting, we generated another validation underlying matrix as $\boldsymbol{S}_{val} \in \mathbb{R}^{N\times(T+1)}$ by generating another set of standard normal random variables $\boldsymbol{\xi}_{val} \in \mathbb{R}^{N\times T}$.

Note that since we set $r = 0$ for simplicity, our simulations become a driftless martingale process.

4.2 Models Settings

As introduced in chapter 3, our strategies can be represented by the networks as a function of previous price stamps, i.e., for each path i, we have

$$\phi_t^{(i)} = f_t(S_0^{(i)}, \ldots, S_t^{(i)}; \boldsymbol{\theta}).$$

We are going to specify how structure of each model is implemented in our experiments. We used the Keras package in the Tensorflow to create, compile and train the deep learning models.

4.2.1 FNN Implementation

Thanks to the *markov property* of the Black-Scholes price, for the **FNN** model structure, the network structure can simplify to

$$\phi_t^{(i)} = f(\frac{t}{T}, S_t^{(i)}; \boldsymbol{\theta}_t), t = 0, \ldots, T,$$

Previously, we have defined the structure of the **FNN** as (2.2.4), following the definition, we implemented the **FNN** structure as

$$f_{FNN} \in \mathcal{N}_4(2, 100, 100, 100, 1; ReLU, ReLU, ReLU, sigmoid),$$

meaning that the network takes in a 2-dimensional input at each time stamp, i.e., input $\boldsymbol{X} = (X_{i,j})_{N\times T}$ where $X_{i,j} = (\frac{j}{T}, S_j^{(i)})$, $i = 1,\dots,N$ and $j = 0,\dots,T-1$. Our implemented structure has 3 hidden layers, for which each of them has 100 units, and equipped with $ReLU$ (2.2.8) as the activation function. The output layer always has a 1-unit output since it represents the the hedge ratio for a underlying asset. The output is bounded by the sigmoid (2.2.6) function as the Black-Scholes hedge ratios for call options are always in between 0 and 1.

4.2.2 LSTM and GRU Implementation

LSTM and **GRU** have very similar overall structures since they are both the extensions of the **RNN**, except for how information proceeds within the cell state. As mentioned in section 2.3.3, **RNN** has the vanishing and exploding gradient problem, so we choose the two more stable recurrent structures in our dissertation.

In our implementation, **LSTM** and **GRU** are sharing the same structure in terms of being time-dependent based models.

Let us now represent a single LSTM layer network structure as

$$f_{LSTM} \in \mathcal{R}_2(T, 10, 1; LSTM, sigmoid)$$

meaning that for each sample path i, our **LSTM** networks takes a whole sample path $\{S_t^{(i)}\}_{t=0,;T-1}$ as input.That is, the input data $\boldsymbol{X} = (X_{i,j})_{N\times T} = (S_{i,j})_{N\times T} = \boldsymbol{S}$, where $i = 1,\dots,N$ and $j = 0,\dots,T-1$.

Then the data flow into the LSTM layer, generating hidden states with dimension 10, which share the same size as the output at each time stamp, as introduced in section 2.3.2. As a result there will be $T = 100$ outputs generated by the $LSTM$ layer and each of the output has length 10.

Then all of the 100 output go through the sigmoid (2.2.6) output layer, transforming each of the output from dimension 10 to 1, representing the hedge ratios.

Similarly, let us define the single-layer **GRU** model in the dissertation as

$$f_{GRU} \in \mathcal{R}_2(T, 10, 1; GRU, sigmoid),$$

How data flow through the structure will be the same as **LSTM** except that it has a **GRU** layer, and the process has been clarified in 2.3.3.

4.2.3 Loss Functions Implementation

From (3.3.7) and (3.3.11), we have our loss functions in general as function of 4 inputs, i.e., for each sample path $\{S_t^{(i)}\}_{t=0,\dots,T-1}$, $i = 1,\dots,N$, we have

$$loss(Z, p, \{\phi_t^{(i)}\}_{t=0,\dots,T-1}, \lambda, k) = \frac{1}{\lambda}\exp\left[-\lambda \cdot PnL(Z, p, \{\phi_t^{(i)}\}_{t=0,\dots,T-1}, k)\right],$$

where PnL is the same as (3.3.4).

Now, following the Definition 3.3.3, the loss function for the deep hedging strategies is

$$loss_{dh} = loss(Z_{call}, p_{dh}, \{\phi_t^{(i)}\}_{t=0,\dots,T-1}, \lambda, k),$$

where $Z_{call} = (S_T - K)^+$, p_{dh}[1] is the initial wealth, $\{\phi_t^{(i)}\}_{t=0,\dots,T-1}$ are the outputs from the deep hedging network, and λ and k are the risk preferences and cost rates that we are going to set for different scenarios.

Similarly, for the indifference pricing training, referred to section 3.3.4 and (3.3.13), we have the loss function for the numerator as

$$loss_{ipr,n} = loss(0, p_{ipr,n} = 0, \{\phi_t^{(i)}\}_{t=0,\dots,T-1}, \lambda, k),$$

[1]Note that the subscript dh stands for deep hedging.

and the loss function for the denominator is set to

$$loss_{ipr,d} = loss(Z_{call}, p_{ipr,d} = 0, \{\phi_t^{(i)}\}_{t=0,\dots,T-1}, \lambda, k),$$

where ipr represents the indifference price, d represents the loss function for the denominator, and n represents the loss for the numerator.

Since our underlying prices are the martingale, our initial wealth p, in this case, does not affect our training, as mentioned in the section 3.3.4. For simplicity, all of them are set to 0 in our implementation, i.e., $p_{dh} = p_{ipr,n} = p_{ipr,d} = 0$.

4.3 Training Settings

The deep learning models optimize the models following the *adam* algorithm as introduced in section 2.4.3. The following sections will present the procedures and parameters for the optimization process.

4.3.1 FNN Training

We set batch size = 256, $epochs = 40$. There are no special procedures for the training with **FNN** model, it follows the adam algorithm as introduced in section 2.4.3. Under this setting, the results are meaningful and as expected.

4.3.2 LSTM and GRU Training

However, the procedures for **LSTM** and **GRU** are more complicated. We first trained the model with a faster learning rate. Then, turned down the learning rate and iterated the training with the same slower one 10 times. The training procedure is shown in the appendix , Algorithm 3. We set the parameters as follows.

The batch size = 256, epochs = 55, and the faster learning is set to 0.01 while the slower learning rate is 0.001.

4.4 Training

By setting up the models and training parameters, we can start the training. Firstly, we are interested in how the cost rates and λ impact our strategies and indifference prices. Secondly, we would like to examine how the results are performing under different deep learning models. We set proportional cost rates k to be $0, 0.05\%, 0.5\%$ and 5%, respectively, involving the situations where the market is complete and with relatively unrealistic higher costs. We have also set our risk-averse coefficient λ to be $0.1, 1, 5$ and 10. In conclusion, there will be 48 sets of results, i.e., results for each model under each cost rate and risk-averse level, generated and all of them can be found in chapter 5.

4.5 Statistical Tests on Result

4.5.1 Kolmogorov Smirnov Test

From the histograms of the $PnLs$ under different training models at the same risk-averse level and cost rate, we can hardly tell if they are significantly different. In this case, the first step is to test if, at the same level of cost rate and λ, different training modes produce the same $PnLs$ distributions. If they produce different results statistically different, then we can further investigate how they are different from each other.

The *Two-sample Kolmogorov Smirnov Test* [24] (KS test) is a non-parametric test on whether the two samples are coming from the same distribution. The null hypothesis H_0 is that the two samples are from the same distributions. In the context of our dissertation, the two samples will be the *PnLs* calculated under different model structures at a specific level of cost rate and risk preference.

The test statistics for the two-sample KS-test is

$$D_n = \sup_x |F_{\text{moded}_1,N}(x) - F_{\text{model}_2,N}(x)|,$$

where N is the number of simulated sample paths, *model* represents the model structures such as **FNN**, **LSTM** and **GRU**, and $F_{\text{mode},N}(x)$ is the empirical cumulative distribution function of *PnLs* resulting from training the N sample paths under a specific model structure, i.e.,

$$F_{\text{model},N}(x) = \frac{\text{number of PnLs under a deep learning model} \leq x}{N}.$$

The rejection of the null hypothesis H_0 happens at α level of significance if

$$D_N > \sqrt{-\log(\frac{\alpha}{2}) \cdot \frac{1}{N}}.$$

4.5.2 One-sample Signe Test

If we are knowing that the *PnLs* resulting from different deep learning models coming from different distributions, we use the *one-sample sign test* to test if the median of *PnLs* for each scenario is non-negative, i.e., for each deep learning model we present a table containing the p-values for the alternative hypothesis,

$$H_\alpha : median(\{PnL_{\text{model},i}\}_{i=1,\dots,N}) < 0,$$

under different cost rates and risk preferences λ's.

We know that our data are not normally distributed or symmetric. The sign test does not assume a prior known distribution of our data, but the disadvantage is that it is not as efficient as other tests on the median such as Wilcoxon signed-rank test which requires symmetric distribution of data.

The one-sample sign test first count for all values in a sample that are above the null hypothesis, median $= 0$, so we have a Bernoulli random variable for each count such that

$$I_i = \mathbf{1}_{PnL_i > 0},$$

for $i = 1, \dots, N$. Then summing all our counts together, i.e.,

$$Y = \sum_{i=1}^{N} I_i,$$

where Y follows the binomial distribution $Bin(N, p_Y)$. The problem is now reformed as a binomial test, determining if $p_Y = p_0$ where $p_0 = 0.5$, as our goal is to find whether Y, the number of values greater than 0 (our null hypothesis of median), is half of our sample size. For a large sample size, the test statistics would be Z $= \frac{Y - p_0 N}{\sqrt{N p_0 (1-p_0)}}$.

4.5.3 Two-sample Wilcoxon Signed-rank Test

The *PnLs* are generated based on the same validation data set but different learning models. The dependent sample test, a two-sample Wilcoxon signed-rank test, is a suitable choice to compare the differences in the medians among different models. we are going to conduct the test on the alternative hypothesis

$$H_\alpha : median(\{PnL_{model1,i}\}_{i=1,\dots,N}) < median(\{PnL_{model2,i}\}_{i=1,\dots,N}).$$

The test first calculates the differences between the two samples

$$\{(PnL_{model1,i} - PnL_{model2,i})\}_{i=1,\dots,N},$$

and then orders its absolute differences and assign the ranks ascendingly, denoted R_i. Next, assign back the sign of its corresponding differences in the sample to the rank R_i, i.e.,

$$\mathcal{R}_i^{signed} = sgn(PnL_{model1,i} - PnL_{model2,i}) \cdot R_i,$$

where sgn is the sign function. Sum up all the $\mathcal{R}_i^{signed}$ with negative signs and W^- is its absolute value, i.e.,

$$W^- = \sum_{\{\mathcal{R}_i^{signed} : \mathcal{R}_i^{signed} < 0\}} |\mathcal{R}_i^{signed}|.$$

Similarly, add up all the $\mathcal{R}_i^{signed}$ with positive values, denoted W^+. Finally, we have the test statistics $W = min(W^-, W^+)$.

The tables in the following sections show the p-values for the two-sample Wilcoxon signed-rank test on sets between

- $\{PnL_{LSTM,i}\}_{i=1,\ldots,N}$ and $\{PnL_{FNN,i}\}_{i=1,\ldots,N}$,
- $\{PnL_{LSTM,i}\}_{i=1,\ldots,N}$ and $\{PnL_{GRU,i}\}_{i=1,\ldots,N}$,
- $\{PnL_{LSTM,i}\}_{i=1,\ldots,N}$ and $\{PnL_{GRU,i}\}_{i=1,\ldots,N}$.

For each pair of comparison, we are going to demonstrate an overall performance, counting the number of scenarios that one model generates better results (higher median in $PnLs$) than the other, and also the scenarios that they perform similarly.

Chapter 5

Results and Discussion

This chapter is going to demonstrate the *out-of-sample* testing on the training under the **FNN**, **LSTM** and **GRU** model. There are 16 scenarios for each training mode, and each of them represents a situation under a unique cost rate and a risk aversion level. We are interested in the impact of the cost rate and risk preference on the hedge ratio.

Three types of plots illustrate different aspects of the effect.

- The plot of the predicted hedge ratios versus the spot price at the near-maturity time. We set the near maturity time as $0.9T$ (section 5.1).

  Practically speaking, when we are reaching the end of the trade horizon, traders are more likely to take action on their portfolio, whether hedging or not. So it is necessary to see the behaviour of our hedge strategies at different spot price levels under different levels of cost rates and risk preferences. All the hedge ratios of our simulated paths at this time will be reflected on the plot so that we can have a clear idea of the general condition of our predictions at the time step.

- The plot of the predicted hedge ratios versus along the trade horizon (section 5.2).

  Although this is a plot just for one of the simulated paths. The plot will illustrate how the hedge ratio change as time increases, so we can see the discrepancy between our predicted hedge ratios and the benchmark BS hedge ratios at each time.

- The plot of $PnLs$ for each training mode under different cost rates and risk preferences (section 5.3).

  This is an important plot since it measures how our hedge ratios work. The skewness and position of the distributions reflect our final holdings after trading.

After examining the performance of each model. We will have a statistical test among models to see how results under 3 training models differ from each other. We will conduct the

- **Two-sample Kolmogorov Smirnov test**: fix cost rate and the risk aversion at the same level, test if the PnL calculated under different trading modes produce the same distribution. The null hypothesis for the test is

  $$H_0 : \text{two samples are from the same distribution.}$$

  We performed the test using the python *scipy.stats.kstest* package.

- **Median Test**:

  - One-sample sign test: test if the median of PnL for each scenario is greater than or equal to 0 using the lower-tail test, and the null hypothesis is

    $$H_0 : median(\{PnL_i\}_{i=1,\dots,N}) \geq 0.$$

We performed the test using the python *scipy.stats.binom_test* library.

– Two-sample Wilcoxon singed-rank test: fix cost rate and the risk aversion at the same level, test if the median under one training model is smaller than the others, i.e.,

$$H_0 : median(\{PnL_{\text{model}_1,i}\}_{i=1,\dots,N}) \geq median(\{PnL_{\text{model}_2,i}\}_{i=1,\dots,N}).$$

We performed the test using the python *scipy.stats.wilcoxon* library.

Ideally, we would like the median of PnL under each training model for different scenarios greater than 0, so under the same level of cost rate and risk preference, we prefer the model with a higher median value.

5.1 Hedge Ratio versus Spot Price

In this section, we explain the plot of the hedge strategies along the spot price at $t = 0.9T$ under 3 training modes. The column of the subplots represents the increase in λ, while the row represents the increase in the cost rate. The blue dashed line is the Black-Scholes delta hedging strategy, so it is free from the change of cost rate and lambda and is risk-neutral. The red solid line is the results after plugging in the sequence of $(t, S_t)_t$ such that we set $S_t \in [0, 2], t = 0.9T$. The red dotted scatter plot is the results from our validation set, i.e., they are the results from simulated discrete-time paths at $t = 0.9T$, so the input is random.

5.1.1 General Behaviour

The general behaviour is consistent among different training modes with cost rate and risk preference changing. The illustration in this section will use the results under the **FNN** (figure 5.1) and **GRU** (figure 5.2) training for simplicity. The results for the **LSTM** (figure B.2) can be found in in the appendix B.

Impact of the Cost Rate

We are examing the results from the figures 5.1 and 5.2. Let us remain lambda, or the risk aversion level, at a fixed rate, i.e., look at rows of the plots. We can see that the hedge ratio curve becomes flattened and flattened. Especially around the out-of-money region, i.e., the region where the spot price is less than the strike price $K = 1$, we see a clear trend that it has a wider and wider upward basis as the cost rate increases, except for the top right one which the predicted hedge ratio is completely 0. This is caused by the small risk-averse coefficient, which we will discuss later.

As introduced in the section 3.2.2, the hedge ratio represents our holding in the risky underlying assets, so $\phi = 1$ means that we hedge all our portfolios, while $\phi = 0$ means that we can keep our current positions. Intuitively, when the cost rate becomes large enough, traders hedge less around the in-the-money region and over hedge around the out-of-money region because this will decrease the sensitivity of hedge ratios towards the price change. The trend can be shown vividly when fixing $\lambda = 1$, i.e., the second row of the plot in 5.1. When the cost rate is large enough, hedge ratios fix at a constant rate to avoid changes in holdings. In other words, no matter whether the price jump to a higher place or lower place, our change in the portfolio will be less to avoid a higher cost in the transaction. This can be demonstrated by the cost function (3.3.2) as well, i.e.,

$$|\pi_{t-1,i} - \pi_{t-2,i}|S_{t-1,i}$$

will become smaller.

Impact of the Risk Preference

We are examing the results from the figures 5.1 and 5.2. The λ is the risk aversion coefficient. The large value in the λ represents that we are more risk-averse, indicating that we would like to perfectly hedge our portfolios and get rid of the risk of under-hedging, while the small λ indicates that we care more about the return; this behaviour is shown explicitly in the third column of the graph 5.1 when the cost rate is 0.5%. We can see that our predicted values become more and more attached to the Black-Scholes hedging strategies as λ increases since in the case of Black-Scholes, making sure all our portfolios are hedged, avoiding the risk arising from the contingent claim.

The sub-figure at the top right in the the figure 5.1 shows that in a very less risk-averse trading environment, the strategies will be more sensitive to the change in the cost rate, and the change in the cost rate will dominate the change in strategies while the cost rate has limited impact on the strategies if λ is large, i.e., more risk-averse and conservative, and vice-versa, i.e., strategies also sensitive to λ in a high-cost trading environment.

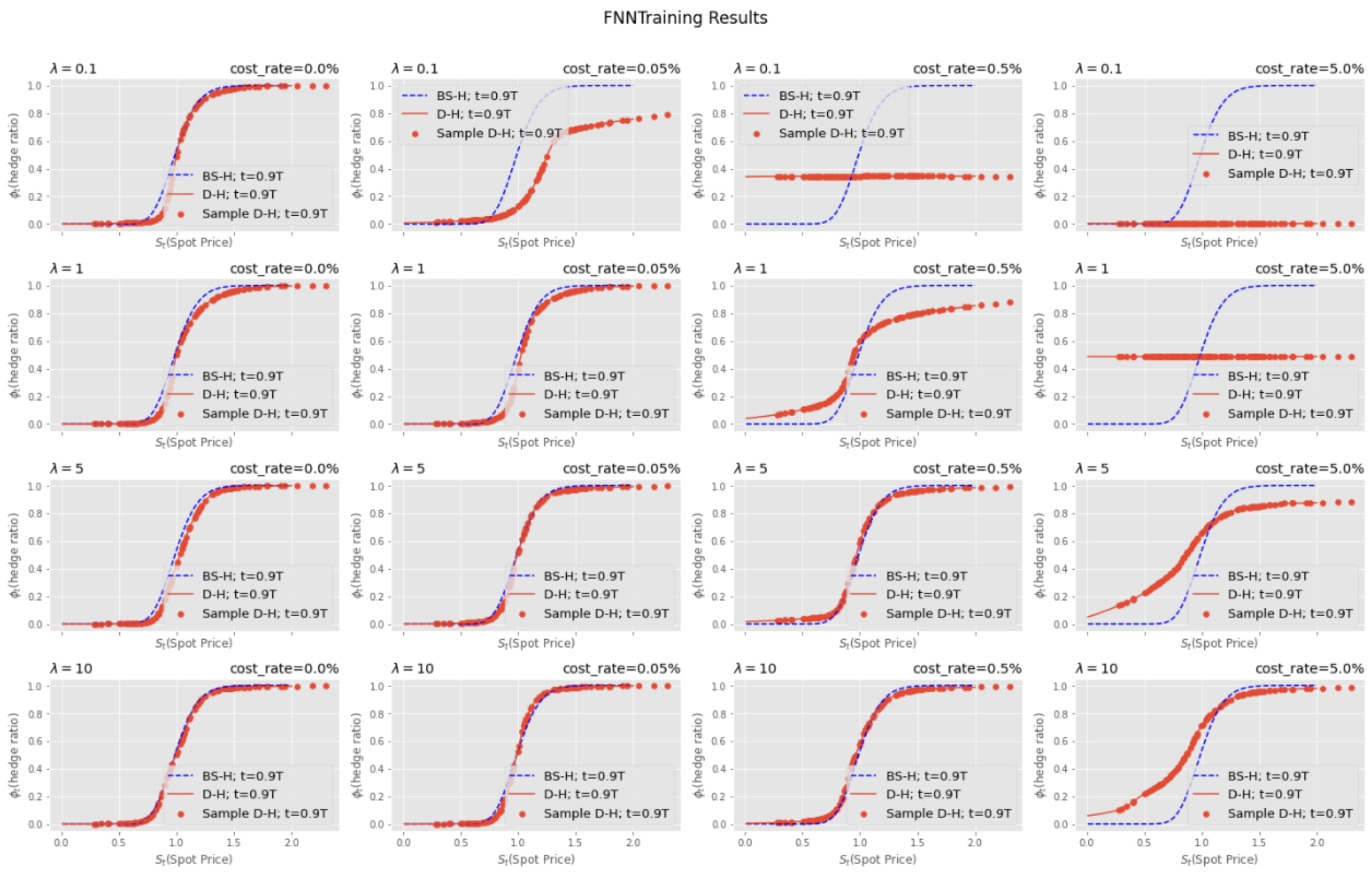


Figure 5.1: The plof of hedge ratio versus spot price with FNN model at different cost rates and risk preferences

5.1.2 Behaviour Specific to FNN

Under the FNN model, as shown in the figure 5.1, we found that the dotted red line overlaps with the solid red line. This is expected since we have trained separate FNN models at each time stamp, i.e., for each pair of input data $\boldsymbol{x}_t = (t, S_t)$, where $t = 0, \ldots, T-1$, we have a unique FNN model $f(\boldsymbol{x}; \boldsymbol{\theta}_t)$ to process it and generate a prediction $\hat{\phi}_t$. It is a one-to-one relationship between the input and output. Whenever a pair of data (t, S_t) comes, the network $f(\boldsymbol{x}; \boldsymbol{\theta}_t)$ will always generate the same output.

5.1.3 Behaviour Specific to LSTM and GRU

We can see the differences when we look into the recurrent network in the figure B.2 and 5.2. The red scatter plot are not stick to a curve. The mechanism for the GRU and LSTM is different from FNN, as mentioned in the section 2.3. They can capture the correlation among the historical data, which means that the data inputs are different from the FNN either. While FNN treats data at each time stamp separately, the two recurrent neural networks take in the whole simulated path as input, so at each time stamp, the network will absorb the useful information from previous time stamps. This is the reason that the out-of-sample scatter plots are radiated since each path learns from its own history and most often never produces the same output for the same spot price value.

Since predictions rarely stayed in lines, except for some extreme cases (small λ's and large cost rates), we also observed the impact on the spread of the predictions when the cost rates and risk preferences are changing. Fixing the cost rates, the large λ not only shifts the general trends of the predictions towards the Black-Scholes delta hedging strategies but also tightens the predictions. On the contrary, large values in the cost rates widen the spread of the predictions, as shown in the second row of the figure 5.2.

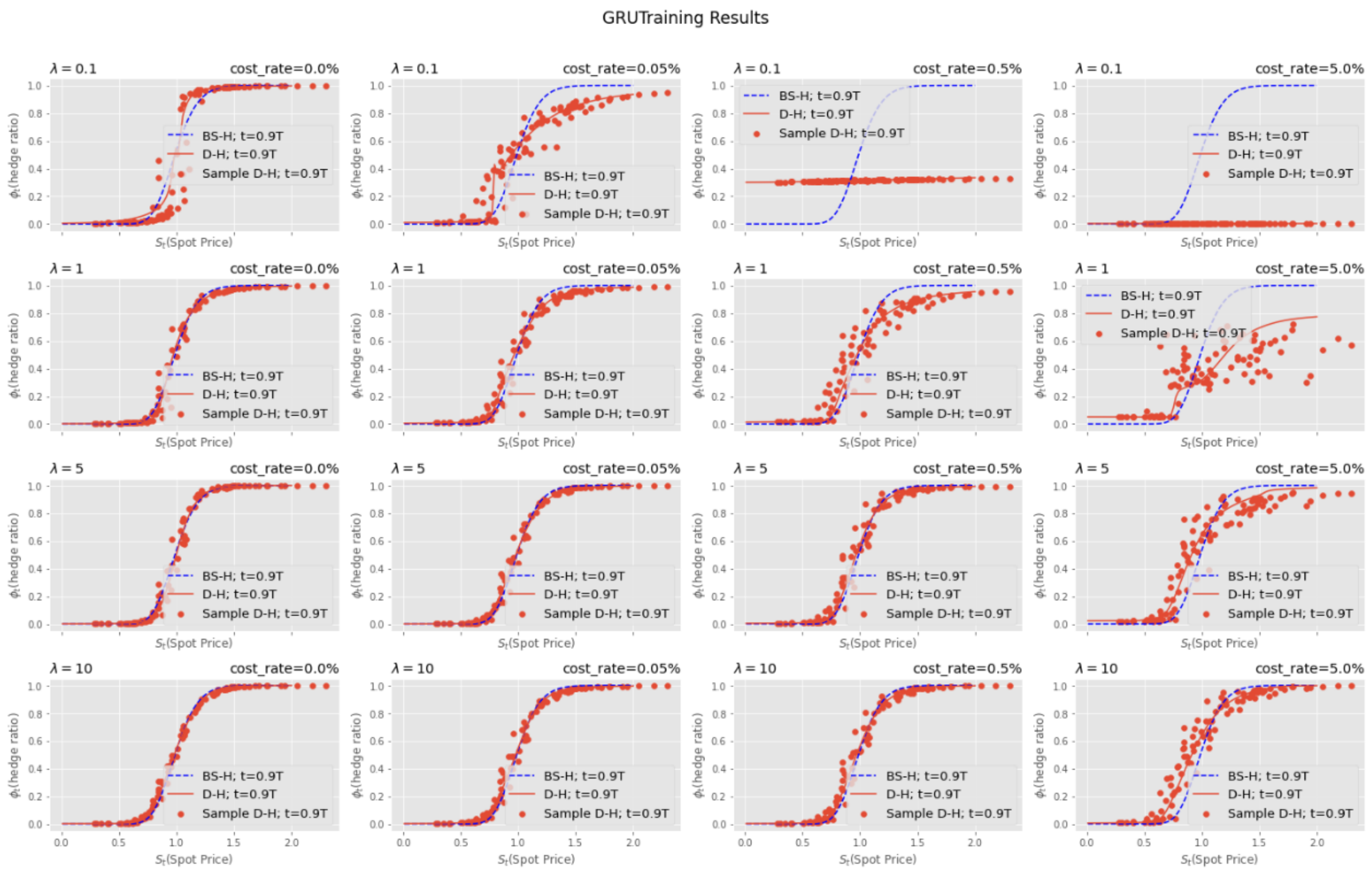


Figure 5.2: The plof of hedge ratio versus spot price with GRU model at different cost rates and risk preferences

5.2 Hedge Ratio versus time

In the previous section 5.1, the results are shown in a freezing time. We can consider it as a snapshot at a specific time spot for all simulated paths. Now, we are going to look at the result from a different angle. This set of plots reflects how cost rates and risk preferences affect the strategies' sensitivity to the price change over time by how a simulated path from the validation set is changing overtime.

5.2.1 General Behaviour

Impact of the Cost Rate

Now let us fix the risk aversion coefficient, i.e., look at only a sub plot in the FNN training results (figure 5.3(a)) and GRU training results (figure 5.3(b))[1]. The red solid line is our benchmark, the Black-Scholes hedging strategies over the time. This time we look at the plot where $\lambda = 0.1$ first since it will be the one that most sensitive to change of cost rate as mentioned in section 5.1.1.

When $\lambda = 0.1$ and cost rate $= 0$, we can see that the volatility of our strategies ϕ_t are following the Black-Scholes strategies more closely, i.e., whenever the Black-Scholes hedging ratio is change, the deep hedging ratio are more likely to follow the changes at the same rate and amount. This is reasonable since the cost is in the tolerance range, and the frequency and magnitude in executing the trades do not cause much extra spending. As cost rate increase to the next level, our deep hedging strategies become smoother and smoother and insensitive to the change in the underlying price.

Moreover, from the plot of ϕ_t v.s. S_t (figure 5.1) as shown in the section 5.1, we know that around the strike price it has the steepest slope, meaning the strategies is most sensitive to the change in price and becomes unstable around strike price. As a result, if the underlying spot price is around K, then at a reasonable amount (refer to values) of cost rate level, the curve will still be volatile due to the relatively high sensitivity to the change in price.

For example, in our plot 5.3 at $\lambda = 0.1$ and cost rate $= 0.5\%$, we can see the purple dashed line still have a relatively large decrease around $t \in (0.5T, 0.6T)$, following the Black-Scholes strategies. It is caused by this sensitivity towards price change around K and results in a sharp increase in the hedge ratio. We can see from the black-schoels strategies that the price of the underlying is staying out-of-the-money at the end of the horizons since the low hedge ratio corresponds to a low spot price, so our deep hedging strategy, represented by the purple line in this scenario, is less responsive to the change in price due to the rise in transaction cost and becomes smooth again.

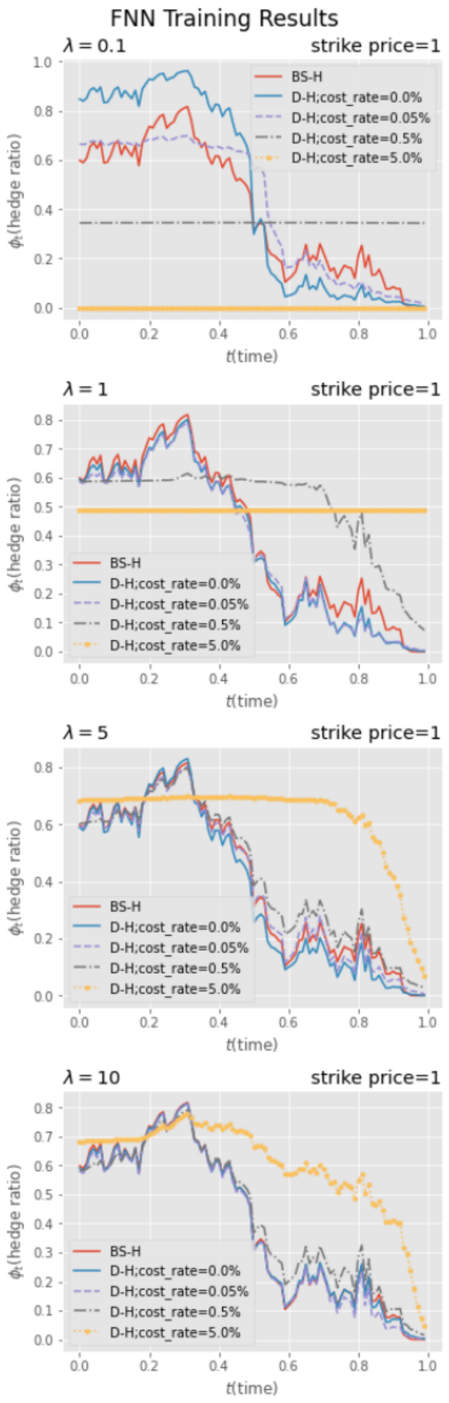
Impact of the Risk Preferences

Let us keep looking at the figure 5.3. Now let us move on to the effect of the λ on the deep hedging strategies. In section 5.1, we know that the deep hedging strategies are sensitive to the change in λ in a high-cost environment, so we are going to look at the yellow dotted line in the four subplots first, which is under the scenario of cost rate $= 5\%$. As λ increases, it becomes closer and closer to the Black-Scholes ones at each time stamp.

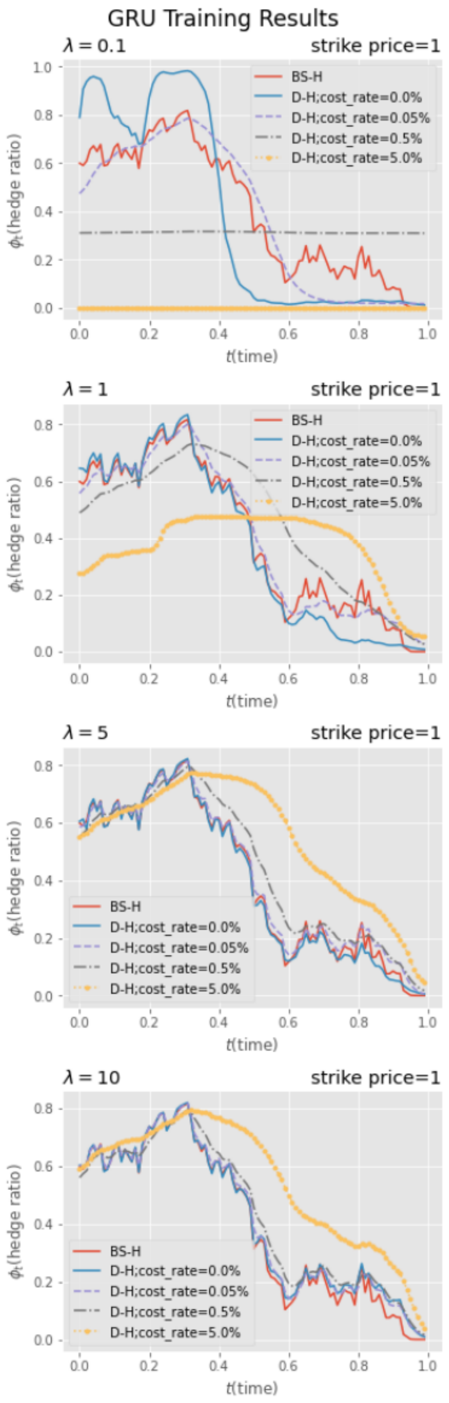
When $\lambda = 0.1$, cost rate $= 5\%$, the plot is matching the situation in the previous plot of hedge ratio versus the spot prices (figure 5.1) under the same cost rates and risk preference levels. At $t = 0.9T$, which is a near-maturity time, we do not hedge at all spot price level and it is intuitively predicable that the previous strategies are all 0 since normally the at the end of the trade horizon, strategies are tend to be more responsive and dynamic.

The blue curves in the figure 5.3 demonstrate typical impacts from the change in λ's. When cost rate $= 0$, intuitively, if we are more willing to expose to the risk and less risk-averse, i.e., $\lambda = 0.1$, the deep hedging strategies would tell trader to hedge more when in-the-money and hedge less when out-of-money as shown in the figure 5.3, so we could have the chance to maximize our PnL instead of hedging perfectly to compensate the potential risk.

[1]The plot for LSTM results is in the figure B.3.



(a) FNN results



(b) GRU results

Figure 5.3: The plot of hedge ratio along the time horizon at different cost rates and risk preferences

5.2.2 Behaviour Specific to LSTM and GRU

As shown in the figure 5.3, if we fix the cost rates at higher levels, as λ increases, the changes are initially occurring from the tails of the flattened curves, which around the maturity. At the end of the trade, traders must decide whether execute the trade or not, so the changing in the strategies are mostly occurs when the trade horizon is approaching to the end. Intuitively, **FNN** provides the strategies for trader to trade only when it is necessary just before maturity to avoid the cost.

5.2.3 Behaviour Specific to RNN

On the other hands, **GRU** and **LSTM** are able to capture more information. In general, as we can see from the figures 5.3(b) and B.3, they can capture more variation in the price change. This can be reflected while the cost rate is high, which is resulting from its features of learning from the past. The **GRU** and **LSTM** are expected to perform well regarding the path dependent product. However, the product we are predicting is the vanilla call option which is not path-denpendent. We expect that when λ is big and without cost rate, the curve should overlap the Black-Scholes curve. Comparing the red curve (Black-Scholes hedging strategies) and the blue curve ($\lambda = 10$, cost-free) in the last sub figure under all three training models, **FNN** 5.3(a), **LSTM** B.3 and **GRU** 5.3(b), From figure 5.3(b), **FNN** has the prediction that are closest to the Black-Scholes'. However, after incorporate the proportional transaction costs, it becomes path-dependent, but in this case we do not have a benchmark to compare to.

5.3 Histogram of PnL

In order to better compare the results, it is necessary to take a look at the PnL since it measures our final gain after applying the strategies. Let us now see the general behaviour of the $PnLs$ under different cost rates and risk preferences.

5.3.1 General Behaviour

Since the general trends for all training modes are similar, in this section we demonstrate the variations regarding the **FNN** results[2], as shown in figure 5.4.

Similarly, there are 16 scenarios for different cost rates and risk preferences. Sub plots in a row illustrate the increases in the cost rate, while for each column it represents the increases in the risk-averse level. Each of the sub plot have a histogram under the deep hedging strategies, as shown in red, and a histogram under the Black-Scholes hedging method, as shown in blue. There are also two lines locating the mean and median of the $PnLs$ resulting from deep hedging strategies. The two lines aim on better visualizing the movement of the general shifts and the skewness caused by the changes in the cost rates and risk preferences.

Impact of the Cost Rate

Fixing λ, in the figure 5.4, for each row, we observe that as cost rates increases,

1. the spread of the histograms becomes larger and larger;

   The widen in the spread is caused by the less traction to the changes in the prices as we discussed in the previous section (section 5.1 and the section 5.2). We know that under the Black-Scholes assumption, the mean of the PnL is around zero since it is calculated under the risk-neutral measure, so the more discrepancy from the Black-Scholes strategies, the more wider the spread is.

2. the histogram of the deep hedging strategies shows a skewness to the left, which can also be demonstrated by the growing distance between the mean and median;

[2]The LSTM PnL results are shown in figure B.4, and GRU PnL results are shown in figure B.5

Let first see an extreme case, the plot at top right corner in the figure 5.4, i.e., $\lambda = 0.1$ and cost rate $= 5\%$, where our predicted hedge ratio $\phi_t = 0$ for all t. If we do note hedge at all, the distribution of PnL will be an inverse L shape, which has a highest median among all plots but the loss can reach a highest among all other situations, either. From (3.3.4), we can know that

$$\begin{aligned} PnL_Z(V_0, \phi; S, k) &= V_0 + \sum_{i=0}^{t-1} \phi_i \cdot (S_{i+1} - S_i) - c(\phi; S, k) - Z \\ &= V_0 - Z \\ &= V_0 - \max(S_T - K, 0), \end{aligned}$$

so when spot price is out-of-the-money, our maximum gain is the premium we receive from the contingent claim Z. In the meanwhile, the potential loss is unlimited as well, demonstrated in the left tail of the histogram.

For a less extreme case, with a reasonable cost ratio, it follows the same logic. We know that the cost would flatten the hedge rate curve with repect to time and spot price, indicating that we hedge less when it is in-the-money and hedge more when it is out-of-the-money. As long as the hedging strategies are not 0 at all time stamps, it will cover the loss caused by the Z using the cashflow gained from premium depends on the frequency and amount a trader would like to spend on it.

3. mean of the deep hedging also shifts to the right.

   We can see that it is not obvious when the cost rates are controlled in a reasonable amount. The obvious shift happens when the cost rate is extremely high and the shift in the skewness is very likely to be the reason.

Impact of the Risk Preference

The changes in the PnL histrogram caused by λ are consistent to our previous observation. That is, fixing the cost rate, when λ is increasing, our strategies are closer to the Black-Scholes delta hedging strategies, pursuing a 'perfect' hedge, and as a result the PnL will becomes more symmetric and tightened.

Details for the performance of histograms under different networks are in section 5.5.

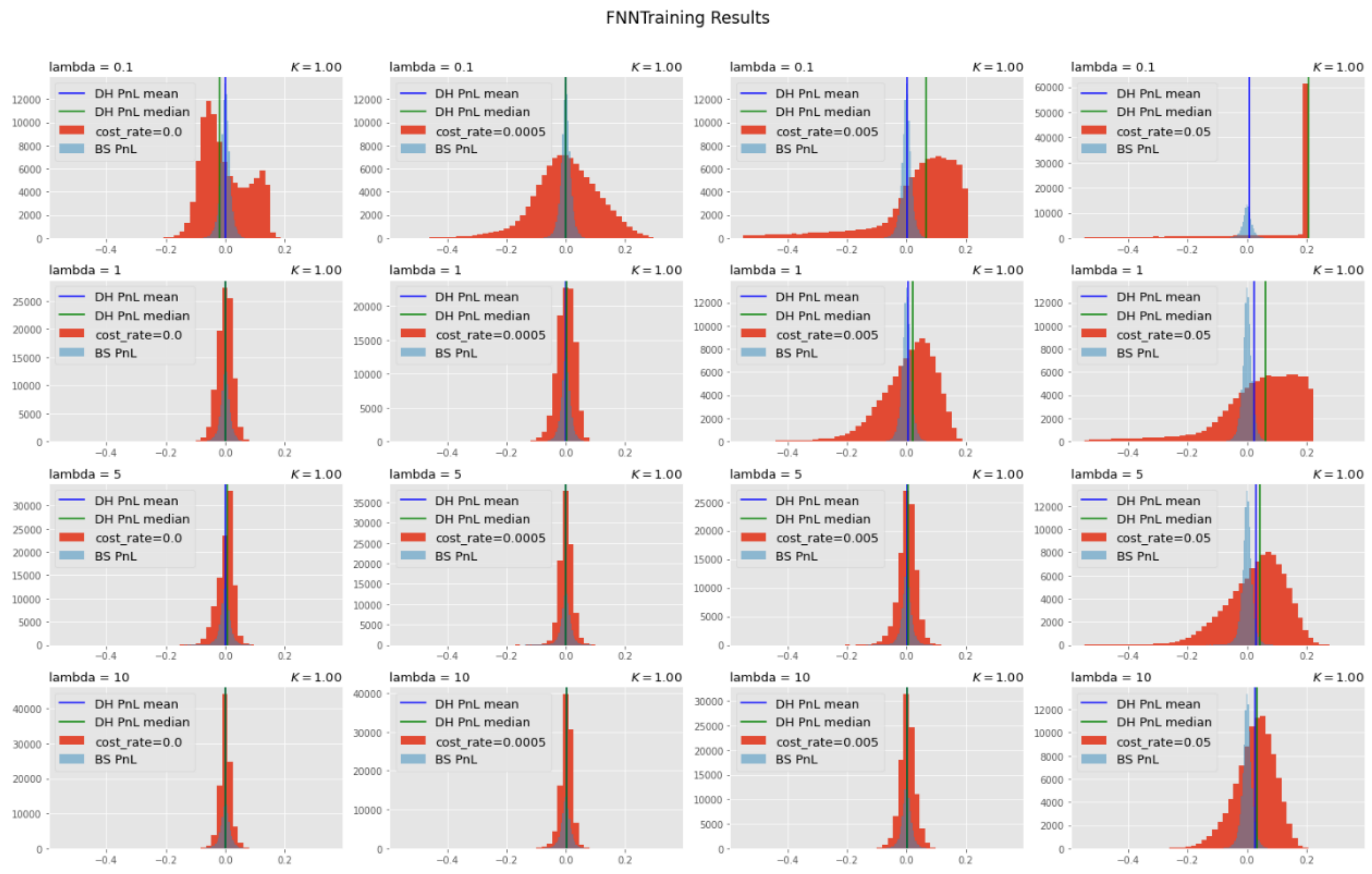


Figure 5.4: PnL under deep hedging strategies with FNN model at different cost rates and risk preferences

5.4 indifference Pricing

The rest of the tables list the indifference prices obtained from the deep learning training. Each table presents the results under 16 scenarios with a specific training mode, where table 5.1 shows the results under **FNN** training, table 5.2 is the results under **LSTM** training and table 5.3 contains the results under **GRU** training. The cells that are bolded in yellow represent that the model has obtained the lowest indifference price compared to the other 2 models.

We introduced in section 3.3.4 that the indifference price is the minimal amount compensating the risk from the contingent claim. We prefer a lower value in the result. As we can see from the table, when the cost rate and λ are the same, the GRU training results reach the lowest amount in the indifference prices for 9 scenarios (as bolded in the table), while the other 3 scenarios have the lowest indifference prices obtained by the **FNN** model. **GRU** obtains better results than the other two models, especially in a high-cost environment.

Moreover, we have done the sanity check on the indifference price as well. [25] pointed out that the indifference price with small transaction cost follows the relation $p_k - p_0 k^{2/3}$, where k is the cost rate. In other words, $\log(p_k - p_0)$ should have a linear relationship with k. We calculated and plotted them to check if our indifference prices were reasonable. The plots for indifference prices calculated under the FNN, GRU and LSTM model are shown in figure 5.5, figure 5.6 and figure B.1, respectively.

When $\lambda = 1, 5$ and 10, the sample points are in a line, and the slopes are all in the range of $(0.7, 0.81)$, which satisfies the linearity condition. For $\lambda = 0.1$, points are not behave as expected. The issues may be caused by the training. Intuitively, the exponential utility function has small gradients when λ is small so the update in gradient descent (section 2.4) is small, causing insufficient training.

Table 5.1: results of FNN indifference prices under different cost rates and risk preferences

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%	BS Price
$\lambda = 0.1$	**0.1975**	0.1992	0.2039	0.2060	0.1974
$\lambda = 1$	0.1977	0.1998	0.2123	0.2694	0.1974
$\lambda = 5$	0.1992	0.2005	0.2170	0.3240	0.1974
$\lambda = 10$	**0.1990**	**0.2014**	0.2188	0.3457	0.1974

Table 5.2: results of LSTM indifference prices under different cost rates and risk preferences

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%	BS Price
$\lambda = 0.1$	0.1978	0.1989	0.2039	0.2060	0.1974
$\lambda = 1$	0.1978	0.1993	0.2080	0.2663	0.1974
$\lambda = 5$	0.1988	0.2008	0.2118	0.2961	0.1974
$\lambda = 10$	0.1996	0.2021	0.2317	0.3143	0.1974

Table 5.3: results of GRU indifference prices under different cost rates and risk preferences

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%	BS Price
$\lambda = 0.1$	0.1977	**0.1987**	0.2039	0.2060	0.1974
$\lambda = 1$	0.1977	**0.1992**	0.2080	**0.2643**	0.1974
$\lambda = 5$	**0.1985**	**0.2002**	**0.2117**	**0.2951**	0.1974
$\lambda = 10$	0.1993	0.2015	**0.2142**	**0.3086**	0.1974

5.4.1 General Behaviour

Impact of the Cost Rate

Fixing the risk-averse level λ, looking at each row, we observe that there is a positive relationship between the indifference prices and the cost rates. Increasing the transaction cost pushes up the price.

Impact of the Risk Preference

If we are looking at the tables regarding the columns, we can find that the more risk-averse we are, the higher the indifference prices are, especially when market frictions are incorporated.

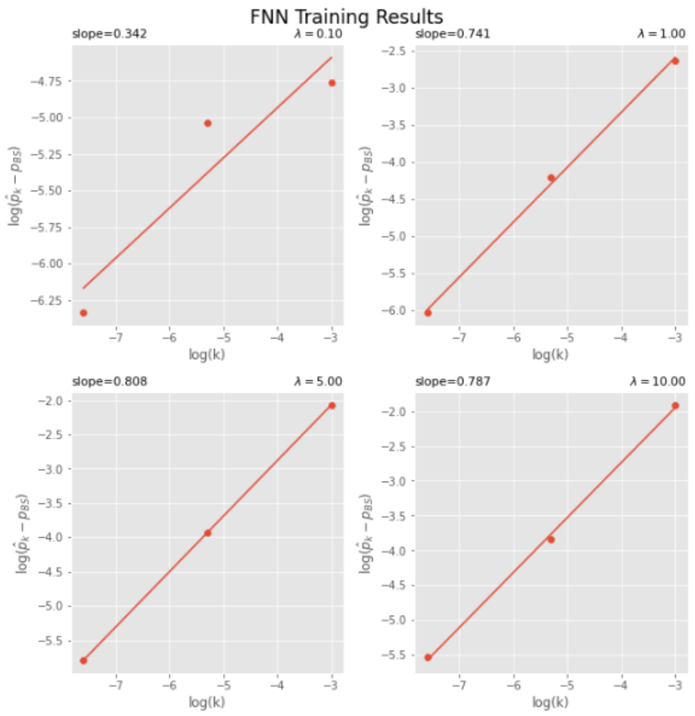


Figure 5.5: sanity check on the indifference price for FNN model under different risk aversion level

5.5 Statistical Test on Results

In order to better compare the results with different cost rates and risk preferences under different models, we would like to examine the overall performance on the PnL. Since the goal for hedging is to ensure that traders are able to pay back the contingent liability, the PnL is expected to be at least 0, i.e., we want the

$$PnL = (\text{initial wealth} + \text{hedge portfolios}) - (\text{contingent claim} + \text{cost})$$

to be no less than 0. This motivates the statistical tests on the PnL distribution. Clearly, from the results, the shape of $PnLs$ under different scenarios is skewed so that the medians can reflect the overall performance better than the means. As a result, the non-parametric tests on distribution and median are adapted in our dissertation.

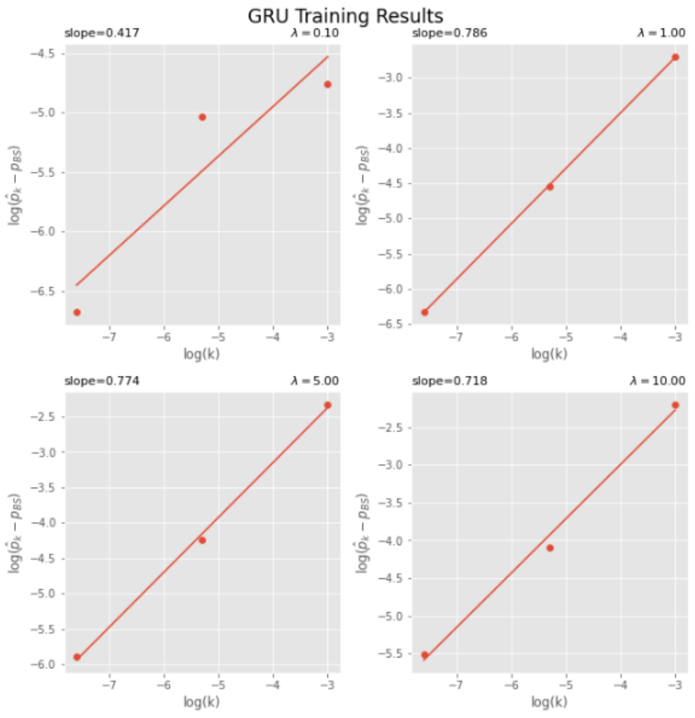


Figure 5.6: sanity check on the indifference price for GRU model under different risk aversion level

5.5.1 Kolmogorov Smirnov Test

Results

In the table 5.4, it presents the tests among the three training models, **FNN**, **LSTM** and **GRU**, where the first column is the p-values for comparing the distribution of $PnLs$ between **FNN** training and **LSTM** training, second column is for **FNN** and **GRU** and third column is for **GRU** and **LSTM**.

We observed that at a 99.9% confidence level, all results suggest that we will reject the null hypothesis that the two empirical distributions are the same. In other words, less than 0.1% of the times that the two empirical distributions are the same. From a statistical point of view, all distributions are significantly different and it requires us to do further analysis on how they are different from each other.

5.5.2 One-sample Signe Test

FNN Results

As shown in the bolded cell in table 5.5, almost all scenarios have large p-values approaching 1, indicating that we do not have enough evidence to reject the null hypothesis $H_0 : median(PnL_{FNN} \geq 0)$ except for the condition where $\lambda = 0.1$ and cost rate $= 0$. That is, less than 0.1% of the times the median of $PnLs$ under the **FNN** training will larger than or equal to 0, and we reject H_0 at a 99.9% confidence level. This can also be reflected by the histogram of PnL under FNN training, as shown in the top left corner of the figure 5.4.

Table 5.4: p-values for the ks-test H_0: $\{PnL_{\text{model}_1,i}\}_{i=1,...,N}$ and $\{PnL_{\text{model}_2,i}\}_{i=1,...,N}$ have the same distribution, where N is the number of simulated underlying paths

scenarios	FNN v.s. LSTM	FNN v.s. GRU	GRU v.s. LSTM
$\lambda = 0.1, cr = 0$	0.0	0.0	0.0
$\lambda = 0.1, cr = 0.05\%$	0.0	0.0	0.0
$\lambda = 0.1, cr = 0.5\%$	0.0	0.0	0.0
$\lambda = 0.1, cr = 5\%$	0.0	0.0	0.0
$\lambda = 1, cr = 0$	0.0	0.0	0.0
$\lambda = 1, cr = 0.05\%$	0.0	0.0	0.0
$\lambda = 1, cr = 0.5\%$	0.0	0.0	0.0
$\lambda = 1, cr = 5\%$	0.0	0.0	0.0
$\lambda = 5, cr = 0$	0.0	0.0	0.0
$\lambda = 5, cr = 0.05\%$	0.0	0.0	0.0
$\lambda = 5, cr = 0.5\%$	0.0	0.0	0.0
$\lambda = 5, cr = 5\%$	0.0	0.0	0.0
$\lambda = 10, cr = 0$	0.0	0.0	0.0
$\lambda = 10, cr = 0.05\%$	0.0	0.0	0.0
$\lambda = 10, cr = 0.5\%$	0.0	0.0	0.0
$\lambda = 10, cr = 5\%$	0.0	0.0	0.0

Table 5.5: p-values for the one-sample sign test $H_0 : median(PnL_{\mathbf{FNN}}) \geq 0$;

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%
$\lambda = 0.1$	**0.0**	0.997434	1.0	1.0
$\lambda = 1$	1.0	1.0	1.0	1.0
$\lambda = 5$	1.0	1.0	1.0	1.0
$\lambda = 10$	1.0	1.0	1.0	1.0

Table 5.6: p-values for one-sample sign test $H_0 : median(PnL_{\mathbf{LSTM}}) \geq 0$;

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%
$\lambda = 0.1$	1.0	1.0	1.0	1.0
$\lambda = 1$	1.0	1.0	1.0	1.0
$\lambda = 5$	1.0	1.0	1.0	1.0
$\lambda = 10$	1.0	**0.0**	1.0	1.0

LSTM Results

Similarly, as bolded in the table 5.6, when we are training using the **LSTM** model, we have a significantly small p-value when $\lambda = 10$ and cost rate $= 0.05\%$. For that scenario, we reject the null hypothesis $H_0 : median(PnL_{LSTM}) \geq 0$ also at a 99.9%-confidence level.

GRU Results

Table 5.7: p-values for the one-sample sign test $H_0 : median(PnL_{\mathbf{GRU}}) \geq 0$;

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%
$\lambda = 0.1$	1.0	1.0	1.0	1.0
$\lambda = 1$	1.0	1.0	1.0	1.0
$\lambda = 5$	1.0	0.999994	1.0	1.0
$\lambda = 10$	1.0	1.0	1.0	1.0

As shown in the table 5.7, for results coming from the **GRU** training, p-values under all cost rates and risk preferences are nearly 1, so we do not have enough evidences to reject the null hypothesis at a $H_0 : median(PnL_{\mathbf{GRU}}) \geq 0$ at a 99.9% significant level for all scenarios. That is, under all circumstances, at least 99.9% of the times the median for the $PnLs$ will be non-negative under the **GRU** model.

5.5.3 Two-sample Wilcoxon Signed-rank Test

In the previous section, we know that the deep hedging strategies trained by different models produce non-negative $PnLs$ at a 99.9% level of confidence for almost every scenario in our experiments. The next step is to identify which model structure provides a relatively higher median.

LSTM v.s. FNN

Table 5.8: p-values for the two-sample signed-rank test H_α : median(LSTM) < median(FNN)

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%
$\lambda = 0.1$	1.0	1.0	1.0	**0.0**
$\lambda = 1$	1.0	0.488999	**0.0**	**0.000837**
$\lambda = 5$	**0.0**	1.0	1.0	**0.0**
$\lambda = 10$	1.0	**0.0**	1.0	0.108538

For results in table 5.8, there are 6 scenarios (bolded in the cells) having $pval < \alpha = 0.1\%$, indicating that we are rejecting the null hypothesis and the medians for **LSTM** training are smaller than it for the **FNN** training at a 99.9% confidence level. Especially when the cost rates are high (cost rate = 5%), the medians under the **FNN** training demonstrates better results.

Table 5.9: p-values for the two-sample signed-rank test H_α : median(FNN) < median(LSTM)

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%
$\lambda = 0.1$	**0.0**	**0.0**	**0.0**	1.0
$\lambda = 1$	**0.0**	0.511001	1.0	0.999163
$\lambda = 5$	1.0	**0.0**	**0.0**	1.0
$\lambda = 10$	**0.0**	1.0	**0.0**	0.891462

On the contrary, from the table 5.9 we observe that **LSTM** performs better than **FNN** in a relatively lower-cost environments (cost rate = 0, 0.05% and 0.5%). Under such environments, over 50% of the scenarios have medians in the $PnLs$ under the **LSTM** model higher than it under the **FNN** model, at a 99.9% confidence level. Moreover, notice that for a low risk-averse setting ($\lambda = 0.1$), **LSTM** also performs better.

Table 5.10: summary on LSTM and FNN median tests at 99.9% level of confidence

condition	count on scenarios	percentage out of all scenarios
median(LSTM) < median(FNN)	6	37.5%
median(LSTM) > median(FNN)	8	50%
median(LSTM) = median(FNN)	2	12.5%
total scenarios	16	100%

Overall speaking, medians are higher under the **LSTM** model than the **FNN** model, while **FNN** can do better in a high-cost environment, as shown in the table 5.10. Visualization on the PnL distribution for the two models under each scenarios are presented in figure 5.7.

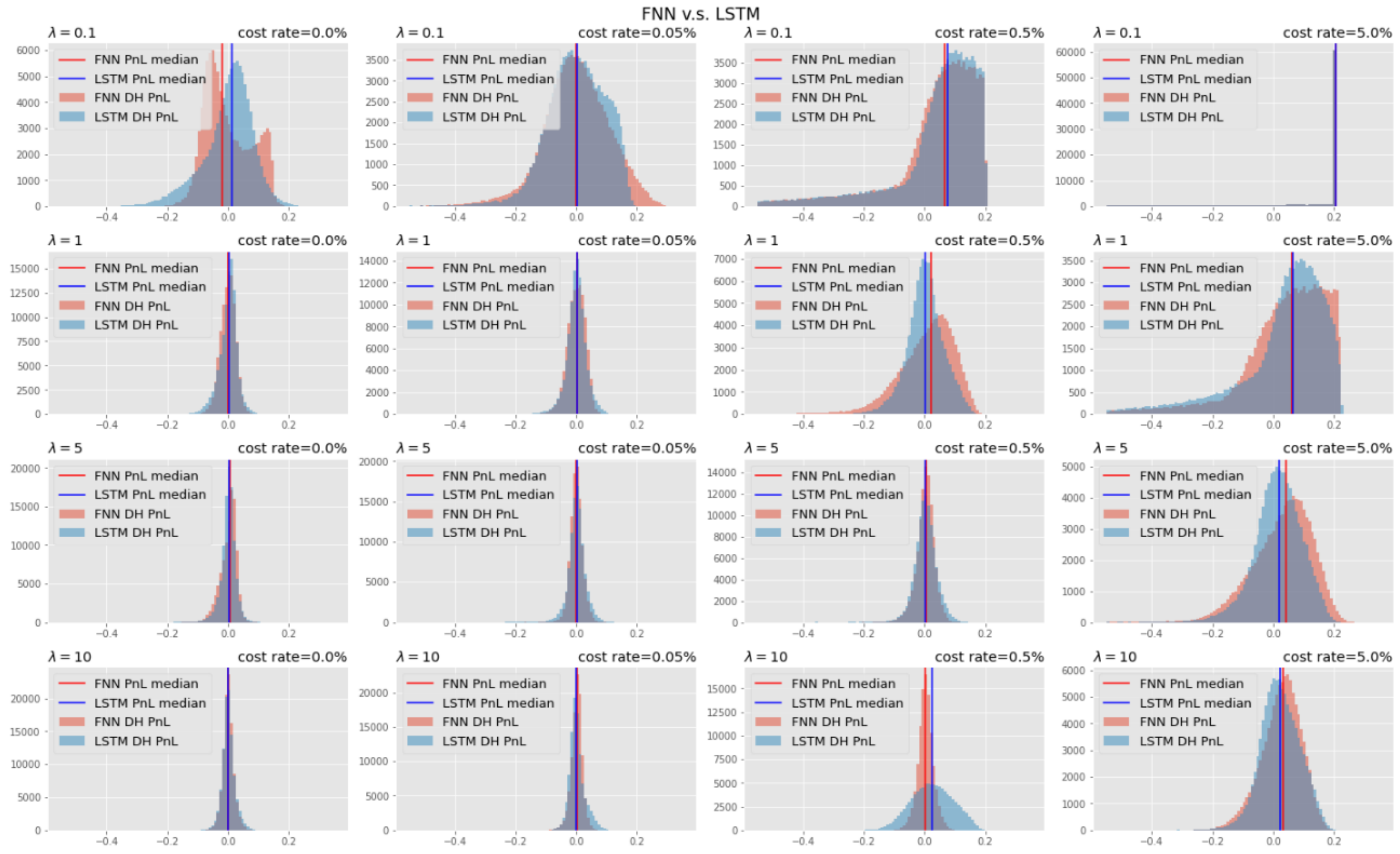


Figure 5.7: PnL of deep hedging strategies with FNN and LSTM at different cost rates and risk preferences

LSTM v.s. GRU

Table 5.11: p-values for the two-sample signed-rank test H_α : median(LSTM) < median(GRU)

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%
$\lambda = 0.1$	**0.0**	**0.000010**	0.630753	**0.0**
$\lambda = 1$	1.0	0.142254	**0.0**	**0.0**
$\lambda = 5$	1.0	1.0	1.0	1.0
$\lambda = 10$	1.0	**0.0**	1.0	1.0

From the table 5.11, we observe that under 6 scenarios that the pvals< 0.1%, meaning that the medians obtained from the **LSTM** model are lower than it from the **GRU** model, at a confidence level of 99.9%. The scenarios mainly occurs when the risk-averse level is low, i.e., $\lambda = 0.1$.

Table 5.12: p-values for the two-sample signed-rank test H_α : median(GRU) < median(LSTM)

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%
$\lambda = 0.1$	1.0	0.999990	0.369247	1.0
$\lambda = 1$	**0.0**	0.857746	1.0	1.0
$\lambda = 5$	**0.0**	**0.0**	**0.0**	**0.0**
$\lambda = 10$	**0.0**	1.0	**0.0**	**0.0**

Moving on to the table 5.12, we find that for 8 scenarios that the p values are smaller than the significance level $\alpha = 0.1\%$ when $\lambda = 1$, 5 and 10, which means that **LSTM** generates higher medians in $PnLs$ when the risk-averse levels are relatively higher, at a 99.9% of confidence level. That is, for $lambda = 5$ and 10, only 1 out 8 scenario for which **LSTM** is *not* better than **GRU**.

Table 5.13: summary on LSTM and GRU median test at 99.9% level of confidence

condition	count on scenarios	percentage out of all scenarios
median(LSTM) < median(GRU)	6	37.5%
median(LSTM) > median(GRU)	8	50%
median(LSTM) = median(GRU)	2	12.5%
total scenarios	16	100%

Table 5.13 demonstrates the overall conditions regarding the results under different cost rates and risk-averse level. Surprisingly the statistics are the same as in table 5.10, but **LSTM** are *better* than **GRU** in a different perspective. While the differences in performance for **LSTM** and **FNN** mostly appear among different cost rates, the **LSTM** model differentiate from the **GRU** model when the risk-averse levels change. **LSTM** performs better in a higher risk-averse environment. The comparison of the PnL distribution between **LSTM** and **GRU** can be found in figure 5.8.

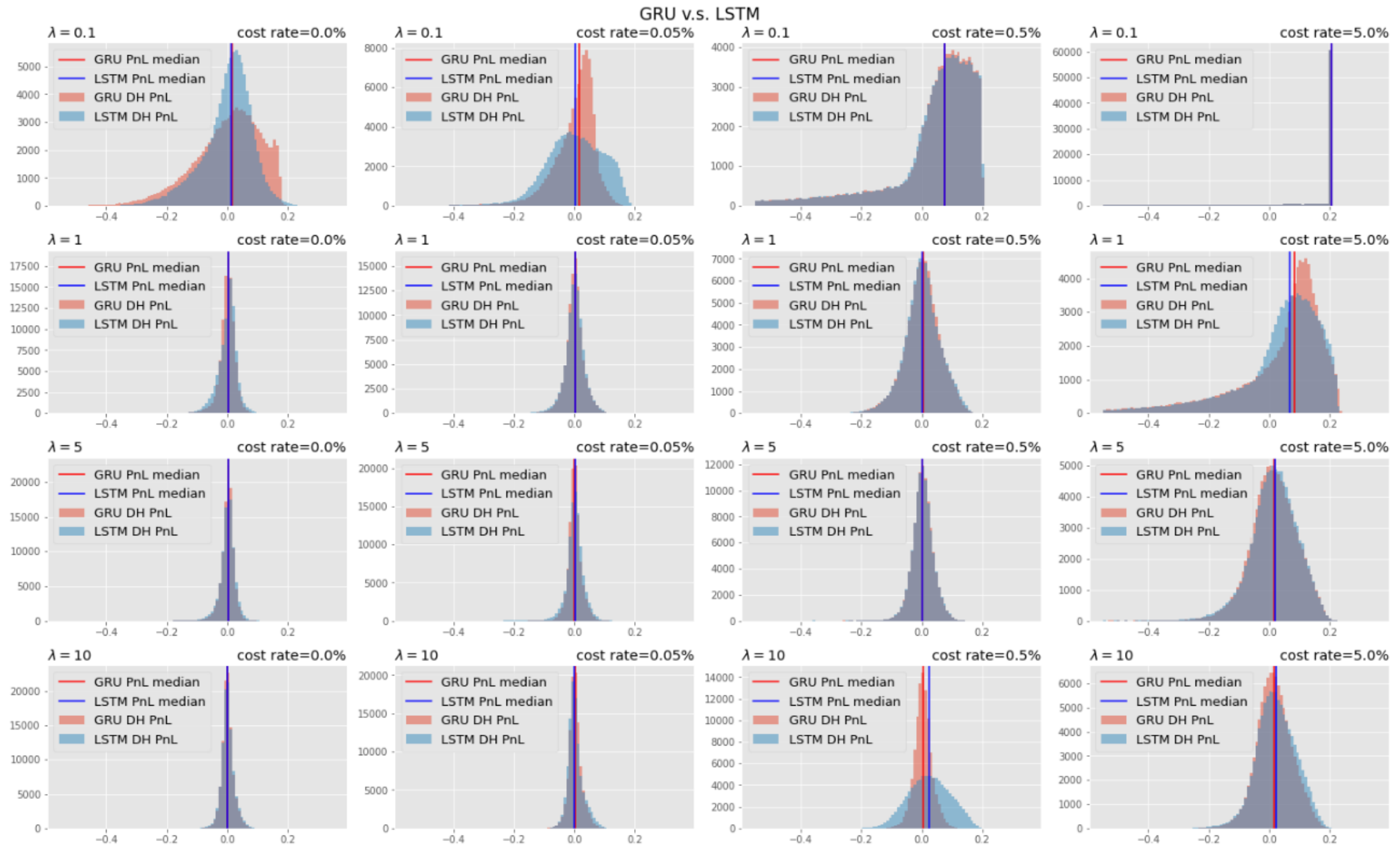


Figure 5.8: PnL of deep hedging strategies with GRU and LSTM at different cost rates and risk preferences

FNN v.s GRU

From the previous two sections, we know that **LSTM** model outperforms the **GRU** and **FNN** from different perspectives when we are comparing the medians under different scenarios. Now we are examining the performances between **GRU** and **FNN** in this section.

Table 5.14: p-values for the two-sample signed-rank test H_α : median(FNN) < median(GRU)

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%
$\lambda = 0.1$	**0.0**	**0.0**	**0.0**	1.0
$\lambda = 1$	**0.0**	0.722204	1.0	1.0
$\lambda = 5$	1.0	0.999521	**0.0**	1.0
$\lambda = 10$	0.999998	1.0	**0.0**	1.0

The table 5.14 demonstrates that under 6 scenarios (bolded in the cells) we reject the null hypothesis that $PnLs$ under **GRU** model are having higher median than it under **FNN** model. At a cost rate of 0.5%, **GRU** performs better for most of the risk-averse levels.

Table 5.15: p-values for the two-sample signed-rank test H_α : median(GRU) < median(FNN)

	cost rate = 0	cost rate = 0.05%	cost rate = 0.5%	cost rate = 5%
$\lambda = 0.1$	1.0	1.0	1.0	**0.0**
$\lambda = 1$	0.999811	0.277796	**0.0**	**0.0**
$\lambda = 5$	**0.0**	**0.000479**	**0.0**	1.0
$\lambda = 10$	**0.000002**	**0.0**	1.0	**0.0**

In table 5.15, under 9 conditions for medians under the **FNN** training are greater than it under the **GRU** training. Moreover, it is consistent that the **FNN** model produces higher medians under the circumstances when cost rates are high (cost rate = 5%). Moreover, the **FNN** also performs better under thigherger risk-averse levels ($\lambda = 5$ and 10).

Table 5.16: summary on GRU and LSTM median test at 99.9% level of confidence

condition	count on scenarios	percentage out of all scenarios
median(FNN) < median(GRU)	6	37.5%
median(FNN) > median(GRU)	9	56.25%
median(FNN) = median(GRU)	1	6.25%
total scenarios	16	100%

The statistics in the table 5.16 show that in general **FNN** are better than **GRU**, especially under a high risk-averse and high cost rates level environment. In other words, **FNN** performs better than **GRU** if we are adapting conservative strategies. Visualizations can be found in figure 5.9.

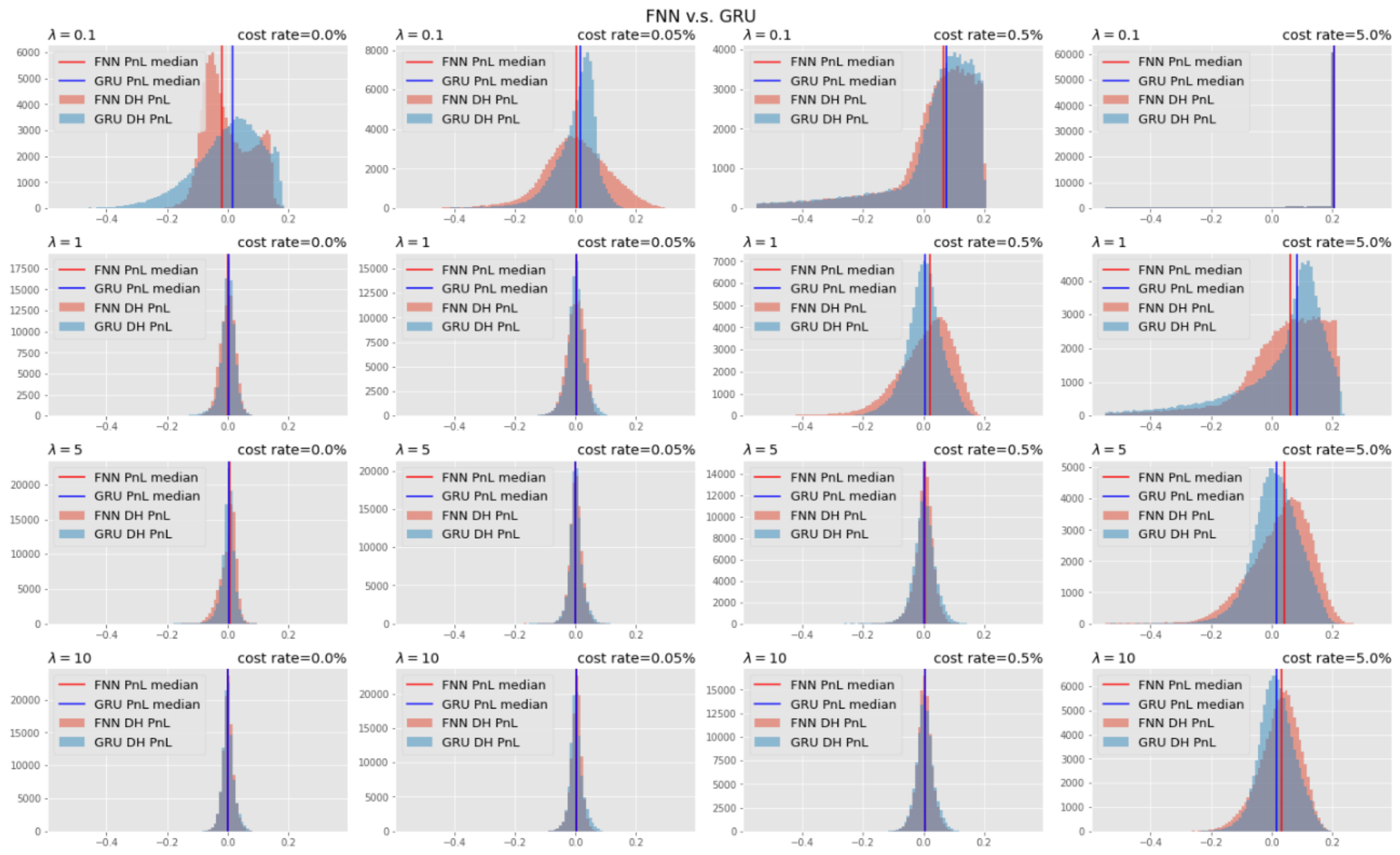


Figure 5.9: PnL of deep hedging strategies with FNN and GRU at different cost rates and risk preferences

Chapter 6

Conclusion

In this dissertation, we used the deep hedging method to solve the pricing and hedging problems of the derivatives under certain market frictions and risk preferences. The hedging and pricing problems can convert to deep learning optimization problems by setting up appropriate input and output values and objective functions. In particular, the input will be the market states where relevant market information such as underlying prices are included. The outputs are the predicted hedging strategies using certain deep learning models, and it is the so-called deep hedging strategies. Finally, we set the objective functions as the risk measurement regarding the PnL resulting from the deep hedging strategies. In the context of deep learning, the deep hedging strategies are a minimizer to the objective functions, while speaking in the language of finance, deep hedging strategies are the strategies that minimize the risks of hedging at a certain level of preference in the risk exposure. Moreover, if we choose a convex measure applied to the PnL, we can get a fair price for the derivatives even under an incomplete market. The pricing problem can also fit into the deep learning framework, essentially the difference between two optimized risk measures regarding the PnL.

We are interested in how the changes in the costs of transactions and risk preferences impact our deep hedging strategies and indifference prices, and how they vary under different deep learning models. We applied the proportional transaction cost which charges at a constant rate on our change in portfolio when executing the trade at each time stamp. Moreover, we incorporated the risk measure by inputting the PnL into the exponential utility functions, so the objective of the deep learning model is to minimize the empirical mean of the exponential utility functions regarding the PnL.

The deep hedging strategies and the indfference prices obtained under different deep learning models, **FNN**, **GRU** and **LSTM**, behave consistently. The increases in the transaction cost make the predicted hedging strategies less sensitive to the changes in spot price over time, making it a smoother version of the black-shcoles hedging strategies. In other words, the amount of changes in the deep hedging strategies will be less than the amount of changes in the Black-Scholes strategies to avoid the costs in transactions. The higher the cost rates are, the less change in the hedge ratios over time. If the cost rate is high enough, traders are expected to trade at a constant hedge ratio, and in an extreme case, such as when the risk-averse level is low and the cost rate is high, the predicted hedge ratios are 0 to avoid the cost. The discrepancy between the deep and Black-Scholes hedging strategies causes a larger variance in our $PnLs$ under a high-cost environment. On the other hand, if we fix the cost rate and increase the risk-averse level, our hedging strategies are closer to the Black-Scholes strategies since under the Black-Scholes strategies we are expected to perfectly hedge the portfolios.

The three deep learning models used in the dissertations generate reasonable results and we conduct the statistical tests on the median of $PnLs$ to investigate the differences in the overall performances under different models. Overall, under 16 different scenarios (with different cost rates and risk preferences), **LSTM** model generates higher medians than the others in most cases, while **FNN** performs the best under a high-cost environment. **LSTM** stands out in producing higher medians in $PnLs$ by its well-performance under the high risk-averse levels. Compared to the

FNN and **LSTM**, the **GRU** model generates relatively lower medians in most of the scenarios. However, speaking of the indifference pricing, **GRU** model obtained the lowest indifference prices for 9 out of 16 scenarios, **FNN** model obtained 3 lowest indifference prices and **LSTM** were not obtaining any lowest prices for any conditions.

We took the European call option as the contingent claim for hedging, which is a path-independent derivative. For future developments, path-dependent options such as barrier option, and Asian option can also be also taken into considerations, where the **LSTM** and **GRU** models are expected to handle them well. It is also more than welcome to include the real market data than the Black-Scholes market simulators. Regarding the model training, we used the expectation of the exponential utility functions as the risk measure to the $PnLs$, while other risk measures such as the *Conditionanl Value At Risk*, $\rho(x) = \inf_{\omega \in \mathbb{R}}\{\lambda \mathbb{E}[(\omega - x)^+]\}$ [4, section 2.5], *mean-variance metrix*, $\rho(x) = \lambda(\mathbb{E}(x^2) - \mathbb{E}(x)^2) - \mathbb{E}$ [4, section 2.5], and sharp ratio are also popular and worth for testing.

Appendix A

Examples

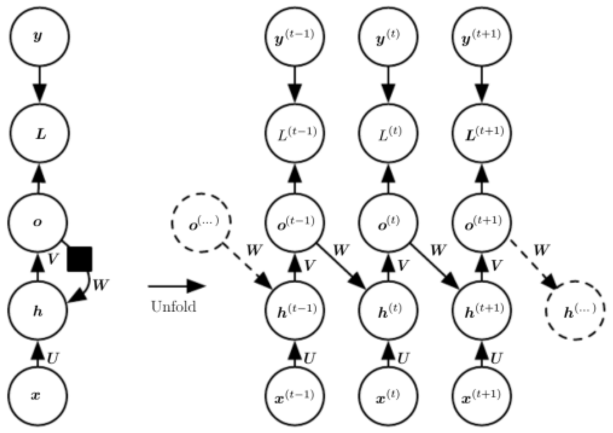


Figure A.1: RNN structure with feedback coming from the output $\boldsymbol{o}^{(t)}$ [1, p375]

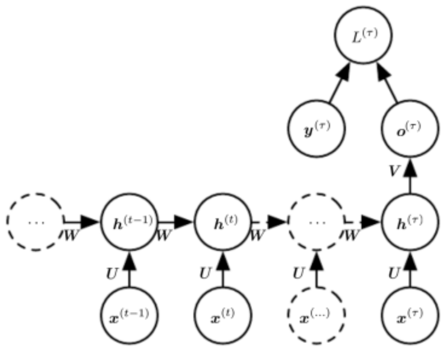


Figure A.2: RNN structure with only 1 output at the last time stamp [1, p376]

Appendix B

Other Related Results

B.1 LSTM Training Result

B.1.1 indifference Price

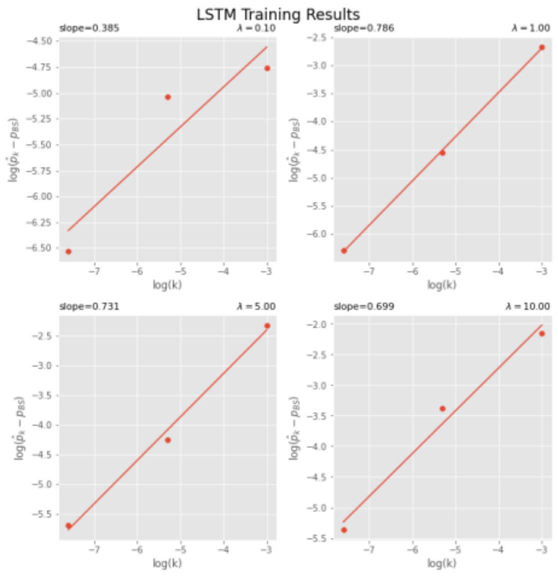


Figure B.1: sanity check on the indifference price for LTSM model under different risk aversion level

B.1.2 hedge ratio versus spot price

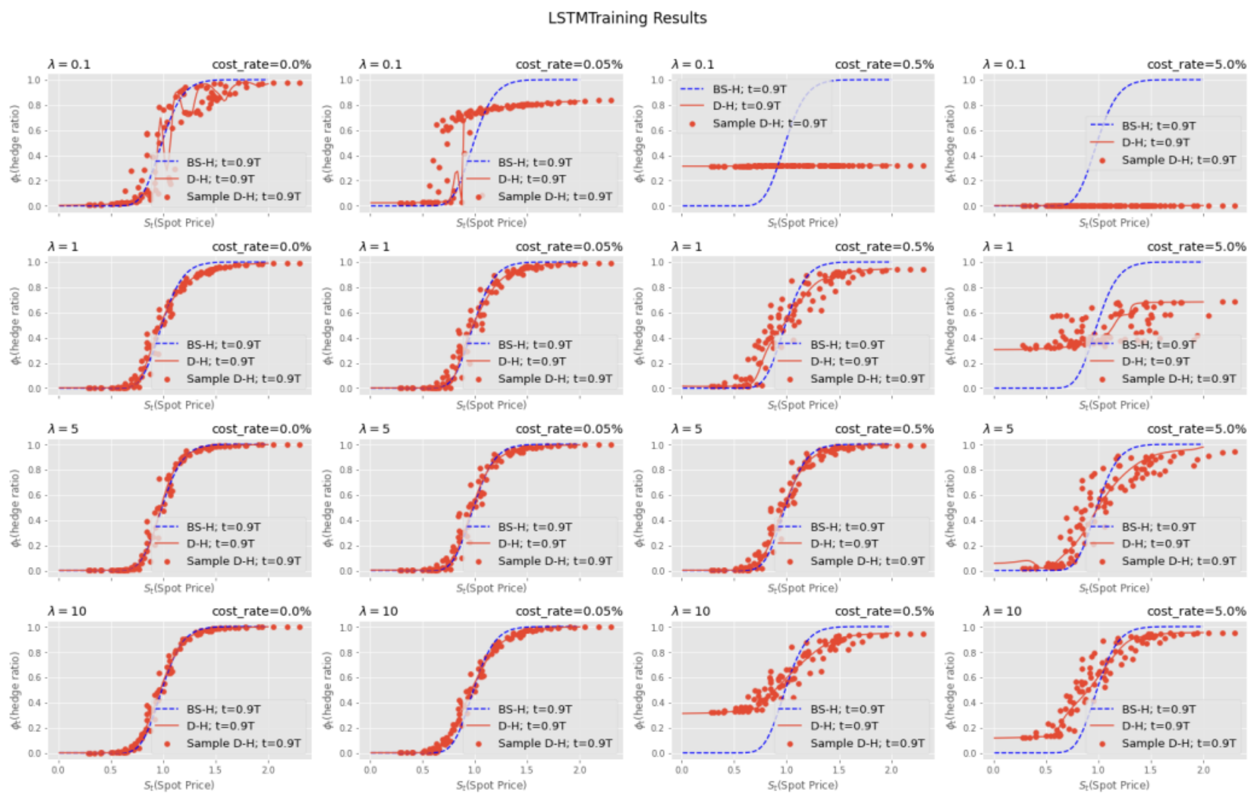


Figure B.2: The plof of hedge ratio versus spot price with LSTM model at different cost rates and risk preferences

B.1.3 hedge ratio versus time

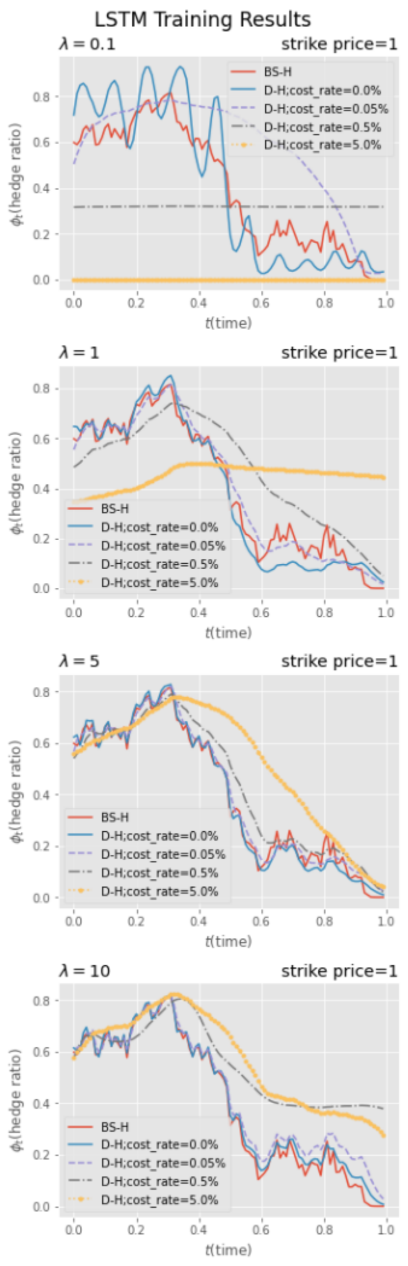


Figure B.3: The plof of hedge ratio along the time horizon with LSTM model at different cost rates and risk preferences

B.1.4 hedge ratio pnl

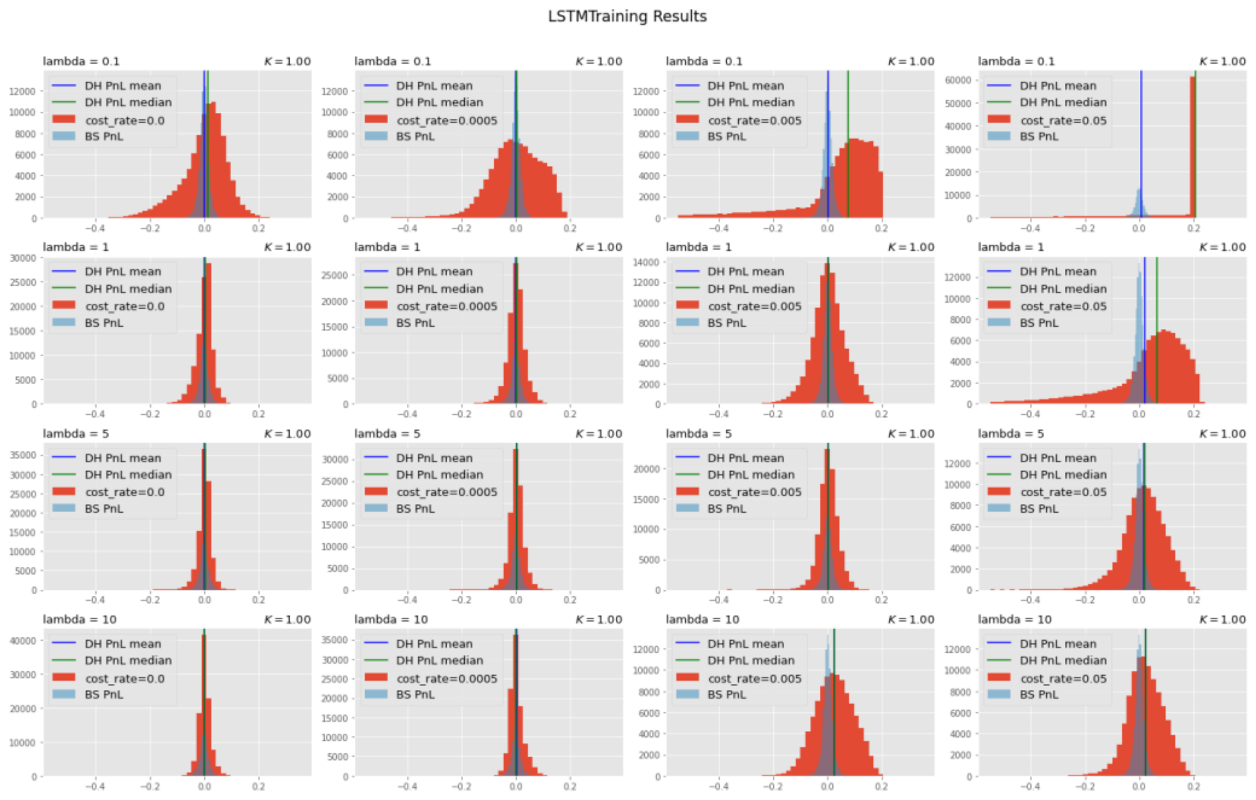


Figure B.4: PnL under deep hedging strategies with LSTM model at different cost rates and risk preferences

B.2 GRU hedge ratio pnl

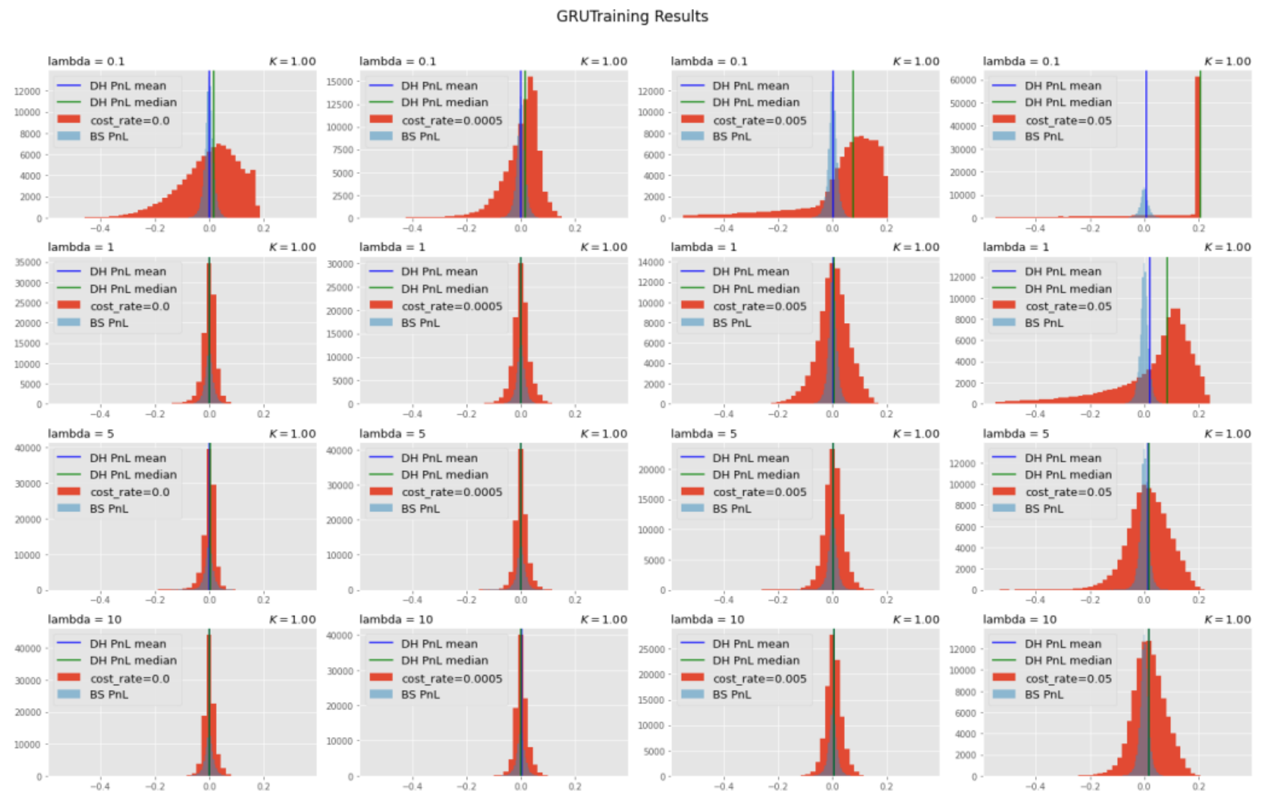


Figure B.5: PnL under deep hedging strategies with GRU model at different cost rates and risk preferences

Appendix C

GRU and LSTM Training Algorithm

Algorithm 3: training procedures for GRU and LSTM

```
   Input  : Initial model parameters: $\boldsymbol{\theta}_0$
   Input  : # epochs: $n_{epoch}$
   Input  : # batches: $n_{batch}$
   Input  : batch size: $b$
   Input  : fast learning rate: $\epsilon_{fast}$
   Input  : fast learning rate: $\epsilon_{slow}$
   Input  : adam_learning_algo: $adam\_learning(\boldsymbol{\theta}, \epsilon)$
   Input  : # slower rate iterations: $n_{iterate}$
   /* train model using adam algorithm with a faster learning rate */
1  for $i$ in $\{1, 2, \ldots, n_e\}$: do
2      initialize $\boldsymbol{\theta}_0^i = \boldsymbol{\theta}_0$ for the first time of training, otherwise, $\boldsymbol{\theta}_0^i = \boldsymbol{\theta}_{n_{batch}}^{i-1}$
3      for $j$ in $\{1, 2, \ldots, n_b\}$ do
4          $\boldsymbol{\theta}_{j+1}^i = adam\_learning(\boldsymbol{\theta}_j^i, \epsilon_{fast})$ // referred to algorithm 2
5      end
6  end
   /* continue training model using adam with a slower learning rate, and iterate the
      training at the slower rate for $n_{iterate}$ times. */
7  for $k$ in $\{1, 2, \ldots, n_{iterate}\}$ do
8      initialize $\boldsymbol{\theta}_{0,k}^0 = \boldsymbol{\theta}_{n_{batch}}^{n_epoch}$ for the first time of training, otherwise, $\boldsymbol{\theta}_{0,k}^0 = \boldsymbol{\theta}_{n_{batch},k-1}^{n_{epoch}}$.
9      for $i$ in $\{1, 2, \ldots, n_e\}$: do
10         initialize $\boldsymbol{\theta}_{0,k}^i = \boldsymbol{\theta}_{n_{batch}}^{n_epoch}$ for the first time of training, otherwise, $\boldsymbol{\theta}_{0,k}^i = \boldsymbol{\theta}_{n_{batch},k}^{i-1}$
            from the last batch
11         for $j$ in $\{1, 2, \ldots, n_b\}$ do
12             $\boldsymbol{\theta}_{j+1,k}^i = adam\_learning(\boldsymbol{\theta}_{j,k}^i, \epsilon_{slow})$ // referred to algorithm 2
13         end
14     end
15 end
16 return $\boldsymbol{\theta}_{n_{batch},n_{iterate}}^{n_{epoch}}$
```

Bibliography

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

[2] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 1973.

[3] Peter Bank, H Mete Soner, and Moritz Voß. Hedging with temporary price impact. *Mathematics and financial economics*, 11(2):215–239, 2017.

[4] Hans Buehler, Lukas Gonon, Josef Teichmann, Ben Wood, Baranidharan Mohan, and Jonathan Kochems. Deep hedging: hedging derivatives under generic market frictions using reinforcement learning. *Swiss Finance Institute Research Paper*, (19-80), 2019.

[5] M. S. Pakkanen, A. Muguruza Gonzalez, and B. Horvath. Data-centric methods. In A. Capponi and C.-A. Lehalle, editors, *Machine Learning and Data Sciences for Financial Markets*. Cambridge University Press, to appear, 2022.

[6] Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quantitative Finance*, 19(8):1271–1291, 2019.

[7] Mikko Pakkanen. *Deep Learning*. MATH97231 Imperial College, October 2021.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[9] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[11] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[13] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[15] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[18] Filippo Santambrogio. {Euclidean, metric, and Wasserstein} gradient flows: an overview. *Bulletin of Mathematical Sciences*, 7(1):87–154, 2017.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[21] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[22] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[23] Damiano Brigo. *Interest Rate Models with Credit Risk, Collateral, Funding Liquidity Risk and Multiple Curves*. MATH97114 Imperial College, October 2021. `http://wwwf.imperial.ac.uk/~dbrigo/masterICmaths`.

[24] Taylor B. Arnold and John W. Emerson. Nonparametric goodness-of-fit tests for discrete null distributions. *The R Journal*, (2):34–39, December 2011.

[25] Jan Kallsen and Johannes Muhle-Karbe. Option pricing and hedging with small transaction costs. *Mathematical Finance*, 25(4):702–723, 2015.