Deep learning for trading and hedging in financial markets

Zheng Gong

A thesis submitted for the degree of

Doctor of Philosophy

at the

School of Computer Science and Electronic Engineering

University of Essex

August 2023

Abstract

Deep learning has achieved remarkable results in many areas, from image classification, language translation to question answering. Deep neural network models have proved to be good at processing large amounts of data and capturing complex relationships embedded in the data. In this thesis, we use deep learning methods to solve trading and hedging problems in the financial markets. We show that our solutions, which consist of various deep neural network models, could achieve better accuracies and efficiencies than many conventional mathematical-based methods.

We use Technical Analysis Neural Network (TANN) to process high-frequency tick data from the foreign exchange market. Various technical indicators are calculated from the market data and fed into the neural network model. The model generates a classification label, which indicates the future movement direction of the FX rate in the short term. Our solution can surpass many well-known machine learning algorithms on classification accuracies.

Deep Hedging models the relationship between the underlying asset and the prices of option contracts. We upgrade the pipeline by removing the restriction on trading frequency. With different levels of risk tolerances, the modified deep hedging model can propose various hedging solutions. These solutions form the Efficient Hedging Frontier (EHF), where their associated risk levels and returns are directly observable. We also show that com-

bining a Deep Hedging model with a prediction algorithm ultimately increases the hedging performances.

Implied volatility is the critical parameter for evaluating many financial derivatives. We propose a novel PCA Variational Auto-Enocder model to encode three independent features of implied volatility surfaces from the European stock markets. This novel encoding brings various benefits to generating and extrapolating implied volatility surfaces. It also enables the transformation of implied volatility surfaces from a stock index to a single stock, significantly improving the efficiency of derivatives pricing.

Acknowledgements

Words cannot express my gratitude to my supervisors, Dr John O'Hara and Professor Carmine Ventre, for their invaluable support and knowledge throughout my PhD study and research. They guided me in this exciting direction for research and career, and I could not have imagined having better advisors for this journey.

I am grateful to Renzo Tiranti, Wojciech Frys and Yingbo Bai, who provided professional comments and suggestions during the research, particularly for their support in connecting the research work with real-life business impacts.

Also thanks my thesis examiners: Prof. Rahul Savani and Dr. Panagiotis Kanellopoulos, for their professional comments and suggestions, which helped me improve my thesis considerably.

Last but not least, I would like to thank my wife for her endless love, support and company through this long journey. Especially when the COVID restrictions caused inconveniences in our ordinary life, her patience and understanding are truly precious.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter is the overall introduction to this research thesis on deep learning for trading and hedging in financial markets, which consists of several aspects and models. The purpose of this chapter is to briefly answer a couple of fundamental questions of (i) ***why*** we are interested in these research directions, (ii) ***what*** questions we are trying to answer and (iii) ***how*** to utilize different methodologies to obtain expected results. Section 1.1 outlines a broad picture of the evolving dynamic regarding deep learning and the interests raised by the financial academia and industry. Section 1.2 specifically discusses what we expect to discover from the combinations of deep learning and financial theories. Section 1.3 introduces this research project's methodologies and provides a detailed discussion on how to achieve the research objectives. Section 1.4 and Section 1.5 present the thesis structure and published thesis results, respectively.

1.1 Overview

Within the past decade, we have seen a lot of promising methods developed in the area of computer vision and natural language processing. Many have profound impacts on the lives of ordinary people, from facial recognition [1], machine translation [2] to till-less grocery stores. A few new industries have also emerged and are making robust progress, and self-driving vehicles would be a typical example [3]. In addition, during the Covid-19 pandemic, new solutions were developed to assist in diagnosing this disease, which is an important and timely breakthrough in medical science [4]. These new technologies have undoubtedly changed how we process and react to various information. The improvements are primarily reflected in the accuracies and efficiencies of the data processing procedure.

We want to emphasize that Deep Learning (DL) or Deep Neural Networks (DNN) is just a way to process numerical information (i.e. matrix and vector). DL is involved in all the applications mentioned above, but it is not a magic tool that can output anything we request. The developer or user of a DL model must first thoroughly understand the context around a problem, focusing on what information could be embedded in the dataset, and then try to tailor the DL model structure and the model training process to extract information from the data. Although there is no comprehensive theory to explain why neural networks are capable in many scenarios, the research community tends to believe that ***Universal Approximation Theory*** plays a crucial role here. Therefore, we think it is necessary to briefly discuss it in Section 2.3.5. These would contribute to the explainability of the methodologies proposed in this thesis.

Since we have seen many impressive developments reshape the horizon for computer vision and other fields, it is natural to consider its feasibility for deep learning to solve financial

problems. When this research project started in 2019, there were already some efforts of applying DL to financial time series prediction problems [5][6][7], portfolio optimization [8][9], credit risk evaluation [10][11], and many other specific tasks. Some papers discussed the rationale for selecting deep neural network models for solving their problems, and others did not. We suggest my reasons from the perspective of trading and hedging financial assets.

First, trading and hedging financial assets usually end up with decisions or strategies generated from analyzing a large amount of numerical data, which is undoubtedly the strength of neural network models. Given their complexities and flexibilities, neural network models are the best candidates to identify appropriate features from the dataset targeted explicitly for the required loss function. It also can learn the features from one domain of knowledge and apply them to another relevant domain, which could be two different foreign exchange rates or stock prices from different shares in the context of the financial market.

Second, as mentioned in previous paragraphs, neural network models can approximate non-linear relationships. One fundamental aspect of financial modelling is to identify relationships; such relationships could be a temporal relationship between historical prices and future prices (forecasting), prices of the underlying asset and its derivatives (hedging) and volatility of particular stock prices and the volatility of stock index price (benchmarking), etc. These relationships are usually dynamic and complex, and neural network models are more appropriate solutions to capture these relationships than pure mathematical equations. Once the neural networks model these relationships, we could get a more in-depth understanding of the evolvement in the financial market. We could also input extra information into the system and use the discovered relationships to evaluate financial products or generate comprehensive asset management strategies.

Third, from the business point of view. Financial institutions are not like manufactur-

ers, and their products are not physically comparable. The mathematical algorirthms behind these products are the key factors to distinguish them. However, we already have many products, services, hedging and trading strategies sharing similar characteristics in the market due to the limitations of the mathematical methods we have. Deep learning models could provide new solutions to many numerical problems in the financial industry and develop different products and services, which could have substantial business impacts and benefits.

1.2 Research Objectives

There are solid reasons to consider deep neural network models when solving problems related to trading and heading in the financial markets. We have seen many research works published in recent years. However, they are far from practical applications. There are two fundamental limitations in those works. First, many models are developed and tested using a small amount of data, which means the dataset was collected from a relatively short period of time, or a limited amount of financial instruments. These papers report superior model performances for the typical dataset, but we know the financial market is constantly evolving. For example, during March and April 2020, stock prices experienced high levels of turbulence because the COVID-19 pandemic was starting, and market participants were not confident about the future. In this scenario, how the model would respond to the market would be a vital issue for everyone. Second, the majority of the models did not consider a model user's knowledge and expectation for the market. A financial market participant should have an intelligent system that suggests different solutions based on various market prospects, and this is because of the typical characteristics of the financial sector. We need to know what drives the model's output, as well as the financial and business implications of the model.

More importantly, the regulators want us to explain how a model meets its purpose.

The limitations discussed above set out a baseline for the questions to be answered by this research thesis. To be precise and applicable, the research objectives are set as follows:

1. To experiment with the neural network's ability to process large amounts of noisy high-frequency financial time series data, which covers a long period of time and many financial instruments.

2. To combine the neural network based pipeline with other machine learning classifiers to achieve higher hedging profits and lower hedging risks for vanilla stock option contracts.

3. To obtain meaningful and interpretable encoding of volatility surfaces with an auto-encoder neural network model and improve the interpretability of synthetic surface generation process.

4. To discover the relationship between the volatility surface of a single stock and a stock index so that a prediction model could be built to generate volatility surfaces of illiquid stock.

1.3 Research Methodologies

To achieve the research objectives listed in Chapter 1.2, we perform three studies to validate these objectives. For each study, we create a new model or upgrade a state-of-art algorithm to solve a specific financial market trading or hedging problem. In this chapter, we briefly state the considerations behind the design of these studies and how their outcomes answer the research questions. More detailed and concrete discussions will be presented in the

relevant chapter for each study.

We want to know if a solution exists for neural networks to process highly noisy financial time series datasets. Although there are already many literatures about forecasting stock prices and foreign exchange rates, most of them are only considering regular sampled time series (daily, weekly, etc.) [12][13][14]. However, the high-frequency tick data has totally different characteristics, and these characteristics present a lot more challenging for extracting useful information from high-frequency data. We propose a new solution called **Technical Analysis Neural Network (TANN)**, which combines technical indicators with convolutional and recurrent neural layers to extract predictive information from high-frequency FX ticks, and discovered that TANN outperforms several conventional machine learning methods regarding prediction accuracy. This study suggests a new direction for high-frequency FX trading and testifies to the universality of neural network models.

Human knowledge and preferences are essential and critical to most trading and hedging tasks, significantly different from other computer science tasks, such as image classification and machine translation, where human guidances or preferences are usually unnecessary. We need intelligent algorithms to assist an experienced investor in making decisions so that the outcome can reflect regulators' requirements and investors' preferences, which could also contribute to the interpretability of a system and make the method one step closer to real business scenarios. We discover a way to input an investor's risk preference into the popular deep hedging algorithm and generate the **Efficient Hedging Frontier (EHF)**. The EHF provides efficient and straightforward messages to a system user about the trade-off between mean (return) and standard deviation (risk) of expected hedging profit so that they can choose the optimal strategy based on individual circumstances. We also include the output of another machine learning algorithm (i.e. random forest) to the neural network

based deep hedging pipeline, which serves as another source of information and can be replaced by human knowledge if necessary. This way, the EHF could be shifted upwards, indicating an improvement in hedging profit at all risk levels. This study experiments to combine various information at the input level of a neural network based pipeline and tailor the outputs to the specific risk preference and predictions from an investor's point of view. The deep hedging method is upgraded and more applicable to real markets.

For the third study, we want to further manipulate the neural network at the intermedia layers instead of just the input level. Therefore, the model's strength of flexibility could be better utilised to fit a more challenging problem, and we can develop a deeper understanding of the mechanism of neural networks when applied to the area of trading and hedging. Auto-encoder is the best category of the neural network model to illustrate the fundamental principle of universal approximation. The input information of an auto-encoder is embedded into a latent vector and then reconstructed by two separate neural networks called encoder and decoder. These two networks are doing similar jobs of dimensionality transferring but in precisely opposite directions. Existing literature uses variation auto-encoder to model implied volatility surface with satisfactory reconstruction performances[15]. We upgrade this model by adding the covariance measurement to the loss function so that the latent factors are independent of each other, which is inspired by the PCA autoencoder model from the computer vision field [16]. The updated auto-encoder model can generate synthetic volatility surfaces from three latent numbers, which define the overall volatility level, the term structure and the skewness separately. We also take a further step to analyse the relationship between the volatility surfaces of a particular stock and its market benchmark (stock index) on the latent encodings so that we can use a simple regression model to predict the latent encoding of a volatility surface for a stock and then provide the predicted encoding to

the decoder neural network to infer the predicted volatility surface. This study shows that the intermedia output of a neural network could be manipulated outside the neural network pipeline and combined with extra information to create an advantageous system to evaluate and trade financial derivatives in the real market.

1.4 Thesis Structure

The thesis structure is based on previous chapters' research objectives and methodologies.

Chapter 2 outlines many aspects to which this research project relates and presents some critical past pieces of literature. This literature review chapter could explain many fundamental concepts and discuss the state-of-art models in financial time series prediction, financial derivative hedging and stock volatility surface parameterisation so that they can be contrasted and compared with the proposed approaches of the research thesis. Chapter 2.3.5 also briefly discusses the Universal Approximation Theory, which explains the he mechanism of neural networks and what factors to consider when we want to use neural network models to solve a specific problem.

Chapter 3 presents the study for processing high-frequency foreign exchange (FX) rates using the technical indicators and convolutional neural network models, and the Technical Analysis Neural Network (TANN) model is purposed to classify future FX price movements. The high-frequency FX tick data is usually considered highly noisy and difficult to be processed. However, the TANN pipeline has overcome many conventional machine learning methods, including K-nearest neighbour, Random Forest and Support Vector Classifier on classification accuracies. It is also discovered that investors' risk preferences will influence a model's classification performance, as the label balance depends on it.

Chapter 4 is the second study in which an investor's risk tolerance plays a more critical role. The deep hedging [17] algorithm opens a new area in which a neural network model could be used to find the optimal hedging strategy for a financial derivative. One obvious drawback of the default deep hedging model is that the underlining asset is traded on a daily basis, which generates a lot of unnecessary trading costs. We try to include an investor's risk tolerance and limit the trading activities to only significant daily price changes of the underlying asset. In this way, the average trading loss is reduced at the end of the derivative contract, but the uncertainty increases. Therefore, the model could generate the efficient hedging frontier (EHF) to clearly outline the balance between risk and return (loss) when trading with the deep hedging model. It could assist a trader in making more reasonable trading decisions. We also found that a prediction agent (such as a random forest model) could be used first to predict the movement of future underlying prices so that the EHFs could be shifted upwards.

Chapter 5 is the third study that considers a deeper combination of external information or instructions with a neural network model to solve more challenging problems. Variational auto-encoder has already been used to model stock implied volatility surface, which is a key parameter to figure out the prices of stock options and many other financial derivatives [15]. However, the existing implementation lacks interpretations for the latent dimensions, and from the financial regulation's point of view, this is not a reliable method. Our work is inspired by the PCA autoencoder model, which uses covariance measures to constrain the training of auto-encoder neural network model, and make the latent vectors independent of each other [16]. In this way, the general volatility level, the term structure and the volatility skew of a single volatility surface are independently controlled by three different numbers. The model is particularly useful for generating synthetic volatility surfaces in different market scenarios

and supporting the stress testing of trading and hedging strategies. We also discovered the relationships of volatility surfaces between a single stock and a stock index at the latent dimension. We use linear regression models to formulate these relationships so that the implied volatility surface of an illiquid stock can be inferred from its stock prices and the implied volatility surface of a liquid stock. This is a significant contribution to methods of financial derivative pricing and hedging.

The thesis is concluded with Chapter 6. It summarises the three studies and highlights their contributions. It also re-emphasises the research questions and their solutions. Further research directions are also discussed in this chapter.

Algorithms and models developed in the thesis are all implemented in Python 3. The source code for Chapter 3 is available at https://github.com/zgong123/TANN, and the source code for Chapter 4 is available at https://github.com/zgong123/EHF. The source code for Chapter 5 cannot be shared publicly as it is a joint research work with UBS AG.

1.5 Publications

Some of the original work done in this thesis has been published in the following peer-reviewed paper:

1. Zheng Gong, Carmine Ventre, and John O'Hara. Classifying high-frequency FX rate movements with technical indicators and inception model. In Proceedings of the First ACM International Conference on AI in Finance, pp. 1-8. 2020.

2. Zheng Gong, Carmine Ventre, and John O'Hara. The efficient hedging frontier with deep neural networks. In Proceedings of the Second ACM International Conference on AI in Finance, pp. 1-8. 2021.

3. Zheng Gong, Wojciech Frys, Renzo Tiranti, Carmine Ventre, John O'Hara, and Yingbo Bai. A new encoding of implied volatility surfaces for their synthetic generation. Workshop on Synthetic Data for AI in Finance, the 3rd ACM International Conference on AI in Finance, 2022.

4. Zheng Gong, Wojciech Frys, Renzo Tiranti, Carmine Ventre, John O'Hara, and Yingbo Bai. A new encoding of implied volatility surfaces for their synthetic generation. In Proceedings of the Fourth ACM International Conference on AI in Finance, pp. 1-9. 2023. Under review.

Chapter 2

Literature Review

This chapter introduces a couple of financial mathematical theories for trading and, in particular, hedging of financial assets. We want to specifically discuss the limitations and weaknesses of these conventional methods when applied in real-life financial markets. This chapter introduces neural network models, from the fundamental computing unit to the high-level model structure. We explain why a neural network is a good candidate for solving trading and hedging problems in financial markets and essential issues to consider when using neural network models.

2.1 Introduction

The fascination of solving trading and hedging problems is not only the potential to make profits from the market but, more importantly, the technics used to dig into a large amount of numerical data and extract in-depth information. We believe the purpose of information extraction (or so-called 'data-mining') is to develop a general understanding of how the different aspects (e.g. financial derivative and its underlying asset) related to each other in the system

based on the data which describes them and use mathematical equations or programmable computer models to represent these relationships. Once we capture these relationships, we could create new aspects (e.g. new product design, trading strategy generation) or validate existing methodologies (e.g. stress-testing). The ultimate objective is to solve problems, increase efficiency in the financial market, and benefit every community member.

In this chapter, we will discuss three broad areas of research: financial theories, neural network models, and other machine learning methods. This thesis focuses on using various neural network models to interpret relationships in the financial market. Therefore, sufficient explanations of financial concepts and classical financial methods must be provided before introducing neural network models.

From Section 2.2, a reader can understand the nature of the problems and identify the relationships we want to model by the various neural network models. It also introduces many widely appreciated financial modelling methods, such as the Black-Scholes-Merton model and stochastic volatility models. Finally, we want to explain the limitations of those conventional methods so that we can emphasize the advantages of our proposed neural network based models.

Section 2.3 presents a thorough discussion of different neural network structures. Our discussions start from a single computing unit (neuron), to the three formats (fully connected, convolution and recurrent) which neurons could be connected to a stand as a computing layer for modelling different types of information, to the high-level arrangement of layers to perform dimensionality reduction (auto-encoder model) and hedging strategy optimization (deep hedging). We introduce the reinforcement learning method in this section, which has prevailed in many practical areas. We also discuss the universal approximation theory and the issues to consider when using neural network models to capture the relationships, espe-

cially for financial time series data.

In Chapter 2.4, we also need to discuss some conventional machine learning methods, such as random forest and linear regression. One reason is that we need to compare neural network-based algorithms with conventional approaches to emphasize the superiority of our proposed method. The second reason is that we want to indicate that financial problems are sometimes very complicated, and neural network-based algorithms have the advantage of working together with conventional methods to achieve the best performances and flexibility.

2.2 Financial Concepts and Theories

To solve a problem or update its existing solution, we first need to understand the problem with its context and concepts. This section considers the most popular theories for modelling financial assets and developing hedging methods. We want to show the basic concepts of the conventional methods, particularly their limitations, so we know what problems are being solved with the proposed methods in this thesis.

2.2.1 High-frequency FX spot rate forecasting

Generally speaking, there are three major financial markets: the foreign exchange (FX) market, the stock market and the bond market. The latter two are usually domiciliary in a specific country or region where the market makers are stock exchanges. All the investors submit their orders to a single exchange. The exchange matches the orders from the buy and sell sides and executes the transactions. Local financial regulators and the exchanges have absolute control over market operations. Therefore, the trading rules and regulations could vary from one location to another. For example, it is only allowed to have a maximum 10%

daily increase or decrease for a stock's close price in the Chinese stock market. But such regulation does not exist in the UK [18].

The FX market is different. It has the exchange-based derivative market, and quote-based spot and forwards markets. The quote-based market is a decentrelised market There is no single market maker. At the top of the market is the interbank market, in which each bank provides prices that they commit to buy or sell currencies from their peers in the market. It, therefore, has relatively weaker and unified regulations around the globe. Government agents, such as central banks and financial regulators, could only influence the FX spot rate by selling or buying currencies in the market just as other market participants, so their controls are much weaker in this market. The FX spot market has specific features such as around-the-clock trading, massive traded volume, and diversity of participants compared to the other two markets mentioned above. According to the Triennial Central Bank Survey, the daily average turnover of the over-the-counter (OTC) foreign exchange market is $7.5 trillion per day in April 2022, and about $2.1 trillion is trading in FX spot [19]. The FX market is one of the most influential markets for global economic growth and stability, and it has stimulated the interests of many investors, policymakers, and academics [20].

On the other hand, the execution mechanism is also significantly different between the exchange-based stock market and the quote-based FX spot market.

Most stock exchanges worldwide facilitate trades using an order-drive method, where all the orders of buyers and sellers are displayed and sorted on the two sides of the Limit Order Book (LOB) [21]. On the LOB, there are several levels of orders on each side. For the bid side, level 1 is the highest price a buyer commits to buy, and similarly, the lowest selling price is on the first level of the ask side. A sample of LOB is illustrated in Figure 2.1. A stock exchange usually has an advanced system to automatically match and execute orders

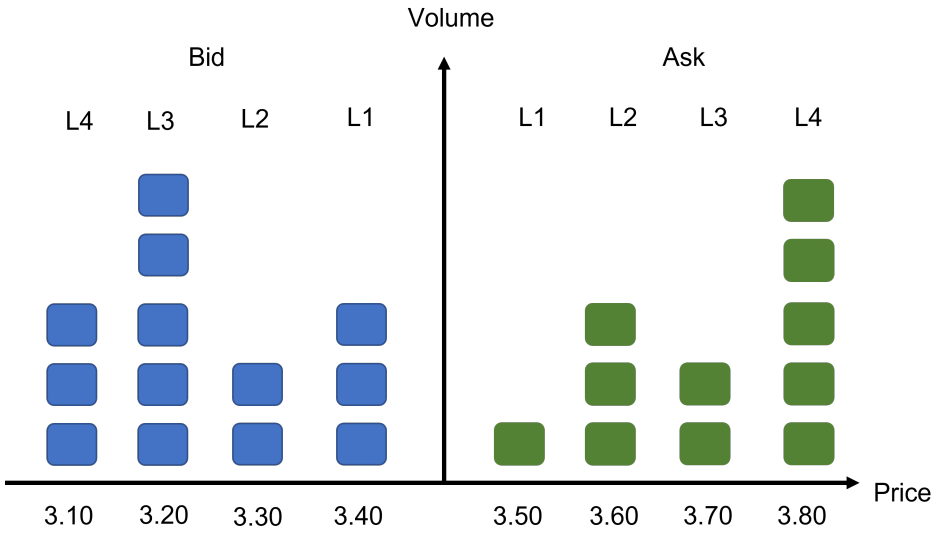


Figure 2.1: An illustration of Limit Order Book

from level 1 to a higher level. If an order is not completed, it stays on the LOB until matched or cancelled. This market microstructure of LOB has the potential to provide short-term information for future prices movement directions because the thickness of orders on both sides of LOB indicates the market anticipation for the underlying stock, and it also influences a trader's trading aggressiveness [22].

For the global FX spot market, it is operating with the quote-driven mechanism instead. In this market, prices are determined from bid and ask quotes made by market makers, which are the major banks mentioned above. The market makers are forced to meet their bid and ask quotes, so order executions are guaranteed, and therefore, a quote-driven market is more liquid than an order-driven market due to the efficient order matching mechanism but lacks transparency, because there is no single insuitation to keep all the records of trading. From the analyst's point of view, the lack of transparency increases the difficulties in anticipating future price movements. If taking a snapshot of the market at a particular instance, we only observe two values of bid and ask prices, and they do not include any information

to influence future prices. While for the order-drive stock market, there is a whole LOB with deeper levels of order at different prices. When the orders at level 1 have been executed, those at level 2 will move to level 1 and wait to be executed in the future. Therefore, it can be considered as the potential drivers of price movements are observable at an early instance, and this typical market mechanism contributes to predictabilities.

One possible way to have more predictabilities in the quote-driven FX market is to increase the frequency of market observations, which is to access high-frequency tick prices. Conventionally, most datasets and studies published on the FX market are based on low-frequency, regularly spaced data points, i.e. daily or hourly. There are emerging concerns from academia and industry on analysing the market at a different scale. The high-frequency tick prices are collected at much smaller and more irregular intervals than the usually accessible price information. Some people believe that financial time series has recurrent graphic patterns which can be identified, and before the pattern appears, there are 'latent signals' generated in the market [23]. Many studies try identifying the signals through high-frequency tick prices in the FX market. For example, Alam et al. used intraday high-frequency prices to model the volatility linkage between the oil and foreign exchange market [24]. High-frequency trading is considered an evolution of trading methodology. High-frequency traders implement sophisticated technical analysis methods combined with powerful computational resources to seize the fleeting opportunities in the market [25].

High-frequency FX prices are expensive to obtain, and the data volume is usually intensive. Until recent years when big data methodologies have significantly improved, people started to consider exploring high-frequency FX ticks prices. A pure mathematical analysis from Martens and Zein suggests using high-frequency intraday squared returns to predict realised volatility is better than implied volatility. Their experiments covered stock, FX and

commodities markets [26]. A couple of studies also focus on predicting high-frequency FX data with neural network models. Kablan and Ng created an adaptive neuro-fuzzy inference system for financial trading, which learns to predict the movements of intraday tick data at a five-minute time horizon [27]. Their model was very similar to a simple neural network with four hidden layers, but the activation function (they called the membership function) and model calibrate process are different to the modern neural network models. Choudhry et al. examined the use of market microstructure variables to classify the short-term returns of log prices on inter-dealer spot markets for three FX pairs; their work is based on a multilayer feed-forward neural network with sigmoid activation function, which is a much-advanced model structure [28]. Villa and Stella also implemented a continuous time Bayesian network classifier for a similar task, taking only bid and ask prices as inputs. They concluded the continuous time classifier could surpass a dynamic Bayesian network [29].

2.2.2 Efficient Market Hypothesis and Technical Analysis

The efficient market hypothesis (EMH) is fundamental for analysing and forecasting financial time series. The security market was believed to be highly efficient, where all known information about an individual stock and the stock market as a whole is reflected in the current stock prices. Any new information spreads quickly and is incorporated into the prices of stock without delay [30]. Therefore, the stock price is a random walk process, where tomorrow's price change will only be influenced by tomorrow's information, and it is independent of today's price. The arrival of information is random, so studying the historical stock prices (i.e. technical analysis) will not produce any better predictions than a purely random guess. This is the weak form of efficient market hypothesis [31].

There is also a semi-strong form of the efficient market hypothesis. In addition to the

weak form, it also disputes the effectiveness of fundamental analysis, which is to research a company's financial statement and macroeconomic factors to determine a stock's intrinsic value. Therefore, it is believed that a manually selected portfolio of stocks is neither generating a higher return than a randomly selected alternative, which means stocks are neither overvalued, nor undervalued in the market [32]. However, people could utilise private information to generate profit from trading [33].

The last one is the strong form of the efficient market hypothesis, where none of the technical and fundamental analyses could produce higher profits than the market average. People who subscribe to this theory believe that all public and private information is reflected in market prices. However, in a much-cited paper, Finnerty concluded that people with inside information could find profitable opportunities in their own companies' stocks [34], which indicated the strong form of the efficient market hypothesis might not be applicable in reality.

Technical analysis is the method to study historical prices of financial assets [35]. Although for all three forms of market efficient hypothesis, technical analysis is considered not helpful for predicting future prices. Academics and investment professionals are still intensely interested in using and developing models based on technical indicators. Lo and MacKinlay implemented a simple volatility-based specification test on weekly stock returns in the UK market. In this well-cited paper, they showed that historical prices could be useful for forecasting future returns and thus rejected the EMH [36]. More recently, Sadeghi et al. proposed an ensemble support vector machine to classify the market signals using daily prices from the FX market, and their method utilised five technical indicators [37]. Ni and Yin proposed a hybrid system combining regressive neural networks and support vector machine (SVM) for modelling and predicting daily FX prices [38]. Their approach unities four technical indicators and a genetic algorithm for integrating the trading rules. They confirmed

that the performances of this hybrid system are better than those of other global models, such as GARCH. From an industrial perspective, Farimani combined technical indicators with market emotional distribution of news as features, and using recurrent neural networks to predict prices in the FX and Cryptocurrency market [39].

2.2.3 Futures, Option, and Hedging

Financial derivatives are financial contracts that are 'derived' from their underlying financial assets such as stock, bonds, currencies, funds and commodities. Futures, also called futures contracts, are agreements between obligate parties to buy or sell an asset at a pre-determined future date and price. The buyer must purchase or the seller must sell the underlying asset at the set price, regardless of the current market price at the expiration date [40]. Futures are one of the fundamental types of financial derivatives.

Options are another basic building blocks of financial derivatives [41]. There are two types of stock options: call option and put option. A vanilla call option is the contract in which the holder has the right but not the obligation to buy the underlying stock for a pre-determined price K (strike price) at a pre-determined date T (expiration date) [42]. For a vanilla put option, the holder has the right but not the obligation to sell the underlying stock. For the research project to be discussed in Chapter 4, we will focus on European call options, wherein the buyer can only execute it at expiration. The opposite category is American call options, wherein buyers can exercise their rights anytime before expiration. There are also Bermuda options, which combine European and American styles. Bermuda options can be exercised only at several pre-determined dates before expiration. It is obviously more sophisticated to model American or Bermuda stock options than a European one.

In financial terms, hedging means making transactions to mitigate the risks of other fin-

ancial exposures. For example, a UK manufacturer sells its products to the US and receives US dollar payments in one month. There is a risk that the US dollar (USD) will reduce its value again to the British pound (GBP) during one month. Therefore, the manufacturer would like to hedge its risk of receiving less GBP than expected by entering a currency derivative contract with a bank. The manufacturer will pay the bank in USD, and the bank will pay GBP back to the manufacturer with a fixed exchange rate. Of course, the manufacturer will need to pay a certain amount of upfront premium. This type of financial derivative is called a currency swap contract.

We are focusing on a European-style stock call option in this thesis and hedging the option contract from the seller's perspective. The buyer of the option has the right to buy the underlying stock at the expiration date. Therefore, as the seller, we must hold a certain amount of the underlying stock during the contract period to prepare for the buyer's execution. The optimal holding amount of the underlying stock is directly influenced by the buyer's likelihood of executing their right. It is radically determined by the difference between the current stock price S_t and the strike prices K of the option contract.

In the following sections, we will introduce the mathematical framework widely adapted by academics and industries to model stock price processes and calculate the hedging strategies for European-style stock options.

2.2.4 Geometric Brownian Motion

Brownian motion was first described by Scottish biologist Robert Brown in 1828 [43]. Robert published a pamphlet about his observations that various pollen particles have similar random oscillatory motions in water under a microscope [44]. From modern chemistry, we now understand that these phenomena of pollen particles are caused by their constant collision

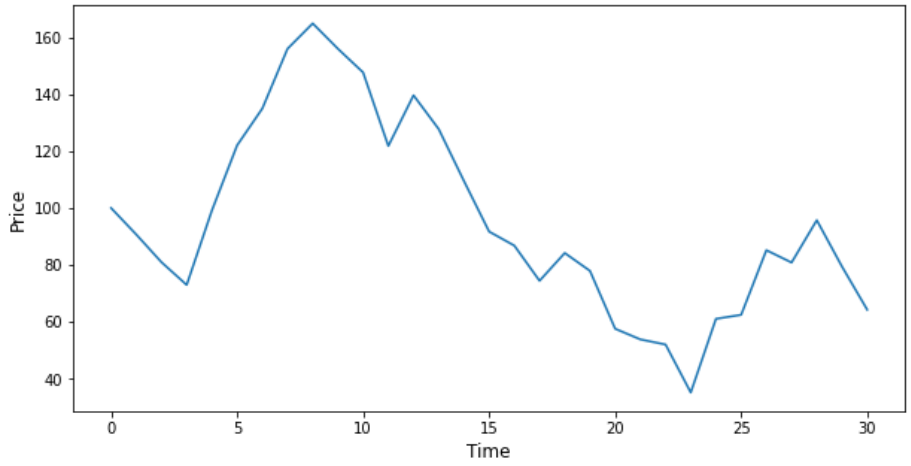


Figure 2.2: An Illustration of Geometric Brownian Motion

with other fast-moving molecules.

In mathematics, we use the Wiener process to describe Brownian motion. A Wiener process is a stochastic process $\{B_t\}_{t\geq 0}$, which satisfying the following conditions:

1. $B_0 = 0$.
2. $\{B_t\}$ is continuous in t.
3. for all $0 < t_1 < t_2$, the increment $B_{t_2} - B_{t_1}$ follows a normal distribution $N(0, t_2 - t_1)$.
4. for all $0 < t_1 < t_2 < t_3 ... < t_n$, the random variables

$$B_{t_2} - B_{t_1}, B_{t_3} - B_{t_2}, B_{t_4} - B_{t_3}, ..., B_{t_n} - B_{t_{n-1}}$$

are independent to each other

Once we have a Wiener Process (Brownian Motion) and an initial value S_0. We want to formulate the distance between the future asset price S_t and the starting asset price S_0 as a Wiener process with a linear trend. The process $\{S_t\}$ became Brownian Motion with drift, which states as:

$$S_t - S_0 = \mu t + \sigma B_t \tag{2.1}$$

And its stochastic differential equation (SDE) is:

$$dS_t = \mu dt + \sigma dBt \tag{2.2}$$

Where μ is the drift parameter indicates a long-term trend of this process, and σ is the scale parameter that controls how my randomness contributes to the process.

The problem with the drifted Brownian Motion process is that randomness W_t is directly embedded in the price process S_t. Therefore, there are possibilities that S_t became negative if randomness B_t is a significantly negative number. However, for financial asset prices, we can't have negative prices. Therefore, we make some modifications and define the Geometric Brownian Motion (GBM) with SDE:

$$dS_t = \mu S_t dt + \sigma S_t dB_t \tag{2.3}$$

By using the initial value of S_0, we could integrate the equation using ito's lemma [45], and obtain:

$$ln\frac{S_t}{S_0} = (\mu - \frac{\sigma^2}{2})t + \sigma B_t \tag{2.4}$$

GBM is alternatively modelling the logarithm of asset price growth as a drifted Brownian Motion instead of the prices. This is because the growth rate $(\mu - \frac{\sigma^2}{2})t + \sigma B_t$ can go up and down, but the values of asset S_t can no longer be negative. Therefore, Geometric Brownian Motion is also called log-normal growth process [46].

An illustration of GMB is shown in Figure 2.2, where the path is simulated with parameters $S_0 = 100$, $\mu = 0.01$, and $\sigma = 4$.

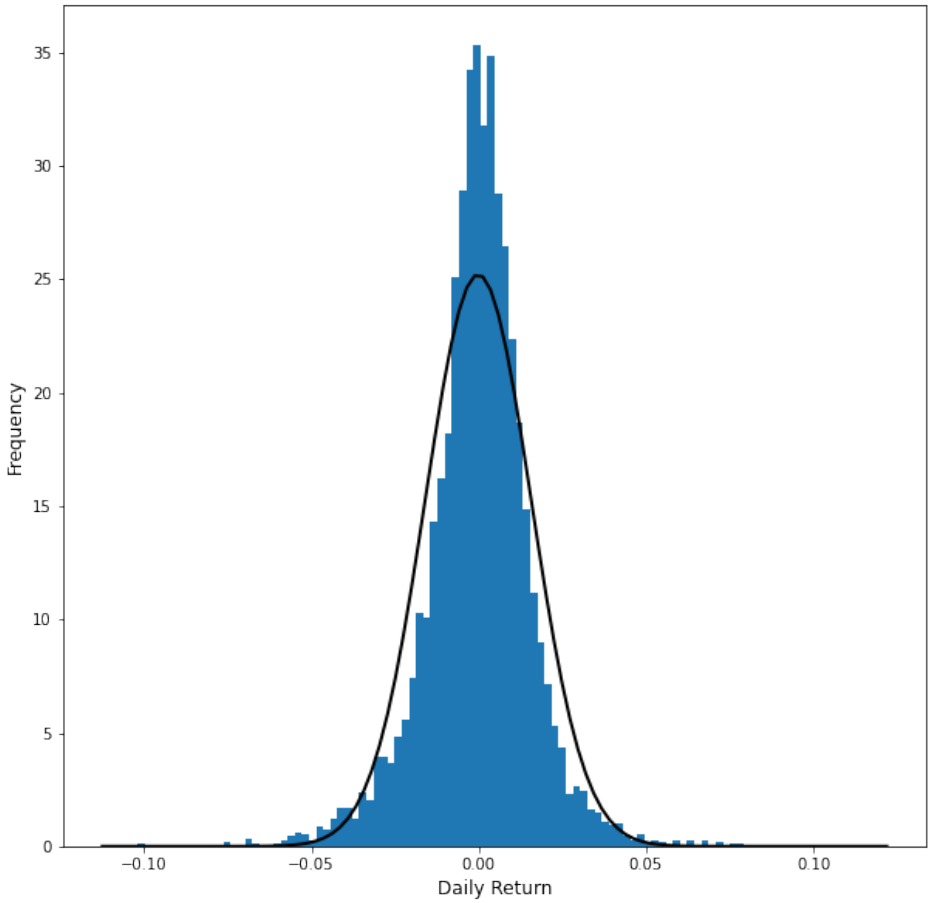


Figure 2.3: Distribution of Daily Stock Returns From 28 US Stocks in 2018

Now we have a proper mathematical model to represent the price movements of a financial asset (i.e. stock). We will then discuss how to derive the prices and trading strategies of a financial derivative (i.e. European option) based on this asset.

2.2.5 Black-Scholes-Merton

In the early 1970s, Black and Scholes [47], and independently, Merton [48] initiated the classic parametric framework for option valuation and hedging, which we refer to as the Black-Scholes-Merton model. It first assumes that stock prices are following the Geometric

Brownian Motion (GBM) process discussed in the previous section:

$$dS_t = \mu S_t dt + \sigma S_t dB_t, \tag{2.5}$$

where S_t represents a stock's price at the time t, μ is the drift, B_t is a Brownian motion contributing uncertainty, and σ controls volatility.

We also have a European call option contract derived from this stock S_t with an option value of $C_t(S_t, t)$. We think the option value is influenced by t, which is the time point when calculating the option value.. As t approaches the expiration date T of the option, its value would decrease. Another influencing factor is the current stock price S_t. For example, if S_t is much smaller than the strike price K, then the option buyer is unlikely to exercise the option, so the option price $C_t(S_t, t)$ would be close to zero. By using Itô's lemma [45] on C_t and substituting Equation 2.5, we obtain:

$$dC_t = \left(\mu S_t \frac{\partial C_t}{\partial S_t} + \frac{\partial C_t}{\partial t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 C_t}{\partial S_t^2}\right) dt + \sigma S_t \frac{\partial C_t}{\partial S_t} dB_t. \tag{2.6}$$

We want to replicate the value of $C_t(S_t, t)$ with a self-financing portfolio P. This portfolio consists of only two assets, which are y_t amounts of the underlying stock and x_t of cash. We consider cash a risk-free asset and accumulates at the risk-free rate r. Therefore, we calculate the value for the portfolio as follows:

$$P_t = x_t e^{rt} + y_t S_t. \tag{2.7}$$

We could calculate its total derivative and substitute Equation 2.5 into the equation. We have:

$$dP_t = (r x_t e^{rt} + y_t \mu S_t) dt + \sigma y_t S_t dB_t. \tag{2.8}$$

We could notice that dP_t has a similar formula with dC_t, where they are both the sum of a proportion of dt and a proportion of $\sigma S_t dBt$. More importantly, we want the portfolio P to

perfectly replicate the option C_t, which means any changes in t and B_t have the same effects on both portfolio P_t and option contract C_t. Therefore, we can equate terms in Equation 2.6 and Equation 2.8 to obtain:

$$y_t = \frac{\partial C_t}{\partial S_t} \tag{2.9}$$

$$x_t = \frac{\frac{\partial C_t}{\partial t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 C_t}{\partial S_t^2}}{re^{rt}}. \tag{2.10}$$

According to Equation 2.9 and Equation 2.10, investors could continuously adjust their portfolio P, which is made up of y_t amount of underlying stock and x_t amount of cash, to always perfectly hedge the risk from the liability with the stock option C_t. In this way, they will always achieve zero profit or losses at the option's maturity. Equation 2.9 effectively calculates the partial derivative of C_t with respect to S_t, which is usually called the delta hedging strategy.

We could further substitute Equation 2.9 and Equation 2.10 back into Equation 2.7 and make $P_t = C_t$ to obtain the famous Black-Scholes-Merton partial differential equation.

$$\frac{\partial C_t}{\partial t} + rS_t \frac{\partial C_t}{\partial S_t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 C_t}{\partial S_t^2} - rC_t = 0. \tag{2.11}$$

So far, we have introduced the hedging strategy if we sell a European call option, and we also need to know how much we will charge for signing the option contract at $t = 0$. The Black-Scholes-Merton equation could be solved to get an analytical solution to $C_t(S_t, t)$. We first need to set a couple of boundary conditions on $C_t(S_t, t)$, assuming the strike price for the option is K, the option expires when $t = T$ and risk-free interest rate of r:

1. $C_T(S_T, T) = max(S_T - K, 0)$.

2. $C_t(0, t) = 0$ for all t.

By solving the Black-Scholes-Merton PDE with these conditions, we have a formula for

pricing the European call option [40]:

$$C_t(S_t, t) = S_t N(d_1) - K e^{-r(T-t)} N(d_1 - \sigma\sqrt{T-t}). \tag{2.12}$$

where

$$d_1 = \frac{ln(\frac{S_t}{K}) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}. \tag{2.13}$$

In Equation 2.12, the function $N(d_1)$ is the cumulative distribution function of standard normal distribution, which states as:

$$N(d_1) = \int_{-\infty}^{d_1} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx. \tag{2.14}$$

This is the complete Black-Scholes-Merton framework for calculating option price and hedging an option contract.

Black-Scholes-Merton (BSM) is considered a benchmark in every literature and is widely applied in the industry. The model provides beautiful closed-form solutions, as stated above. Investors could manage their hedging positions by calculating “Greek Letters”. For example, we have seen delta hedging in Equation 2.9, which is basically how the option price reacts to the change in stock price. There are also other Greek Letters measuring the relationship between an option’s price and different quantities [49].

Despite its popularity, the BSM model is built on idealised assumptions that are not applicable in real-life scenarios. First, the underlying price is modelled as a GBM process with constant volatility σ. Therefore, it cannot model the fat tails of observed probability density, which means extreme high/low stock returns are observed much more frequently than assumed [50]. The distribution of actual stock returns is shown in figure 2.3. We calculate the daily percentage return of 28 list stocks in the US market during 2018 and plot the distribution of around 7000 numbers. We also plot a normal distribution using the mean of 0 and standard deviation of 0.0158, the realised mean and standard deviation calculated from these

observations. We could see from the figure that extreme returns, which are higher (lower) than 0.05 (-0.05), appeared much more often than the normal distribution. This gives rise to underestimated risks and is one of the causes of the financial crisis in 2008 [51]. Second, it assumes there are no trading costs and trading limitations for every participant in the market. Third, traders should continuously re-balance their positions to achieve zero profits (losses) at the contract's maturity, which is financially and practically infeasible. Therefore, when the option expires, there is always a loss for the option issuer (seller). The actual option premium is based on the theoretical price and the discounted values of the expected losses.

2.2.6 Stochastic Volatility Model

From the above sections, we know that Geometric Brownian Motion (GBM) is one of the simplest models to simulate the price processes of financial assets. However, as stated in Equation 2.5, the fundamental drawback of GBM is the constant volatility value. Therefore, even though GBM works perfectly with the Black-Scholes-Merton (BSM) framework, which gives beautiful mathematical solutions to option prices and hedging strategies, we cannot rely only on them to trade in the real world.

We could use stochastic volatility models to simulate the underlying prices to take one step closer to reality. As its name indicates, there is a stochastic element in the volatility of the underlying price process. Heston model is the most classic stochastic volatility model [52]:

$$\begin{aligned} dS_t &= \mu S_t dt + \sqrt{v_t} S_t dB_t^1. \\ dv_t &= \kappa(\theta - v_t)dt + \sigma\sqrt{v_t} dB_t^2. \end{aligned} \tag{2.15}$$

In the above stochastic differential equations, B_t^1 and B_t^2 are two one-dimensional Brownian motions, with correlation in $[-1, 1]$. v_t controls the volatility of S_t, a mean-reverting stochastic

process instead of a constant. In the stock market, we usually observe a high volatility period after a significant change in stock price. This phenomenon is modelled by a high correlation between B_t^1 and B_t^2 and the mean-reverting property of S_t. The parameter σ is called the volatility of the volatility, which means how much the volatility σ is swinging around its theoretical mean v_t. If the value of σ is high, the volatility v_t is likely to have extremely values because the stochastic process B_t^2 is making more contributions.

2.2.7 Fractional Brownian Motion and rough Bergomi model

Fractional Brownian Motion (fBm) is another type of stochastic process which recently raised lots of interest in physics [53], and energy price modelling [54]. The most significant characteristic of fBm is that the increments at every step are not independent. Therefore, the values of this random process at every step have the following covariance function:

$$\mathbb{E}[B_t^H B_s^H] = \frac{1}{2}(|t|^{2H} + |s|^{2H} - |t-s|^{2H}). \tag{2.16}$$

Where H is called the Hurst parameter, which controls the level of covariance, it can be noticed from Equation 2.16, when $H = \frac{1}{2}$, the fBm process becomes a classic Brownian motion which covariance between different steps are zero.

Gatheral et al. [55] were the first to bring the ideal of Fractional Brownian Motion into the financial context. They calculated spot price volatilities for four different assets (DAX futures contracts, Bund futures contracts, S&P index and NASDAQ index). They discovered that the difference in log volatility at a fixed step size has a linear relationship with the step size. Therefore, they conclude that the log-volatility process of these assets may be modelled using fBm, and they also estimated their Hurst parameters H, where the values were between 0.06 and 0.2.

Bayer et al. [56] extended this idea from discrete steps to continuous time. They found it is the non-Markovian version Bergomi model and therefore named it the rough Bergomi (rBergomi) model. They tested that the rBergomi model could fit the implied volatility surface of SPX very well. The rBergomi model is formulated as follows:

$$
\begin{aligned}
S_t &= S_0 \exp\left\{\int_0^t \sqrt{V_u}\mathrm{d}W_s - \frac{1}{2}\int_0^t V_u \mathrm{d}u\right\}. \\
V_t &= V_0 \exp\left\{\eta Y_t - \frac{\eta^2}{2}t^{2H}\right\}. \\
Y_t &= \sqrt{2H}\int_0^t (t-u)^{H-\frac{1}{2}}\mathrm{d}W_u. \\
W_u &= \rho B_u^1 + \sqrt{1-\rho^2}B_u^2.
\end{aligned}
\tag{2.17}
$$

In the above equations, S_t is the asset price at time t, and V_t represents its spot variance. W_u is equivalent to a combination of two independent Brownian motions B_u^1 and B_u^2 with correlation parameter ρ. According to [57], Y_t is a Volterra process and a fractional Brownian Motion. It can be noticed that the rBergomi is similar to the stochastic volatility process discussed in Section 2.2.6, but the spot volatility process V_t is modelled using fBm. H is the Hurst parameter, and $H = 0.5$ indicates no correlation between the increments of the process. $H < 0.5$ when the correlation is negative and $H > 0.5$ is positive.

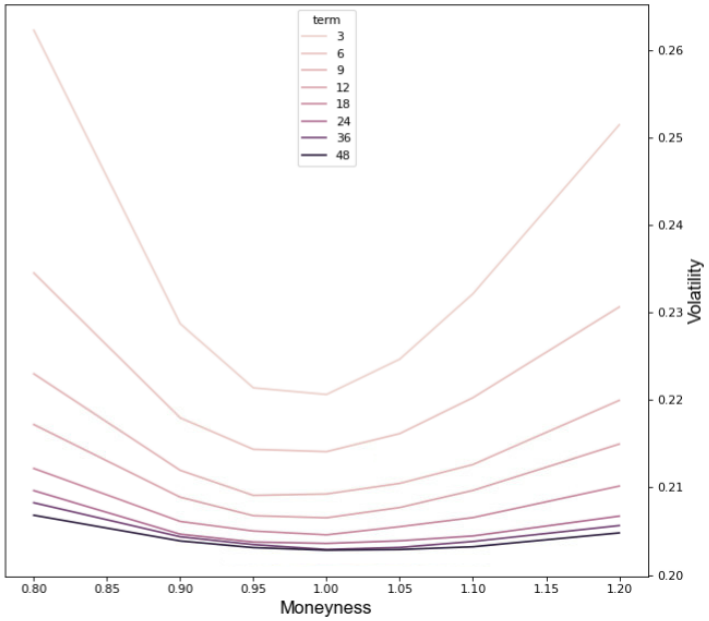
For our subproject to be discussed in Chapter 4, We simulated the underlying stock prices with GBM, Heston and rBergomi processes. We experimented with various parameters to present different market scenarios. We want to compare the hedging performances of our proposed neural network mechanism under different simulations, as well as compare it with the benchmark BSM method.

2.2.8 Efficient Frontier

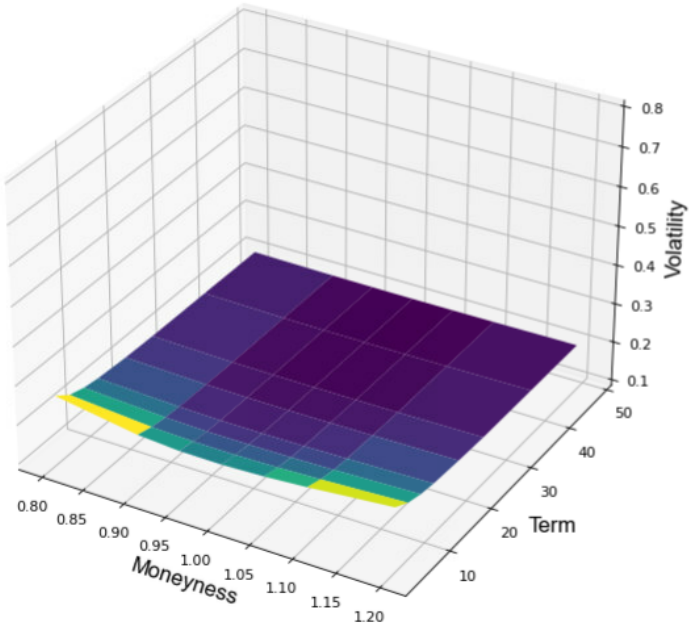
The efficient frontier is an essential concept from portfolio theory, and the theory was derived by Markowitz in 1952 [58]. It proposed that by combining stocks into a portfolio, the expected variance of returns would reduce but not change the expected return. Markowitz's paper was the first to provide a mathematical formulation of diversification of investment [59]. The most influential concept that Markowitz delivered in his paper was that we should consider one asset's contribution to the risk of a whole portfolio instead of looking at it in isolation. This is the basis of modern financial economics.

Based on Markowitz's work, Merton created the concept of efficient portfolio frontier in 1972 [60]. He provided an analytical formulation and proved that with a fixed set of risk assets, with different combinations of these assets, the relationship between the portfolio's expected variance and the portfolio's expected return is a parabola. Thus, investors could balance their positions by varying the portfolio composition to achieve their desired return or level of risk.

We borrow this idea of Efficient Frontier and apply it to the neural network model of deep hedging. We modify the standard deep hedging pipeline so investors can select their desired hedging frequencies instead of the fixed daily hedging. If hedging is performed less frequently, the contract results in fewer hedging losses but increases final values' volatility (uncertainty). We could plot the relationship between the mean for the termination losses and volatility for the termination losses on a plane and notice it is a parabola. We call it the Efficient Hedging Frontier (EHF). More detailed discussions will be shown in chapter 4.



(a) The Volatility Smile



(b) A Synthetic Impied Volatility Surface

Figure 2.4: Volatility Smile and Implied Volatility Surface

2.2.9 Implied Volatility Surface

We discussed the Black-Scholes-Merton method for calculating the price of stock options in Section 2.2.5. To simplify the model, we write the price of an option contract as a function of various pricing parameters:

$$C_t(S_t, t) = BSM(t, T, S_t, K, r, \sigma) \tag{2.18}$$

From Equation 2.18, we know that there are seven parameters to work out the option price. Among these parameters, some values are directly observable from the open market (r, S_t), some are specified in descriptions of the individual option contract (T, t, K), and there is only one parameter that is playing a crucial role in differentiating our price of the option product to our competitors. Put another way, every participant in the option market needs to forecast, from their knowledge and experience, the future volatility of the underlying asset's price. This is represented as the σ parameter in the option pricing formula and therefore reflected in the option contract's price. If this prediction is closer to the truly realised volatility from the market, the option insurer would have better hedging of the option contract. Otherwise, they may lose money for the over/under-estimated stock price volatility.

The σ parameter discussed above is equivalently the implied volatility. As mentioned in [61], the formal calculation of implied volatility is that, for a particular trading day t, read the market prices of various option contracts with different combinations of time to expiration $\tau = T - t$ and moneyness $m = \frac{K}{S_t}$. Then, put the observed market price into Equation 2.18 together with other observable parameters, and find the best value of σ, which satisfies the pricing formula. Because we know that, even for the same trading day, different τ and m combinations have different option prices. Therefore we come up with different values of $\sigma(\tau, m)$, which is a function of time to expiration τ and moneyness m. This also emphasises

that constant volatility assumptions in GBM and BSM methods are invalid in the real market.

We could get a surface if we plot implied volatility $\sigma(\tau, m)$ against a grid of different values of time to expiration τ and moneyness m. There are typical features of an implied volatility surface. First, if we slice the volatility surface by fixing the time to expiration τ value, we could get a curve showing the relationship between moneyness m and implied volatility $\sigma(m)$. We will notice that when m increases from zero to one, the $\sigma(m)$ value decreases. when the value of m is increasing away from one, the $\sigma(m)$ value raises again. It may not always be the case for the minimum of σ to be around $m = 1$, but the shape of a smile face is similar for all assets and all dates. This observation is widely referred to as the volatility smile. We generate a synthetic implied volatility surface and demonstrate its volatility smile in Figure 2.4a. If we consider a call option, the right section of a volatility smile where $m < 1$, we consider option contracts are 'in-the-money' contracts because the option contract has positive intrinsic value [62]. The opposite contracts, where $m < 1$, are 'out-of-the-money'. Suppose the 'at-the-money' contracts have higher implied volatility than the 'out-of-the-money' contracts. We consider that the implied volatility has a skewness, which means it is not symmetrical. The volatility smile and skewness are the most important characteristics of implied volatility surfaces. Figure 2.4b shows a sample of generated synthetic volatility surface.

2.3 Deep Learning Models

Deep Neural networks have been a popular topic across many academic areas and industries. In Section 1.1, we have mentioned several recent advanced applications of neural networks, and they utilised different neural network model structures to fit their purposes

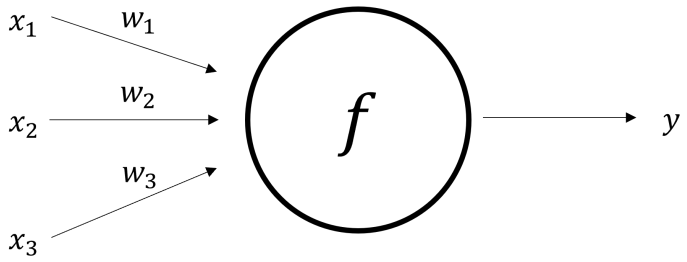


Figure 2.5: An Artificial Neuron

of various kinds. In this section, we will first explain the low-level building block of neural networks — a neuron, and introduce the fully connected neural network as the first network structure. Then we will continue to other high-level arrangements of neurons, which are fitted for different purposes, such as convolutional neural network, recurrent neural network and auto-encoder. After that, we try to explain why nested computing units (a neuron network) can perform various tasks through the universal approximation theory. Finally, we introduce an important neural network application for trading and hedging, which is the Deep Hedging model.

2.3.1 Fully Connected Layers

A neural network contains many single computing units called artificial neurons or neurons. It was designed to emulate a human neuron in many ways [63], and for a single neuron, it only performs simple non-linear computation defined by its activation function. We show a simple illustration in Figure 2.5.

We can see from the figure that a neuron takes some input values and outputs a single value. The mathematical formulation is:

$$y = f(\sum_{i=1}^{k} w_i * x_i + b) \tag{2.19}$$

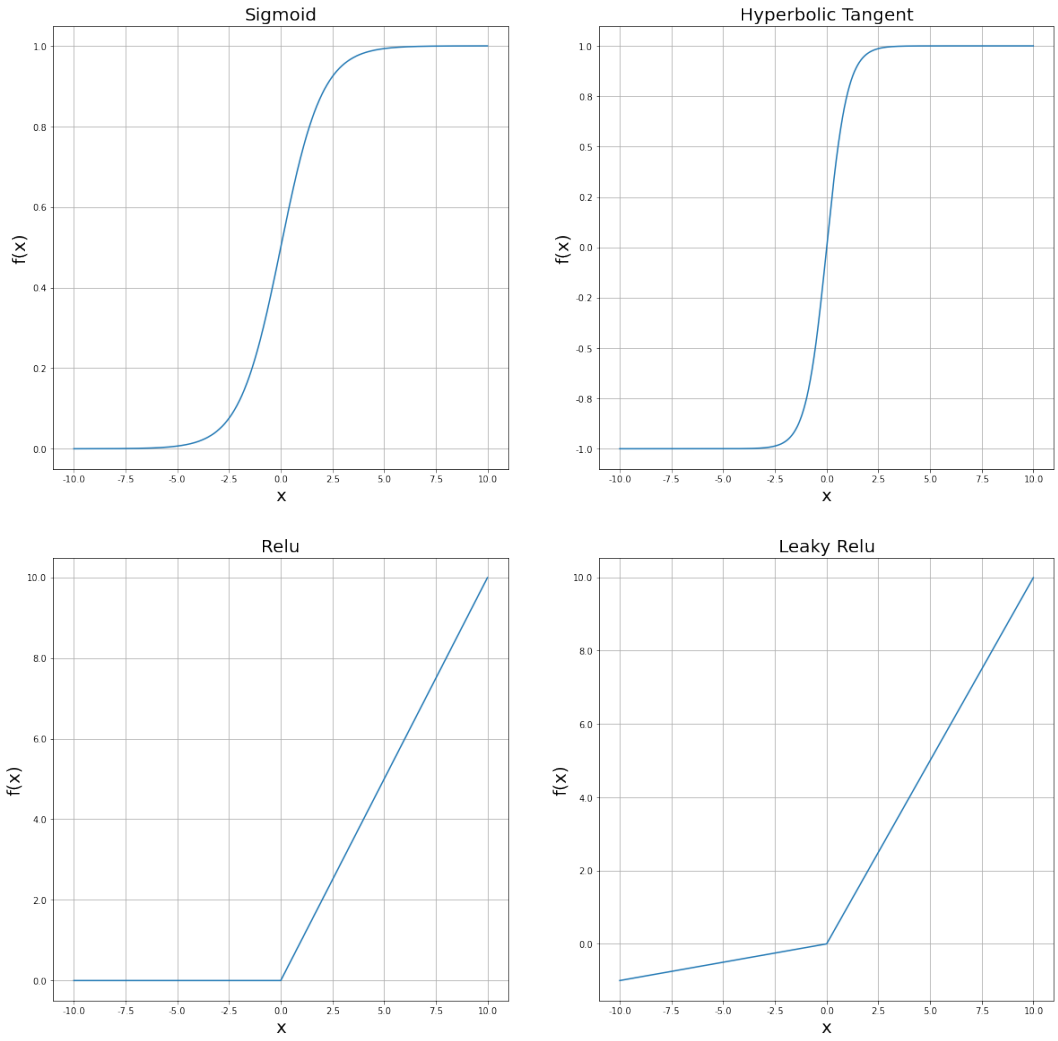


Figure 2.6: Activation Functions

In the above equation, k represents the number of input values to a single neuron, and w_i is the weight associated with the i-th input. We can see that input values are linearly combined and then given to the activation function f. The b is a bias term added to the linear combination.

The activation function does not need to be very complicated. But it has to satisfy two requirements [64]. First, it has to be a non-linear function. This is obvious, as nested linear functions are still linear functions, which could not perform non-linear transformations.

Second, the action function should be differentiable for any input value because we use backpropagation [65] methods to update the parameters (weights) in the neural network, and we need to calculate the derivative of the activation function. The most commonly used activation function nowadays is the sigmoid function, the hyperbolic tangent (tanh) function, the rectified linear unit (ReLu) function and the Leaky Relu function. They are shown in Figure 2.6.

From our experiments on various projects with neural networks, we found the selection of activation function is not the most important for a neural network model. The only consideration of the activation function is the range of output. For some tasks of financial prediction, the output value is not bounded between 0 and 1, for example, the volatility of stock prices. Therefore, it would not be appropriate to select the sigmoid function as the final output activation, as any value larger than one will never be obtained from the network. Apart from the range of output, the selection of activation function mainly influences the efficiency of model training. For computer vision related tasks, it is believed that the ReLu function and its variants help the neural network converge faster during model training than the sigmoid function [66].

The number of neurons, and particularly how they are connected to each other, is more critical when designing the neural network for a specified purpose. For example, we could combine many neurons in one layer to take the same input values with different weights and connect many layers to construct a fully connected neuron network (also called a multi-layer perceptron neural network). A sample is illustrated in Figure 2.7. We define a fully connected neural network structure by its number of layers and the number of neurons at each layer. With various sizes of neurons and layers, the neural network model has the flexibility for modelling datasets with different sizes and features. For example, in 1989, Yann LeCun

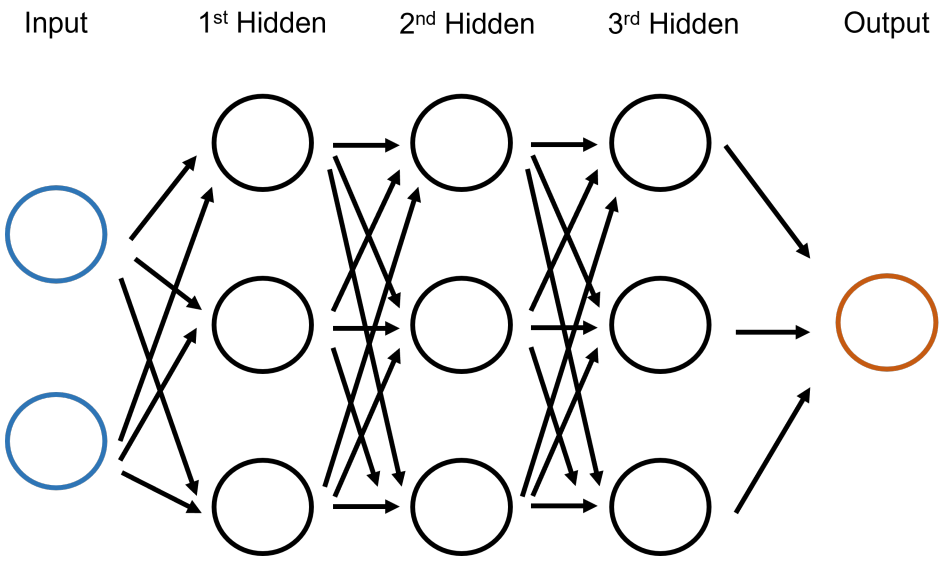


Figure 2.7: Fully Connected Neuron Network

used a fully connected neural network to recognise handwritten digits of postcodes provided by the US postal service. The dataset consists of 9298 segmented numerical digits [67]. The dataset was considered extensive when the paper was published, and this is the first time a neural network model is applied to a real-life application.

The functionality or ability of a neural network model is effectively controlled by its loss function during training. The loss function provides a direction for the neural network to be optimised (trained) to. Because neural network models are too flexible, this 'direction' defines the functionality of a neural network model. The selection of the loss function should fit the data's nature and the model's purpose. Different loss functions lead to very different models, even for the same dataset and model structure. For example, Zhao et al. [68] compared a couple of other loss functions in several areas of computer vision and concluded that loss function influences not only model performances but also training efficiencies. Deep Hedging is defined as a particular loss function to evaluate the value-at-risk (VaR) of option contracts. The VaR is a typical risk measurement widely used by financial institutions and regulations

[17]. For our research subproject of implied volatility surfaces to be discussed in Chapter 5, we also try different formations of loss functions to enforce various characteristics on the generated synthetic surfaces. The most common loss function used in neural networks is mean square error (MSE) and mean absolute error (MAE). These two functions compare the neural network output with the expected output (truth) and calculate their differences. The formulae are:

$$L_{MAE} = \int_{i=0}^{N} |y_i - \hat{y}_i| \tag{2.20}$$

$$L_{MSE} = \int_{i=0}^{N} (y_i - \hat{y}_i)^2 \tag{2.21}$$

In the above equations, we assume there are N data points in the dataset, and L_{MSE} is more sensitive to the difference between the model output and the truth. When the difference $y_i - \hat{y}_i$ is a value that is less than one (e.g. daily stock return), then the square function makes the loss value smaller, where sometimes it is difficult for a neural network to decide how to adjust its weights. In this case, L_{MAE} is a better candidate.

After defining the loss function, the weight and bias terms in the network should be optimised to generate the minimum loss value with the training data and validation data if validation data exists. The progress is called model training or model optimisation. This is a two-step process. The first step is calculating the derivative of the model output's loss value with respect to the model parameters (weights and bias). This is called backpropagation, and it implements the chain rule of calculus to calculate derivatives [69]. The second step is to modify the parameters in the neural network according to the calculated derivatives to reduce loss value. This iterative process runs until the local minimum is found or the re-

quired value of the loss is reached. This is usually referred to as gradient descent. Most of the time, we use stochastic gradient descent (SGD) to optimise the parameters of a neural network model, and it uses estimated gradients instead of actual gradients. The estimated gradients are calculated from a random batch of input data, beause calculating the gradients from the whole input dataset could cost lots of time and computer memory. There are many SGD methods available in the popular python package of Pytorch and Tensorflow, such as Adam and resilient backpropagation (Rprop) [70][71]. Both Adam and Rprop are adaptive stochastic gradient descent methods, which means their learning rate is variable during the optimization process, and this will be discussed in the next paragraph.

The selection of an optimiser would not determine what a neural network model is capable of. Still, it determines whether or not the model finds the best parameters to exert its abilities. Sometimes, for a particular combination of dataset and model, several optimisers could reduce the training loss, but the real issue is that one could achieve a training loss that is small enough to enable the trained model to perform its designed functionality. We could easily observe the model being trained (i.e. training loss reducing), but not particularly useful (i.e. training loss small enough). There is another essential hyperparameter to consider, called the learning rate. It controls how much to change the model parameters in response to the derivative of loss during model training. It can also be considered the 'step size' when searching for an optimal point on the surface of all possible parameter values. If the 'step size' is too large, the searching process may miss the optimal point easily and never find a local minimum of training loss. If the 'step size' is too small, the algorithm may stick to a local minimum, which is not sufficient for the model to be useful. Therefore, the selection of the learning rate is more important than the choice of the optimiser. In practice, some people would like to reduce the learning rate during model training to guide the model

to the lowest possible training loss value. The Adam and Rprop optimisers mentioned above are designed to adaptively change the learning rate during model training in order to find the best parameter space. The Adam method uses the second-order derivative to normalise the learning rate when updating parameters. Rprop keeps the moving average of the squared gradients and divides the learning rate by the square root of the moving average [72] [73].

2.3.2 Convolution Layers

In the previous section, we introduced the fully connected neural network, which is the most classic structure for arranging the computational graph of neurons. This section introduces convolutional neural network structure, typically favoured for computer vision models.

In mathematics, convolution operation measures the amount of overlap for one function g as it shifts over another function f. The measurement usually takes the sum or integrates [74]. For instance, the mathematical expression of convolution over the range $[0, t]$ is:

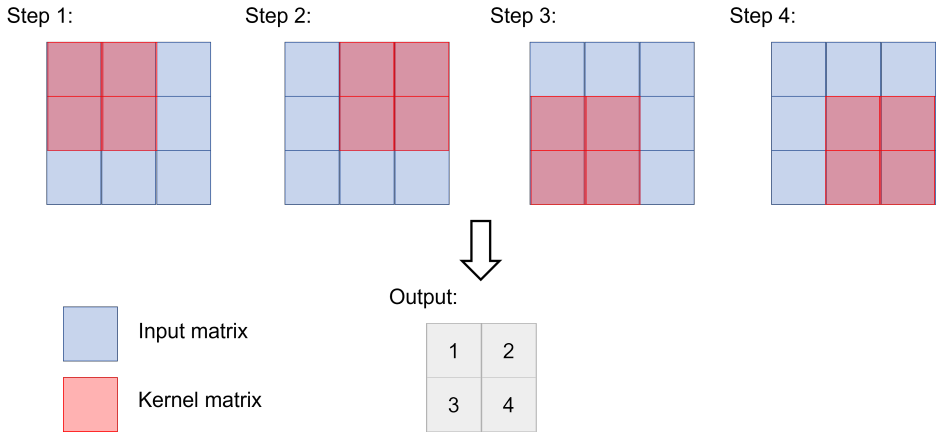
$$[f * g](t) = \int_0^t f(\tau)g(t-\tau)d\tau \tag{2.22}$$

Where f is the input function and g is the kernel function. Before the neural networks came out to the public, convolution was widely adapted in signal decomposition [75], radio communications [76] and many other areas.

When we take the concept of convolution into a neural network, the input function becomes the input matrix to be processed by the neural network, and the kernel function becomes a kernel matrix formed from a neural network layer. For example, when processing an image with 3*3 pixels, if the kernel size is 2*2 and shifts 1 unit to the right every time, the output would be a 2*2 matrix. This is illustrated in Figure 2.8

In Figure 2.8, the output is a 4*4 matrix where each element is computed as a sum of

Figure 2.8: Convolution on an image matrix



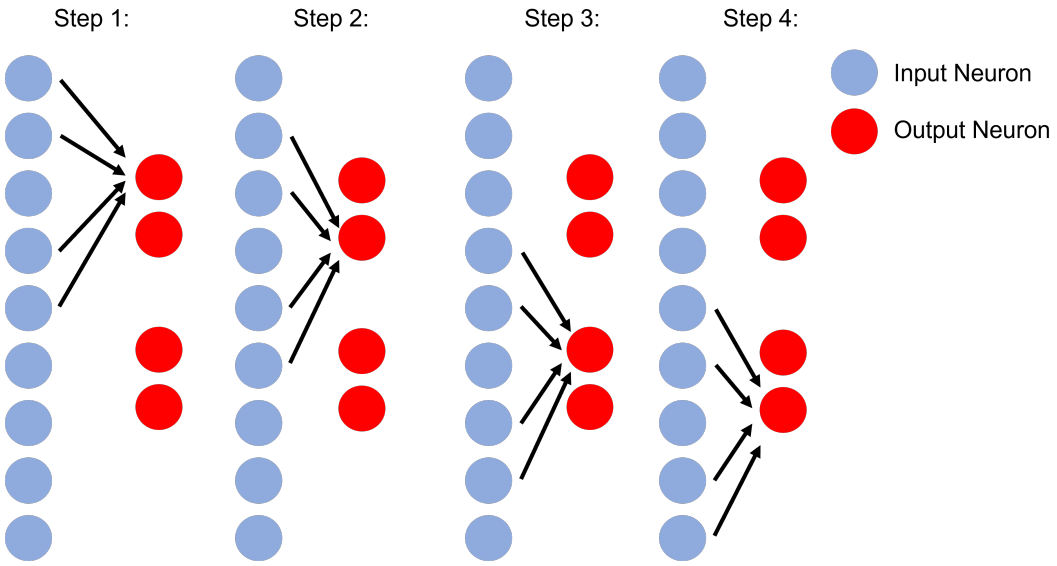
the products of every pair of values between the input matrix and the kernel matrix. The mathematical expression is:

$$y(i,j) = \sum_m \sum_n x(i+m, j+n)\kappa(m,n) \tag{2.23}$$

In the above formula, y is the output matrix for this convolution operation (also called the feature map), x is the input matrix, and κ is the kernel matrix. The range of m and n depends on the size of the kernel, and the kernel size is usually smaller than the size of the input.

We could look at convolution operation from another perspective. If we flatten all the input and kernel neurons, we treat them as long vectors. For example, a 3*3 matrix would become a vector of 9 numbers. We could compare one convolution layer with one fully connected layer, shown in Figure 2.9. We could observe two differences. First, there are fewer connections between the input and output neurons than fully connected ones. This indicates image processing. A convolution layer could detect small areas in an input image matrix (instead of processing the whole image together) and usually identify meaningful and visible patterns. This is referred to as sparse connectivity. It also vastly reduces the memory requirement for the model because of fewer parameters, which increases statistical efficiency during model

Figure 2.9: Convolution operation in flattening view



training. The second difference is called parameter sharing. In a fully connected setting, one weight value is associated with one single input neuron and used only once. For a convolution layer, the connection between input and output neurons in Figure 2.9 is the convolution kernel, which is shared between different input neurons as it slides over the input matrix. So that repeated patterns in the input matrix could be captured, which is more suitable for image processing problems.

The rise and success of modern deep learning are arguably due to the use of Convolutional Neural Networks (CNNs) for computer vision tasks. If a neural network model with most of its layers are convolution layers, we call it a CNN. As discussed above, convolution layers are used as the feature extraction elements in an end-to-end learning system. The epoch-making work is the AlexNet model published in 2012 [77]. The model consisted of five convolution layers and three fully connected layers to classify images into 1000 different classes. Their work utilised a huge benchmark dataset of ImageNet, which contains over 1.2 million labelled high-resolution images. It also inspired many following works to use GPU

(Graphics Processing Unit) to accelerate the training of neural network models, which is now standard practice. After that, CNN-based models have successfully pushed the boundaries in many computer vision applications, such as object detection [78], image and video super-resolution [79], and anomaly detection in videos [80]. In recent years, CNN-based neural network models have also improved many other scientific research fields, which depended on the success of image classification models. For example, skin cancer diagnosis [81], pig face identification for precise breeding [82], and chemical process fault diagnosis [83].

There have been many studies adopting CNN for financial time series analysis. In the stock market, Ghoshal et al. analysed more than 20 famous standard chartist pictograms in the stock market using statistical analysis and the CNN model, where the chartist pictograms are visual patterns in the stock price candlestick graphs, and many people believe these patterns would indicate typical future behaviour (increase or decrease) of stock prices. Their work concluded that there is little evidence of the prediction power of standard chartist pictograms [84]. Sezer et al. took a classical CNN model from the computer vision domain and applied it to classify daily price positions of Dow 30 stocks and 9 Exchange Traded Funds (ETFs). Their method encodes financial time series information into image-like matrices by calculating 15 different technical indicators at various window sizes [5]. In this way, critical features in the historical prices could stand out from many noisy signals. Temporal relationships are also preserved as calculations are performed at various window sizes. Sood et al. proposed a novel method to visually tranfer market charts into matrices of price distributions, and use convolutional neural networks to scan them and perform prediction of fututre price charts [85]. There are also other studies focused on prediction of market price movement [86][87][88][89], and making direct trading decisions in the stock market [90][91][92].

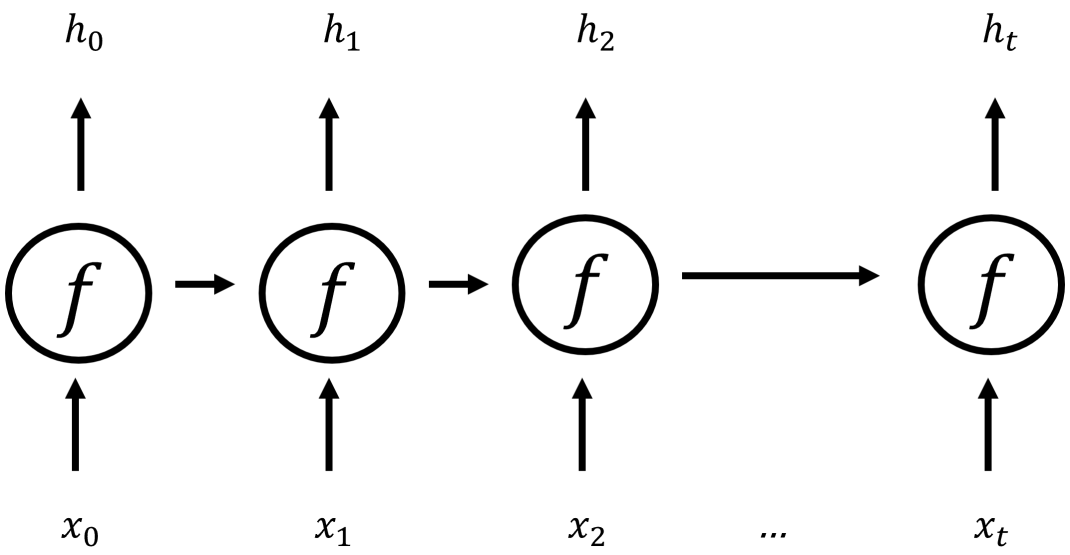
In the foreign exchange market. Ni et al. proposed a combination of convolutional neural

network and recurrent neural network to forcast daily foreign exchange rate of nine currency pairs during ten year's time [93]. Their method is a regression method, which outputs the daily close prices of the foreign currency, and they only compared with various neural network configurations but not any other model types. Rassetiadi and Suharjito used stock index valus and prices of commodities to assist the prediction of foreign exchange rates, they conluded that the value of FTSE and natural gas is the best combination to improve the prediction of EURUSD rates [94]. There are other literatures on prediction of FX rate with convolutional neural networks [95][96], as well as on news sentiment analysis in the FX market [97][98].

For other areas in the financial industry, convolutional neural networks also have many application potentials. They can be helpful to improve the efficiency and accuracy of fraud detection [99][100][101], improve the management of counterparty risk [102], and optimise portfolio allocation [103][104].

There is a particular structure of a convolutional neural network called the inception model. It was first introduced by Szegedy et al. in 2015 [105]. It has shown outstanding efficiency in improving model performance with relatively low computational costs. In a standard convolution structure, most convolution kernels, except those at the first layer, are applied to an abstract representation of the input image processed by previous layers. By incorporating inception modules, convolution kernels with various sizes are simultaneously used on the same input so that local interactions in different regions and scales can be captured and stacked together. Regarding financial time series analysis, the concept of inception is similar to taking simple moving averages at different window sizes.

Figure 2.10: A Simple Recurrent Layer



2.3.3 Recurrent Layers

Recurrent neuron is another category to define how neuron are arranged in the network. Many people prefer to use the terms 'Convolution Neural Network' and 'Recurrent Neuron Network'. However, we think they are not really appropriate names, neuron networks are rarely composed with only convolution or recurrent layers, they are usually combined with fully connected layers. So, we just call them convolution layers and recurrent layers (or neurons).

The concept of designing neuron network layer in a recursive manner was first proposed by Elman in 1990 [106]. He tried to use a network system to represent information that has temporal relationship in embedded. In Figure 2.10, we show how a classic recurrent layer is structured. We could see from the figure, that input information is a time series of X_0, X_1, X_2 until X_t. Every X value is given to a computing neuron, where it has two outputs. One output is given to h, which is called context neurons, to save the output for this time stamp.

And other output is given to the computing neuron at the next time stamp. In this way, a simple connection between two consecutive time stamps are established. When processing information for time t, information related to $t-1$ is also considered. Note, the indicator t is not necessary to represent time, it could represent any type of order. For example, the sequence of words in a sentence. If we choose to output all values of h, then this layer is generating a vector as the same length of input (for example sentence translation), or we could also just output the final value of h (for example sentence sentiment classification), this is depended on the specific task requirement.

A significant problem with this classic recurrent arrangement of neurons reveals when the model is trained. During training, similar to fully connected layer and convolutional layer mentioned above, we compare the model output value with the truth value, and compute the partial derivatives for updating the weights in the neural network.

$$\frac{\partial l_3}{\partial w_3} = \frac{\partial l_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial w_3} \tag{2.24}$$

Where $l_3 = f(y_3)$ and f is the loss function. $w3$ is the model weights associated to the third computing neuron. We also know that $s3$ is a combination of $x3$ and $s2$, and if we include every previous time stamp:

$$\frac{\partial l_3}{\partial w} = \frac{\partial l_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial w_3} + \frac{\partial l_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial w_2} + \frac{\partial l_3}{\partial y_3}\frac{\partial y_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial w_1} \tag{2.25}$$

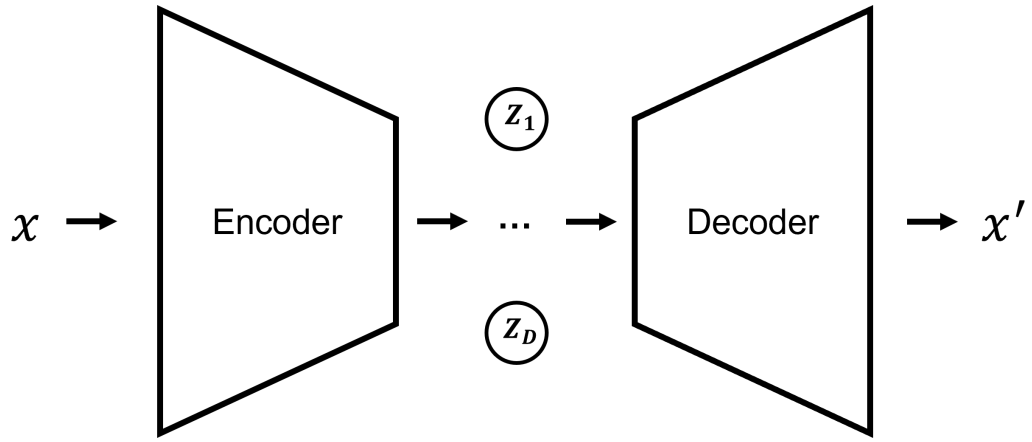
Here we only show an example at $t = 3$, if the recurrent chain become very long, then the calculation involves a long term of multiplying many partial derivatives. The multiplication would cause the gradients to be exceptionally large or small, and make it impossible to update the model weights. This is called the exploding and vanishing gradients problems

(EVGP) [107].

There are a couple of solutions to overcome the EVGP problem with classic recurrent neurons. A popular one is call long short-term memory (LSTM) neuron, proposed by Hochreiter and Schmidhuber in 1997 [108]. LSTM neuron has three gates that control what information to be taken by current activation (input gate), to pass to the next neuron (output gate) or to discard from the current state (forget gate). Instead of linearly connecting neuron at every time stamp, it is filtering out what past activations to connect to the neurons at a later time stamp. It is similar to break the connections for a fully connected layer, and this solves the EVGP problem to some extent. A Gated Recurrent Unit (GRU) is a new variant of LSTM, which fundamentally combines the forget and input gate into a single update gate [109], which is more suitable when the input sequence is short. Chung et al. reported that LSTM and GRU surpass traditional recurrent units with faster convergence and better final performance [110].

Recurrent neural network structure, especially LSTM and GRU, are widely applied in modern deep learning models for natural language processing tasks. Zhou et al. proposed a neural network model, which combines convolution layer and LSTM layer for sentence sentiment classification and question classification task [111]. There are also LSTM based neural network model for named entity recognition in text[112], as well as automatic language translation machine [109]. For applications in the financial market, LSTM is usually combined with some pre-processing steps to perform its tasks. For example, Zhang et al. used LSTM layers with empirical mode decomposition and principal component analysis as pre-processing steps to forecast closing price of stock index in the next trading day [113]. Zhang et al. implemented a LSTM layer after inception modules for classifying stock prices from limit order book data and stated that LSTM could reduce overfitting because the number

Figure 2.11: An Auto-encoder Neural Network



of parameters is largely reduced as opposed to a fully connected layer [89].

2.3.4 Auto-Encoder

We have introduced three ways (fully connected, convolution and recurrent) in which artificial neurons could be arranged at a layer level. These different types of layers could be mixed into a neural network to perform specific tasks.

If we look at a neural network at a higher level, what we have discussed so far is that a network takes some input information (matrix or vector) and outputs a label (classification) or a numerical value (regression). In this section, we will discuss a particular high-level structure of the network called the auto-encoder (or encoder-decoder) model. Usually, auto-encoder is only referred to the high-level arrangement of layers in a model, meaning each layer could be fully connected, convolution or recurrent.

The model structure of the auto-encoder model is shown in Figure 2.11. As we can see, it is combined with two separate neural networks called the encoder and decoder. Usually, these two networks have the same structure and are just reflections of each other, which

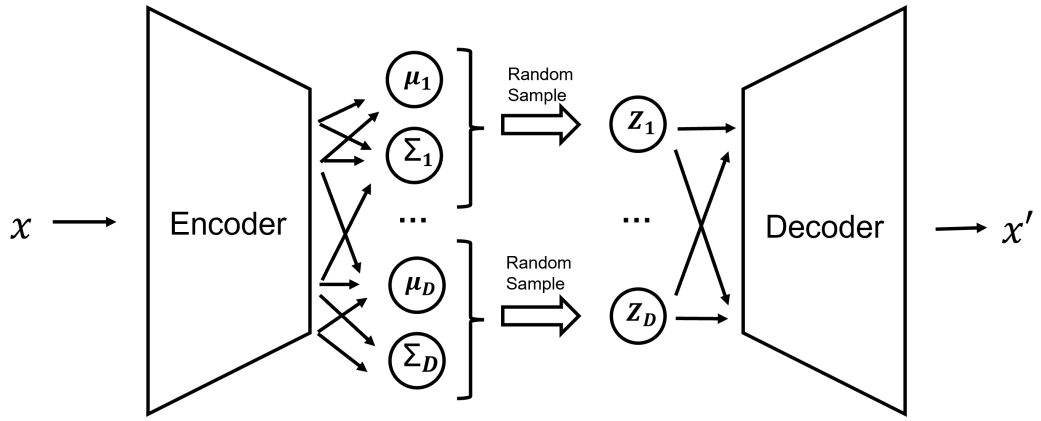
means the encoder transfers a high-dimensional input into a lower dimension. The decoder transfers the low-dimension representation back to the original high dimension. The input X and output X' are identical matrixes or vectors for an auto-encoder model, meaning the auto-encoder effectively approximates the identity function $f(X) = X$. When the difference between X and X' is small enough, the latent representation Z, the output of the encoder network (input to decoder network), is considered to have successfully stored the information about the input X in this latent space. Usually, the dimensionality of Z is much smaller than X (i.e., dimensionality reduction).

The dimensionality reduction functionality of the auto-encoder model has a good deal of practical usefulness, particularly in image and language processing. Zhang compared different auto-encoder neural networks with fully connected layers and convolutional layers for performing image compression and image de-nosing tasks and concluded that convolution layers are better candidates [114]. Chen et al. used a modified auto-encoder model to learn specific features of a text document with each hidden neuron and perform downstream tasks of documentation classification [115]. Liou et al. used an auto-encoder model with recurrent layers to encode polysemous words into a latent sentiment space, where the latent vectors represent different meanings of a word [116].

Variational Auto-Encoder (VAE) models are a popular extension of the autoencoder framework. It was first proposed by Kingma and Welling in 2013 [117] and later expanded in finer details [118]. Figure 2.12 shows a variational auto-encoder model structure.

Instead of producing latent encodings Z deterministically like the classic auto-encoder, the VAE randomly samples them from a normal distribution parameterised by the encoder neural network. This unique arrangement has one particular advantage. Because encoder and decoder networks are not directly linked, when the encoder takes the same input batch

Figure 2.12: A Variational Auto-encoder Neural Network



in training, the changes in Z are not only due to the updated parameters in the encoder but also the randomness caused by the sampling process. This forces the decoder network to learn a distribution of Z values instead of deterministic values, which results in a more powerful decoder for synthetic data generation. This also explains why a VAE model takes longer training time than a classic AE model on the same dataset, even with identical training parameters.

Bergeron et al. implemented variational auto-encoder models on the implied volatility surfaces from the foreign exchange market [15]. They encoded implied volatility surfaces into various numbers of latent dimensions (from 2 to 4) but did not clarify how many dimensions are optimal. This is because their method could not separate the features of a surface into each latent dimension so that every time a new dimension is added, the model gains a certain degree of accuracy. However, whether that dimension is encoding a meaningful feature or just market noise still needs to be answered. In addition to that, the encoding vectors obtained from their model are not helpful for a stock-specific generation due to the lack of interpretability.

2.3.5 Universal Approximation Theory

In previous sections, we introduced three different formulations for a single layer in a neural network. We also discussed some applications with the classic input-output structure and the typical auto-encoder neural network where input and output are identical. In this section, we will briefly explain why neural networks can handle many challenges across various fields. With a more profound understanding of the mechanism behind neural networks, we could better utilise the models to solve a wide range of problems or improve the efficiencies of existing solutions.

As mentioned in Section 2.3.1, the activation function of a single computing neuron has to perform some non-linear transformation from the input value to the output value. If we only use linear activation functions when connecting many neurons to construct the neuron network, the network would be only able to model linear relationships because the linear combination of linear functions is still a linear function, which is not practically useful for many problems in real life.

Many problems we are interested in can be considered to find a non-linear relationship between two entities. For example, an image classification problem is deemed to look for the relationship between an image matrix and its classification label vector. Stock price prediction problems can be seen as finding the relationship between historical stock prices and future price values or price movement directions. The language translation problem is to find the relationship between two sentences in different languages.

The universal approximation theorem states that a feedforward neural network with at least one hidden layer (with non-linear activation function) can appropriate any non-linear function from one finite-dimensional space to another with any desired non-zero amount of

error, as long as the network has enough hidden neuron [119]. Please note the theorem confirms the ability of a neural network to represent the non-linear relationship but does not guarantee the successfulness of learning the optimal parameters (weights) in the neural network. The neural network learning process may fail due to many reasons.

One obvious scenario is that the relationship, which needed to be identified, does not exist or does not dominate the training data. For example, when we use a neural network to predict asset prices in the financial market, it is usually best to train the model using data from recent history and predict the near future. However, suppose the time lag between the training and testing dataset is significant. In that case, the model performances on the test dataset are usually bad, even though it fits well for the training dataset. The explanation is that, in the financial market, the underlying market environment evolves as time passes, and the relationship between the input features (e.g. some typical technical indicators) and the output signals (price movements) learnt from the past market environment is no longer valid in the future market environment. In many machine learning textbooks, this is also called 'óverfitting', which is the concept of a model that focuses too much on the relationship that does not exist in the testing dataset.

Another possible failure is caused by inappropriate model structure or training parameters. In previous sections, we introduced three neuron structure methods (fully connected, convolution and recurrent). In the early years of developing neural networks for image classification problems, many models were based on fully connected neurons only, and they could achieve some degree of satisfaction. However, after convolutional neural networks were proposed, people found they could achieve much higher accuracy and cost much less training efforts. Moreover, the convolution operation is naturally more suitable for the image matrix because it focuses on small windows on the image and shares the learnt parameters

across the matrix. Similarly, recurrent neural networks, especially LSTM neural networks, are more suitable for information with temporal relationships, such as sentences and time series. In theory, we could use fully connected layers to process image and time series input, but it is not efficient, and more importantly, it is usually challenging to find the optimal hyperparameters for training the network successfully.

Because of the universal approximation ability, one superiority of neural networks over other machine learning solutions is the model learned for solving one problem can be applied to a different but related problem, also known as transfer learning. This is particularly beneficial when sufficient well-annotated data cannot be acquired easily. It is conventionally assumed that test and train datasets are independent and identically distributed. However, from the transfer learning aspect, this condition is no longer required [120]. Network-based transfer learning reuses for a task the neural network model (including its model structure and learned parameters) trained for another task. We know that a neural network model trained on static images could be transferred for recognising actions and objects from video clips [121]. Features from different languages could also be shared in the hidden layers of a neural network structure, and a multilingual transfer model surpasses monolingual models with a clear margin [122]. Sirignano and Cont trained universal neural network models with over 1,000 US stock and uncovered that a universal price formation mechanism exists in stock limit order book [123]. Their universal model outperforms stock-specific models for the out-of-sample test and even for stocks, not in the training sample. The existence of universal features, which improve the prediction power of neural networks, has also been observed in the context of cryptocurrency markets [124].

In summary, the universal approximation theorem tells us that neural network models are universal approximators of non-linear relationships. However, we need to optimise a neural

network model to learn the relationship from our training dataset and apply it to the testing dataset. The challenge is to find the optimal training hyperparameters and the most appropriate model structure for the specific task. Neural network models are data-driven solutions, meaning data quality is the key to success. We must ensure the non-linear relationship is consistent in the training and testing datasets. This is usually guaranteed for computer vision tasks rather than financial time series problems. The universal approximation ability brings the advantage of transfer learning for the neural network model, which could be helpful if training data in the target domain is insufficient.

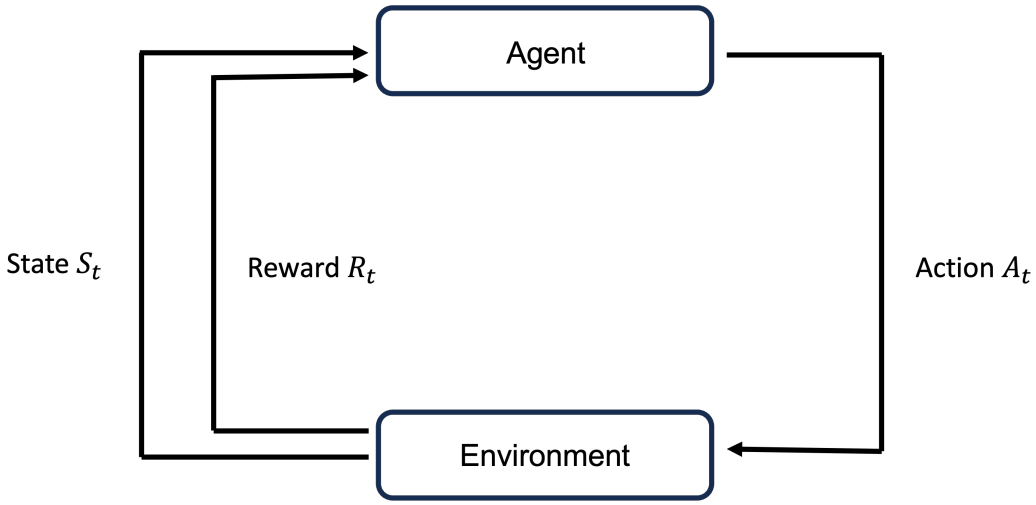
2.3.6 Reinforcement Learning

Reinforcement learning (RL) is about an agent interacting with the environment, learning an optimal policy, by trial and error, for sequential decision-making problems [125]. The elements of an RL system are the agent and the agent's current situation (state), the environment in which the agent interacts, the reward that the agent receives from the environment and the policy to map the agent's action to states. Their relationship could be summarized in Figure 2.13.

In Figure 2.13, we have an agent that takes action A_t at time t and receives reward R_t from the environment. The agent's state changes to S_t due to the action and the reward from the environment. The agent also has a set of policies to map states S to actions A. Therefore, at time $t + 1$, the agent will take a new action $A_{t+}1$ based on S_t and its policies, and the learning process will repeat itself until the termination state has been reached. The goal of an RF agent is to find the optimal policies that maximize the cumulative reward [125].

In recent years, using deep learning or deep neural networks to solve reinforcement learning problems has prevailed. Neural networks can process large amounts of high-

Figure 2.13: A Reinforcement Learning System



dimensional data, optimised from end to end, with little domain knowledge. Deep neural networks model the relationship between the actions and rewards in the reinforcement learning pipeline. The deep Q-network is an example where a deep neural network agent with the input of only pixels and game scores can surpass the performance of all previous algorithms across a set of 49 computer games. More importantly, the network is trained with the same network architecture and hyperparameters for all the games [126].

There are also many applications of reinforcement learning for financial problems. It could be used to learn a portfolio allocation agent which makes investment decisions and try to maximise the risk-adjusted return of an investment portfolio [127]. There are also reinforcement learning solutions for trading S&P500 futures contracts, and the DDQN model can adjust its policy to different market circumstances and achieve higher trading profits than the benchmark long-and-hold strategy [128]. The deep hedging pipeline, which will be discussed in the next section, is also a reinforcement learning solution for hedging the stock option contracts, and it optimises a neural network model for generating hedging strategies (policies) when the underlying stock price (environment) changes from one trading day to

the next [17].

2.3.7 Deep Hedging

From previous sections, we know that the strength of a neural network is to approximate non-linear relationships. For example, one well-studied relationship in the financial markets is the prices of the underlying assets and their option price. The mathematical-based Black-Scholes-Merton method has been introduced in Section 2.2.5. We could also use neural networks to capture this relationship.

The first article discussed using neural network models for pricing, and hedging financial derivatives could be dated back to 1994. Hutchinson et al. simulated many artificial option prices and then trained neural network models to replicate the option prices. It effectively tests the network's ability to approximate the non-linear Black-Scholes-Merton formula [129]. They found that the neural network's performance is remarkably good, and they argued that neural networks could be valuable substitutes when conventional parametric methods fail. However, they only experimented with a single asset S&P 500 futures options from 1987 to 1991. They also did not consider market trading costs when simulating the option prices. Therefore their work is far from real-life applications.

Deep Hedging is the recent state-of-art framework utilising a neural network for trading financial derivatives [17]. Let us recall Equation 2.7, while we want to replicate the option contract using a simple portfolio with only accumulated cash e^{rt} and the underlying stock S_t.

$$P_t = x_t e^{rt} + y_t S_t \tag{2.26}$$

$$y_t = \frac{\partial C_t}{\partial S_t} \tag{2.27}$$

The default version of deep hedging uses a neural network model to find the value of y_t when we observe S_t in the market. From the classic Black-Scholes-Merton, y_t is the partial derivative calculated from the equation above, but we know it only works under idealised conditions of (i) underlying stock prices follow geometric Brownian motion movements, (ii) trading costs and other trading constraints do not exist, and (iii) continuous trading and portfolio rebalancing is possible. When these perfect conditions do not apply, the replication portfolio P_t will incur a loss at the termination of the option contract, which is the basis of the price for engaging the option contract.

The neural network structure for Deep Hedging is not very complex. It is not very "deep" compared with other purposed networks for financial time series classification or prediction. There are only two fully connected layers, with fifteen neurons each. There is also an input layer, which takes the value of S_t and an output layer, which output the value of y_t according to the equations above.

One fundamental advantage of neural networks is the capability to utilise various loss functions for different purposes. For example, regarding financial derivatives, loss functions could represent someone's unique risk appetite or utility function. In other words, how they evaluate these cashflows according to their specific situations. Therefore, the loss function implemented by Deep Hedging differs from those adopted by image or language processing networks, such as the MAE or MSE functions we introduced in Section 2.3.1. The authors of Deep Hedging proposed two different types of functions. The first one is entropy risk measures:

$$\rho(X) = \frac{1}{\lambda} log\, \mathbb{E}(e^{-\lambda X}) \tag{2.28}$$

λ is a constant number larger than zero, considered some risk appetite. As λ gets larger, an investor is willing to take more risks. The user of Deep Hedging could set their value of λ, which reflects specific risk tolerance, shareholders' requirements and regulatory constraints.

Another loss function is very commonly seen in regulation documents, which is expected shortfall:

$$\rho(X) = \frac{1}{1-\alpha} \int_0^{1-\alpha} VaR_\gamma(X) d\gamma \tag{2.29}$$

The logic behind this is that one could first decide a probability associated with standard scenarios, say 90%. Then calculate the expected losses if the remaining 10% extreme events happened, and consider this value a baseline for risk preferences. For example, if the value of α is set to 99%, then the calculation is based on more extreme events than α is 90% so that expected losses are larger. In their experiments, they simulated 10^6 different possible trajectories of the underlying assets, using the model to hedge each of them and calculate the mean of the worst 1% termination losses. This could be the basis for determining the option premium. In the formula, VaR is the Value at Risk.

By linking the loss functions with the risk preferences, Deep Hedging is more suitable for financial market participants. The regulation requirements are also easier to be met. Deep Hedging started a new topic to find the non-linear relationship between the prices of the underlying assets and their derivative instruments with neural networks. This way, some unrealistic assumptions from the Black-Scholes-Merton method could be dropped. In addition, including risk preferences in the neural network training makes the algorithm more applicable and explainable.

However, many aspects could be improved with the default version of Deep Hedging.

Instead of trading daily, there could be a mechanism to automatically make trading decisions when the underlying price reveals specific patterns. This is what we did in our second research project, which will be discussed in Chapter 4

Deep Hedging is the most comprehensive study on developing hedging strategies from neural network models. There are also a couple of similar methods. Ruf and Wang also used a neural network to generate hedging strategies for stock options, but their approach does not use the price of the underlying asset directly as the network input, and they use hedging error as the loss function instead of a risk measure [130]. There is another similar model where the objective of the neural network is to optimise profit and loss of hedging at the end of the contract, and the rebalancing interval is fixed at several days or weeks [131].

2.4 Other Machine Learning Methods

This thesis is focused on neural network models and their applications on various topics on trading and hedging. First, however, we must compare our novel models with conventional methods to demonstrate superiority. Therefore, we introduce several classical machine learning methods in this section. Some of these methods are compared with our proposed algorithm, and some work as supplement models to improve our neural network based pipeline.

Logistic Regression (LR) is the most straightforward classification algorithm in machine learning. It uses a logistic function to map the input features into probability distributions corresponding to each classification category. The results are pretty fast to obtain with the maximum likelihood method, but LR assumes that the conditional distribution of a data label given the input features is a Bernoulli distribution [69].

K-Nearest neighbour (KNN) is a popular and intuitive classification algorithm. During the testing phase, it calculates the Euclidean distances between a test sample and the whole training dataset provided. Then, the predicted label of the test data is decided from a certain amount of points on the feature space that is closest to it [132]. The idea is straightforward, but an obvious drawback is the computational cost when the feature space has many dimensions. Its success is highly dependent on the selection of features [133].

Random Forest (RF) is an ensemble method used for classification or regression. The classification output is the class label produced by most ensemble decision trees. A decision tree is a non-parametric method that recursively partitions the feature space of the dataset. Its objective is to find a set of decision rules which maximise some particular scoring criteria [134]. The flexibility to choose an arbitrary tree size makes it a good candidate for data mining and feature engineering but also raises the problem of overfitting.

A Support Vector Classifier (SVC) is a natural choice for a classification problem. This method is similar to LR but outputs class density instead of probabilities [69]. It separates the data points in the feature space with a hyperplane. Support vectors are those data points closest to the hyperplane and influence its position. Unfortunately, SVC can hardly overcome the high computational cost for large datasets. Nevertheless, it was the starting point for modern deep learning when Hinton used a neural network to outperform SVC on the MNIST dataset [135].

The Linear Discriminant Analysis (LDA) method is commonly used for dimensionality reduction. This method projects data points onto a line that maximises the ratio of between-class variance to within-class variance and thereby guarantees maximal separability. The drawback is that LDA assumes that different classes are linearly separable in the projected space, which is untrue for most financial-related problems.

Chapter 3

Technical Analysis Neural Network

For this study, we propose a novel pipeline to predict price movements on noisy high-frequency data from the foreign exchange (FX) market. Our solution is inspired by recent advances in the adoption of AI to forecast stock price movements, which are (i) using technical indicators to encode financial time series into image-like matrices, (ii) forecasting future price changes by classifying these matrices using a neural network model, (iii) leveraging across-asset features to achieve improved prediction accuracy. We combine these three approaches and explore their effects on the high-frequency FX market, wherein the unevenly sampled tick data is noisy, and the data depth is limited to only one level. Ultimately, we wonder if these approaches continue to work in a quote-driven market.

We want to demonstrate that a neural network model is capable of handling the high-frequency tick data from the FX market. Convolution layers are utilised to extract features across five technical indicators, whereas inception modules and Gated Recurrent Units (GRU) capture useful spatial structures. We assess the model by testing its out-of-sample classifications on future price movements. We show how our model outperforms several

well-known machine learning methods. We also study how the user's expectation towards higher returns would change the balance of the classification categories, and therefore influence the performances of the model. Finally, we found that training a universal model with all FX pairs could further improve the classification results.

3.1 Introduction

The foreign exchange (FX) market is one of the most influential markets for the financial industry, due to its around-the-clock trading, massive traded volume, and diversity of participants. Those typical characteristics have stimulated the interests of many investors, policymakers, and academics [20]. However, FX rate movements are widely considered difficult to predict [28] [136]. In a much-cited paper, Meese et al. concluded that structural models could not surpass a random walk model for out-of-sample prediction on FX data [137]. The phrase 'exchange rate disconnect' is used to describe the lack of connection exchange rate and other macroeconomic variable, and many people using various business and economic model to formulate it [138][139][140].

Conventionally, most datasets and studies published on the FX market are based on low-frequency, regularly spaced data points. There are emerging concerns from both academia and industry on analysing the market at a different scale. "High-frequency data" or "Tick data" for the FX market are collected at much smaller and irregular intervals and according to Dacorogna et al., it should be the primary object of research for those who are interested in understanding financial markets [141]. Many people have been using high-frequency data to performance various analysis in the foreign exchange market. Glattfelder et al. discovered 12 empirical scaling laws in FX market from high-frequency data, and these

are important contribution to the theoretical explanations of the market mechanisms [142]. Li et al. used high-frequency data to analysis relative volatility in the FX market [143]. High-frequency trading is considered an evolution of trading methodology. High-frequency traders implement sophisticated technical analysis methods combined with powerful computational resources to seize the fleeting opportunities in the market [144]. Because of the riasing of high-frequency trading, people have also found that liquidty of the FX spot market has been influenced by high-frequency trading behaviour [145].

In recent years, many computational intelligent techniques originated from various research fields. Several neural network models achieved outstanding performances and started new eras for computer vision and natural language processing [105] [146]. In the financial area, there are also some encouraging achievements in the context of stock prices analysis.

Our work is motivated by three complementary approaches in the literature. Sezer and Ozbayoglu converted technical indicators into image-like matrices and used a CNN for classifying stock prices [5]. This work shows an interesting take in leveraging the success of AI in different application areas and importing them to finance. Sirignano and Cont proved that there exist universal features among stock prices, and that models trained with universal price information could beat asset-specific ones [123]. This might be due to the market microstructure of the limit order books collecting the bids and asks for stocks and an underlying common dynamics in the price formation across assets. Zhang et al. created DeepLOB, which is the state-of-art architecture, based on inception networks, for predicting short-term stock price movements on the FI-2010 benchmark dataset of stock limit order books [89]. An important observation here is that DeepLOB relies on quite expensive and rarely available limit order book data of depth ten (i.e., ten best bids and asks).

For this study, we investigate whether the combination of the approaches in [5], [123]

and [89] would warrant similar performances in FX markets. The main motivation behind our study is twofold. Firstly, as argued above, currencies are an important and fundamental class of assets that people trade. Secondly, FX markets are not order-driven but quote-driven, which means that some of the market fundamentals behind the success of previous research are not present. The question is to what extent having very little and shallow data will deteriorate the performances of conveniently encoded technical indicators, inception networks and universal models. We propose a novel pipeline to study the question above. More specifically, we convert high-frequency FX information into image-like matrices of technical indicators, utilising inception models and gated recurrent units (GRU) to classify short-term FX rate changes into three classes: up, stationary and down. We name our model **Technical Analysis Neural Network** (TANN). We show that TANN can sustain reasonable performances with very limited “level 1" (i.e., bid and ask from the market maker) data. In particular, we compare our model with a number of popular machine learning classification methods, including K-Nearest Neighbor (KNN), Linear Discriminant Analysis (LDA), Logistic Regression (LR), Random Forest (RF) and support vector classifier (SVC). These models are all implemented in Python with scikit-learn package version 0.24.0 and with default model parameter settings [147]. Our model outperforms the best machine learning method we have experimented, which is LDA, by roughly 5% of classification accuracy. As the input information is limited, our TANN runs pretty quickly (including the training phase) – and not for much longer than those machine learning algorithms – on relatively cheap hardware, available to the majority of financial institutions. We also confirmed that universal features exist in high-frequency FX data by training a universal TANN model with ten different currency pairs and comparing test performances of the universal model with currency-specific models.

The remainder of this chapter is organised as follows. Section 3.2 describes the data

Table 3.1: Minute Data for EURUSD

Time	Open	High	Low	Close
00:00	1.14168	1.14187	1.14168	1.14184
00:01	1.14184	1.14184	1.14183	1.14183
00:02	1.14183	1.14202	1.14177	1.14180
00:03	1.14179	1.14179	1.14168	1.14172
00:04	1.14179	1.14179	1.14172	1.14172

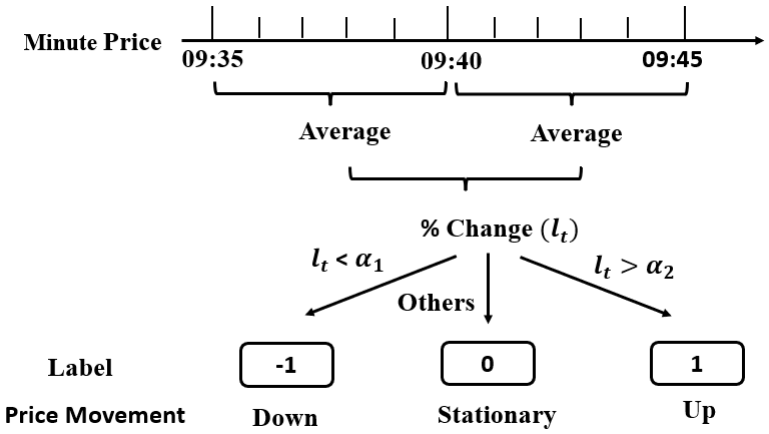


Figure 3.1: Labelling Process

used and the important data pre-processing steps for our model. Section 3.3 discusses the model structure and the training of the model. Section 3.4 discusses the performances of our models comparing with other machine methods and with different threshold settings. Section 3.5 concludes this chapter with some final observations.

Table 3.2: Statistics of Time Intervals Between Consecutive Tick Points (Seconds)

	(a) July 2017			(b) July 2018		
FX	Min	Mean	Max	Min	Mean	Max
AUDJPY	0.093	0.500	377.250	0.160	0.500	285.747
EURSEK	0.076	0.267	999.250	0.186	0.250	1078.750
EURTRY	0.047	0.514	980.500	0.126	0.500	299.000
EURUSD	0.076	0.267	167.000	0.170	0.266	321.497
GBPCAD	0.060	0.500	726.500	0.170	0.500	111.000
GBPJPY	0.047	0.250	443.000	0.126	0.250	263.500
USDCHF	0.076	0.280	274.496	0.126	0.500	656.750
USDDKK	0.060	0.267	589.243	0.126	0.267	951.247
USDJPY	0.047	0.250	181.750	0.160	0.263	210.250
USDNOK	0.076	0.484	323.746	0.186	0.500	186.750

3.2 Data

We want to re-emphasize that the global FX market is not subject to formal regulations, where there is no single institution recording all the transactions. There is the interbank market on the top level of FX transactions. Therefore, there is no comprehensive data source like limit order books from stock markets. One data vendor may have its own data source, which differs from another vendor.

Our datasets are from histdata.com. We downloaded ten FX pairs for a period of five consecutive years (2014 to 2018). Those historical prices are collected in two different formats, namely tick data and minute data.

In the minute dataset, prices are presented at a regular interval of one minute. For the convenience of explanation, we identify price information for one minute as a data point, i.e., one row in the data table. For every data point, we have four dimensions for the minute dataset. They are Bid Open, Bid High, Bid Low, and Bid Close. Table 3.1 shows a subset of minute data we have for EURUSD on the 3rd of July 2017.

In the tick dataset, each data point consists of two features: bid price and ask price. For each FX pair, there are roughly 164M ticks in our modelling period (2014 to 2018). Time intervals between two consecutive points are small and irregular. Table 3.2 summarises time intervals between consecutive points in tick dataset for two particular months of July 2017 and July 2018. It shows that the mean of time intervals between two tick prices are less than one second, however sometimes the interval become even larger than a couple of minutes. The arrival of ticks is very irregular and primarily depend on the trading activities in the market.

Instead of putting raw tick prices directly into the model, we calculate five technical indic-

ators at eleven different window sizes, and format them into an image-like matrix, where each row represents one window size and each column represents a technical indicator. There are three considerations for this. First, for high-frequency FX data, there are only two dimensions (bid and ask) for a single data point. There is no information regarding the depth of market, i.e., what are next best prices people want to buy/sell the asset. The fundamental benefit we receive from high-frequency data is high density of historical information. Abstracting historical information at various windows sizes is a simple solution to take advantage of this. The second reason is that tick data are often very noisy, and calculating technical indicators is the common way to filter financial time series. Finally, the third consideration is from the model perspective. In theory, a neural network model could ultimately remove noise and abstract meaningful features from input data. However, in practice, if we were to simply feed the network with an exceptionally long vector or a large image, a simple model with a few layers would struggle to converge and become non-trainable. It is possible to increase the complexity of the model as well as its training process to make it works with raw data, but we choose to pre-process the data and keep the model simple.

Differently from image classification or speech tagging, there are no natural labels for financial time series data. Financial time series are highly stochastic. If price movements are calculated by simply comparing current price p_t and next price p_{t+1}, the resulting labels will be very noisy. Therefore, we adopt the method from DeepLOB model [89], and denote m_- as mean of previous k prices, and m_+ as mean of next k prices, that is,

$$m_-(t) = \frac{1}{k}\sum_{i=1}^{k} p_{t-i} \tag{3.1}$$

$$m_+(t) = \frac{1}{k}\sum_{i=1}^{k} p_{t+i} \tag{3.2}$$

where k is the prediction horizon, please note that p_t is not included in either the calculations of $m_-(t)$ or $m_+(t)$. Our labels are calculated based on bid close prices on minute dataset, and k takes value of 5, 10 or 15 minutes.

We compare the percentage change l_t of past and future average prices to decide movements, and l_t is defined as follows

$$l_t = \frac{m_+(t) - m_-(t)}{m_-(t)}. \tag{3.3}$$

Lastly, l_t is compared to a pre-defined threshold α for deciding a label for each data point. If $l_t > \alpha$, we assign it label up (1), and if $l_t < -\alpha$, we label it down (-1). For anything else, we consider it stationary (label 0). The labelling process is illustrated in Figure 3.1.

Since the price ranges for each FX pair are so different, the thresholds α are defined ad hoc for each FX pair. The thresholds we used are shown in Table 3.3. For a single FX pair, the split is close to 33% for each class based on the minute data for the whole period. From a financial perspective, the value of the threshold α can be interpreted as a proxy for the investor's risk preference. For a fixed FX pair, the higher the value of α the larger (smaller, respectively) the price increase (decrease, respectively) needs to be for the asset to be considered up (down, respectively). In other words, the higher α the more conservative the trading signals will be for a more risk-averse investor who is expecting higher returns for taking risks. We will compare our model's performances with fixed (as in Table 3.3) and dynamic thresholds in Section 3.4. The fixed thresholds are obtained by calculating the distribution of all returns for a particular FX symbol (e.g. AUDJPY) with a specific prediction horizon (e.g. $k = 5$) and selecting the 33rd percentile and 66th percentile as α_1 and α_2 respectively.

Table 3.3: Values of α for each FX pair ($\times 10^{-4}$)

FX	$k=5$		$k=10$		$k=15$	
	α_1	α_2	α_1	α_2	α_1	α_2
AUDJPY	-1.13	1.18	-1.46	1.55	-1.71	1.84
EURSEK	-0.38	0.41	-0.55	0.59	-0.68	0.74
EURTRY	-1.07	1.11	-1.45	1.50	-1.76	1.80
EURUSD	-0.71	0.70	-0.93	0.92	-1.10	1.09
GBPCAD	-0.84	0.86	-1.09	1.11	-1.27	1.30
GBPJPY	-1.00	1.03	-1.29	1.34	-1.51	1.59
USDCHF	-0.69	0.72	-0.91	0.97	-1.09	1.16
USDDKK	-0.70	0.71	-0.92	0.93	-1.10	1.10
USDJPY	-0.77	0.80	-1.01	1.09	-1.21	1.31
USDNOK	-0.92	0.95	-1.21	1.26	-1.45	1.51

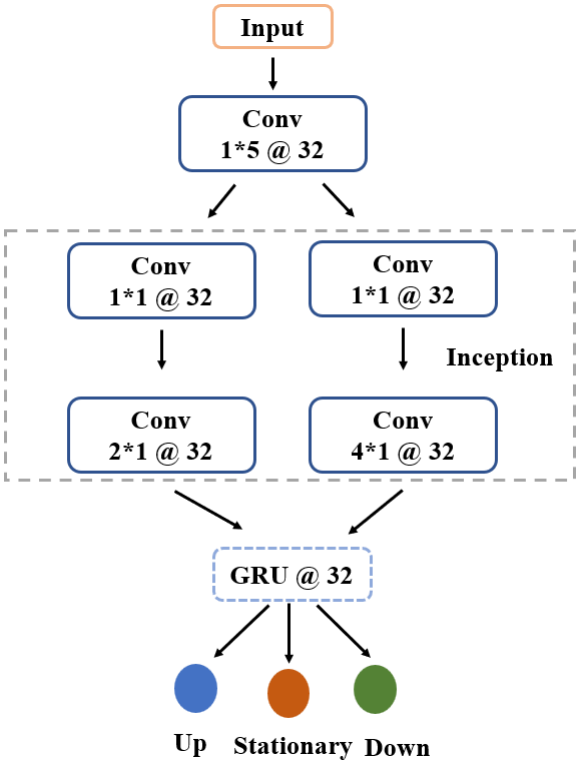


Figure 3.2: Overall Structure of TANN

Table 3.4: Matrix for Bid Channel

AUDJPY, at 08:01 on 10th August 2017

History	RSI	WMA	SMA	CMO	ROC
60	42.105	86.667	86.666	-15.790	-0.024
80	44.865	86.657	86.660	-10.270	-0.022
100	42.326	86.626	86.657	-15.349	-0.038
120	40.784	86.647	86.654	-18.431	-0.054
140	41.489	86.641	86.649	-17.021	-0.055
160	40.823	86.635	86.645	-18.354	-0.067
180	39.560	86.631	86.642	-20.879	-0.088
200	42.132	86.627	86.639	-15.736	-0.072
220	41.801	86.624	86.636	-16.397	-0.082
240	42.735	86.621	86.633	-14.530	-0.079
260	43.307	86.619	86.631	-13.386	-0.079

3.3 Model

An overview of our Technical Analysis Neural Network (TANN) can be found in Figure 3.2. Our architecture consists of three building blocks: a convolution layer, inception modules and a GRU layer.

The inputs to our model are image-like matrices. Each image has two channels, bid channel and ask channel, which are matrices derived from bid prices and ask prices of tick dataset, respectively. These two channels together represent the information needed for classifying the price movement for a particular minute. Within each row of the matrix, there are five technical indicators[1] calculated from a fixed length of ticks before this minute. There are eleven rows for one matrix. An example is shown in Table 3.4.

The selection of indicators is inspired by [5]. Sezer et al. used fifteen different technical indicators for processing stock prices. However, unlike the stock prices, we do not have volume, high and low price features in the FX dataset. Therefore, five (williams %R, commodity channel index (CCI), chaikin money flow indicator (CMFI), directional movement indicator (DMI) and parabolic SAR) out of the fifteen indicators are not applicable. Among the remaining ten indicators, we rank them by the mathematical complexity of calculating them. For example, the percentage price oscillator (PPO) and triple exponential moving average (TEMA) are the most complex because they are derived from the exponential moving average (EMA). EMA is more complicated than simple moving average (SMA) and weighted moving average (WMA). We want to reduce the number of indicators by comparing the classification performances on the July 2017 data with and without the most complex indicator.

[1]The five technical indicators are: Relative Strength Index (RSI), Weighted Moving Average (WMA), Simple Moving Average (SMA), Chande Momentum Oscillator (CMO) and Rate of Change (ROC).

If the model classification results are not reduced by removing one indicator, we think the removed indicator is redundant. We ended up with the five indicators mentioned above.

The first layer in the TANN is a classic convolution layer with 32 filters of size 1*5. We set strides to be 1*5 as well, so that they scan each row of the input matrix. It is equivalent to taking a weighted average of all the five indicators for a particular length of historical prices.

Then we have two inception modules. Each inception module consists of two classic convolution layers. First convolution layers are both 1*1 convolution, and are followed by 2*1 and 4*1 convolution. The 1*1 convolution could transform input into low dimensional representations, according to the Network-in-Network approach [148]. The following layers are effectively taking convolution operations at the time horizon for capturing temporal relationships.

The outputs of two inception modules are concatenated and reshaped to form a short vector of length eleven but with sixty-four dimensions. Usually people incorporate a dense layer before the output layer for classifying the features abstracted from previous layers. However, a dense layer assumes all input are independent, which is clearly not true for our problem. As discussed in Chapter 2, we choose a GRU layer for modelling the temporal dynamics of the time series input. It has much smaller number of parameters than a dense layer, which alleviates the problem of overfitting.

The output layer consists of three neurons with softmax activation function to produce the probability of labelling the input image-like matrices with one of the categories. All other layers implement with sigmoid activation function.

The design of model structure is inspired by the DeepLOB model [89]. The size of one input matrix for the DeepLOB is 100 rows with 40 columns, and each row consists of ten price-volume pairs. Therefore the filter size is selected to be 1*2, which every price figure

is multipied by the same parameter on the filter, and so does the volume figure. For our problem, every indicator present different characteristic of the market, it is not sensible to share parameter among them, so the filter size for the convolution layer is selected to be 1*5. The filter size for the inception modules is similar to the DeepLOB, and it is designed to take small steps at the temporal dimension of the input matrix.

For training of the model, we use the Adaptive Moment Estimation (ADAM) method and the optimal learning rate is set to 0.005 by grid search [72]. The search is from large values (i.e. 0.1), and if the training loss could not be reduced because the learning rate is too high, then the rate is reduced by multiplying 0.1. We found the training loss starts reducing with learning rate of 0.001. Then we try learning rate of 0.005 and it also works. With rate 0.005, the training is faster with 0.001. The loss function is categorical cross-entropy loss. We train with mini batches of size 8. There are literatures to suggest that large batch sizes tend to lead the training into a deep local minimum of the loss function, whilst small batch sizes consistently converge to a flat minimum and allow the model to generalise better [149].

Our TANN could be trained on the free Google Colaboratory platform with a single NVIDIA K80 GPU. The time needed for training 80 epochs of one currency-specific model with prices of a single trading day, i.e., 1440 images, is approximately one to two minutes. However, as we are training an extensive dataset with more than 37K neural network models, a single GPU is not feasible. Therefore, our results are processed from a cluster of 320 virtual CPUs spread evenly on ten cloud-based virtual machines, which gives us the advantage of training and testing multiple models in parallel. For the single model, the training time on the CPU is longer than a GPU, but parallel computing reduces overall running time dramatically. We also utilise other could-based infrastructures such as network-attached storage (NAS) and NoSql database for storing and processing our time series dataset and model evaluation

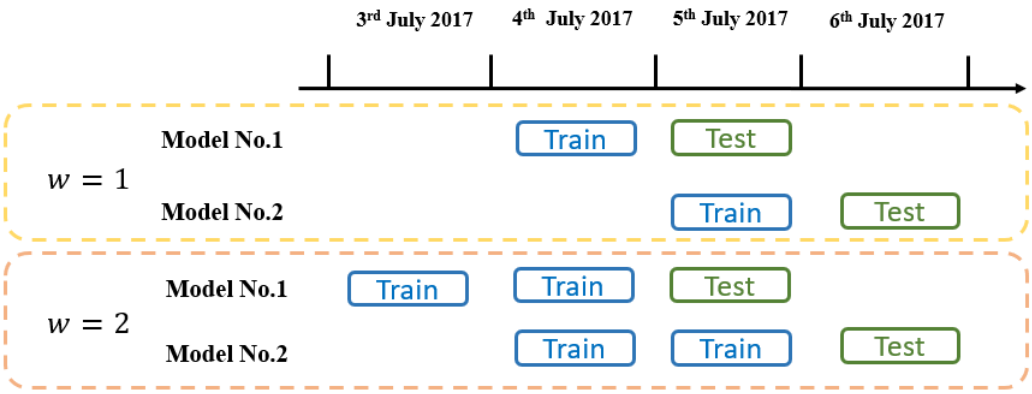


Figure 3.3: Moving windows training and testing

results.

3.4 Experimental Results

Our models are trained with a sliding window approach. The window size w represents the number of past days used to train a model for a target date. Then we calculate the classification accuracy from the prices within the target date. This approach is illustrated in Figure 3.3.

3.4.1 Baseline Results

We first take w = 1 so that the model is trained with prices from one trading day and tested on the next. We use the same method to train other machine learning algorithms (KN, LDA, LR, RF, SVC). Therefore, we end up with roughly 1265 models for each FX pair in the five years for every algorithm. The average classification accuracy is summarized in Tables 3.5, 3.6 and 3.7.

It is observed that KNN, RF, and SVC methods are only slightly better than random

Table 3.5: Average classification accuracy from 2014-01-01 to 2018-12-31 with k = 15

FX	KN	LDA	LR	RF	SVC	TANN	Average
AUDJPY	36.18%	42.77%	50.30%	37.04%	35.72%	47.47%	41.58%
EURSEK	35.98%	40.07%	36.20%	36.83%	35.68%	47.23%	38.67%
EURTRY	35.84%	39.85%	40.74%	36.59%	35.38%	45.70%	39.02%
EURUSD	36.13%	42.45%	42.35%	37.08%	35.50%	47.56%	40.18%
GBPCAD	36.45%	42.47%	36.61%	37.53%	36.30%	47.90%	39.54%
GBPJPY	36.00%	41.83%	42.70%	36.75%	35.25%	46.20%	39.79%
USDCHF	36.57%	42.98%	38.46%	37.55%	36.50%	48.72%	40.13%
USDDKK	36.13%	42.21%	42.69%	37.03%	35.56%	47.11%	40.12%
USDJPY	36.17%	43.13%	43.85%	37.02%	36.09%	47.57%	40.64%
USDNOK	36.20%	41.30%	40.05%	37.02%	35.87%	46.91%	39.56%
Average	36.17%	41.91%	41.40%	37.04%	35.78%	47.24%	39.92%

Table 3.6: Average classification accuracy from 2014-01-01 to 2018-12-31 with k = 10

FX	KN	LDA	LR	RF	SVC	TANN	Average
AUDJPY	36.55%	43.58%	49.76%	37.41%	36.03%	48.06%	41.90%
EURSEK	35.98%	39.94%	36.60%	36.84%	35.76%	47.32%	38.74%
EURTRY	35.68%	39.43%	39.19%	36.37%	35.13%	45.11%	38.49%
EURUSD	36.43%	42.99%	41.81%	37.41%	35.76%	48.01%	40.40%
GBPCAD	36.49%	42.55%	36.40%	37.81%	36.25%	47.83%	39.56%
GBPJPY	36.34%	42.66%	43.28%	37.20%	35.49%	47.06%	40.34%
USDCHF	36.39%	43.01%	39.72%	37.56%	36.36%	48.60%	40.27%
USDDKK	36.26%	42.47%	43.12%	37.19%	35.66%	47.28%	40.33%
USDJPY	36.36%	43.90%	45.10%	37.34%	36.10%	48.27%	41.18%
USDNOK	36.17%	41.53%	38.27%	37.15%	35.80%	47.03%	39.33%
Average	36.27%	42.21%	41.32%	37.23%	35.83%	47.46%	40.05%

Table 3.7: Average classification accuracy from 2014-01-01 to 2018-12-31 with k = 5

FX	KN	LDA	LR	RF	SVC	TANN	Average
AUDJPY	35.62%	41.71%	44.72%	36.48%	35.08%	45.82%	39.91%
EURSEK	35.27%	38.47%	37.08%	36.09%	34.91%	45.24%	37.84%
EURTRY	34.73%	37.75%	38.46%	35.41%	34.16%	42.37%	37.15%
EURUSD	35.62%	41.50%	40.73%	36.52%	35.12%	45.75%	39.21%
GBPCAD	35.41%	40.65%	34.44%	36.33%	35.22%	45.16%	37.87%
GBPJPY	35.70%	41.80%	39.51%	36.60%	35.04%	45.37%	39.00%
USDCHF	35.34%	40.98%	40.02%	36.17%	35.10%	45.76%	38.90%
USDDKK	35.39%	40.83%	41.60%	36.17%	34.89%	44.85%	38.96%
USDJPY	35.55%	42.34%	42.42%	36.49%	35.30%	46.03%	39.69%
USDNOK	35.26%	39.93%	36.80%	36.18%	34.98%	44.77%	37.99%
Average	35.39%	40.60%	39.58%	36.24%	34.98%	45.11%	38.65%

guess (33%), since as mentioned above our thresholds α have been defined to have a uniform distribution in three classes. LR shows very volatile performances, and it is slightly better than our TANN model in rare cases. However, we know that LR is just considering the relationship between input features and output class labels as a simple logistic function, which is very sensitive to the general trend in the dataset and not robust to the wide range of the real market conditions we experimented. From Tables 3.5, 3.6 and 3.7, we also found that LDA is the best conventional machine learning algorithm we have tested, and it could achieve an accuracy of up to 43%. The performance of our neural network-based TANN model is roughly 5% higher than the LDA method across all the FX pairs and prediction horizons, which certainly beats random guessing with a large margin. The TANN method can be used as a blueprint for further studies in the area, if not for trading signals directly.

From Tables 3.5, 3.6 and 3.7, we also noticed that for very short future period ($k = 5$), all models could not achieve classification performances as good as longer periods ($k = 10$ or $k = 15$). This is because FX prices are noisier and closer to random-walk processes for short time periods [150].

3.4.2 Dynamic Thresholds

All the test results discussed in Section 3.4.1 are using the labels (Up, Stationary and Down) calculated from fixed risk preference thresholds α defined in Table 3.3. These thresholds are calculated from the five years of data as a whole, which could indicate a long term-risk preference for a particular FX pair. However, this long-term fixed risk preference may not be appropriate for every trading day. From the data science perspective, inappropriate threshold α results in a very imbalanced training and testing dataset. Therefore, we could relax this constraint and adapt dynamic thresholds for every testing date. The dynamic thresholds are

Table 3.8: Classification accuracy differences between fixed and dynamic threshold settings with k = 15

FX	KN	LDA	LR	RF	SVC	TANN	Average
AUDJPY	0.86%	1.60%	-4.41%	0.80%	2.36%	1.91%	0.52%
EURSEK	-0.18%	-0.31%	-0.57%	0.04%	0.04%	0.39%	-0.10%
EURTRY	-0.01%	0.27%	-2.94%	0.19%	1.36%	1.57%	0.07%
EURUSD	0.56%	1.43%	-5.16%	0.63%	1.69%	2.20%	0.22%
GBPCAD	-0.12%	0.39%	-1.02%	-0.17%	0.42%	1.11%	0.10%
GBPJPY	0.87%	2.03%	1.86%	0.92%	2.82%	2.18%	1.78%
USDCHF	0.22%	1.32%	-1.32%	0.40%	1.31%	1.76%	0.61%
USDDKK	0.47%	1.61%	-4.13%	0.55%	1.82%	2.29%	0.43%
USDJPY	1.26%	2.34%	0.93%	1.16%	3.14%	2.68%	1.92%
USDNOK	0.02%	0.70%	-0.98%	0.14%	0.70%	1.45%	0.34%
Average	0.39%	1.14%	-1.77%	0.47%	1.56%	1.75%	0.59%

calculated based on the training window specifically to every test date. From the investment point of view, the risk preference changes every time we train a new model and is calculated from the most recent observations in the market. From the data science point of view, we assume the minutes' prices changes in training and testing datasets have similar ranges. Hence, their classification labels are both equally divided into three categories.

In Tables 3.8, 3.9 and 3.10, we list the classification accuracy of the dynamic threshold method minus the fixed threshold method averaged for every FX pair and every model we tested. There are a couple of important observations. First, dynamic threshold setting could

Table 3.9: Classification accuracy differences between fixed and dynamic threshold settings with k = 10

FX	KN	LDA	LR	RF	SVC	TANN	Average
AUDJPY	0.75%	1.66%	-3.61%	0.88%	2.59%	1.96%	0.70%
EURSEK	-0.17%	-0.09%	-1.01%	0.24%	0.18%	0.44%	-0.07%
EURTRY	-0.09%	0.51%	-1.47%	0.56%	1.29%	1.74%	0.42%
EURUSD	0.49%	1.84%	-4.37%	0.39%	1.94%	2.34%	0.44%
GBPCAD	-0.15%	0.70%	-0.66%	-0.35%	0.68%	1.31%	0.26%
GBPJPY	0.82%	2.09%	1.46%	0.66%	2.87%	2.28%	1.69%
USDCHF	0.28%	1.25%	-2.63%	0.29%	1.49%	1.74%	0.40%
USDDKK	0.51%	1.96%	-4.29%	0.63%	2.25%	2.44%	0.58%
USDJPY	1.41%	2.57%	0.20%	1.09%	3.50%	2.78%	1.93%
USDNOK	0.13%	0.76%	0.95%	0.07%	0.96%	1.39%	0.71%
Average	0.40%	1.33%	-1.54%	0.45%	1.78%	1.84%	0.71%

Table 3.10: Classification accuracy differences between fixed and dynamic threshold settings with k = 5

FX	KN	LDA	LR	RF	SVC	TANN	Average
AUDJPY	1.48%	3.26%	2.37%	1.66%	7.58%	4.71%	3.51%
EURSEK	-0.38%	-0.48%	-1.84%	-0.27%	0.25%	-1.15%	-0.65%
EURTRY	0.07%	1.55%	-0.55%	0.32%	2.28%	2.19%	0.98%
EURUSD	-0.96%	4.39%	4.93%	-0.88%	6.08%	5.52%	3.18%
GBPCAD	-0.63%	1.09%	4.14%	-0.48%	2.41%	2.07%	1.43%
GBPJPY	-0.10%	4.74%	8.22%	-0.12%	8.20%	5.47%	4.40%
USDCHF	0.93%	3.30%	0.74%	1.31%	4.36%	4.28%	2.49%
USDDKK	0.32%	4.47%	2.85%	0.20%	6.03%	5.69%	3.26%
USDJPY	1.34%	3.78%	3.75%	1.29%	7.25%	4.86%	3.71%
USDNOK	-0.63%	2.01%	4.92%	-0.42%	2.05%	2.81%	1.79%
Average	0.14%	2.81%	2.95%	0.26%	4.65%	3.65%	2.41%

achieve higher classification accuracy than fixed threshold setting, particularly for LDA, SVC, and our TANN model. This indicates that if an investor actively adjusts their risk preference to the current market environment, the model will perform better. However, this does not guaranteed a better profit in trading, as a trading strategy involves many other considerations such as capital availability, timing, risk management and practical constraints. Prediction of price change is only one area of concern. Second, some FX pairs (such as EURSEK) behave differently from others across all the models. Third, the LR model shows the most inconsistent performances for different FX pairs and prediction horizons. This proves again that a simple linear relationship assumption is not applicable to our problem.

3.4.3 Larger Training Window

We also experimented with using longer periods of data to train a TANN model as opposed to the default one-day training shown in Sections 3.4.1 and 3.4.2. We selected values of 2 and 3 for w. Since a larger dataset will significantly increase the training time of the model and computational cost, we only tested the year 2014 with a dynamic threshold setting. The comparison is shown in Table 3.11. We can see that when $w = 1$ and $w = 2$, the average test accuracy is not significantly improved. When w increases to 3, the model performances are deteriorated by more than 5%. Therefore, we could conclude that including long periods of historical information is not helping the model and probably adds more noise.

3.4.4 Universal Model

We also trained a universal model by combining all ten FX pairs of a single day, and tested it on the next trading day for individual FX pairs in this set. We take the data from three single months from September 2018 to November 2018 and set the prediction horizon k to 15

Table 3.11: Comparing different training window sizes

FX	$w = 1$	$w = 2$	$w = 3$
AUDJPY	51.93%	52.02%	41.32%
EURSEK	47.08%	44.05%	47.41%
EURTRY	47.35%	51.24%	43.44%
EURUSD	53.28%	53.26%	48.13%
GBPCAD	50.48%	60.00%	45.78%
GBPJPY	51.23%	49.62%	40.59%
USDCHF	52.93%	54.37%	48.66%
USDDKK	52.57%	52.02%	47.45%
USDJPY	53.41%	46.95%	45.34%
USDNOK	49.76%	53.39%	46.48%
Average	51.00%	51.69%	45.46%

minutes. The performances of the universal TANN are then compared to currency-specific TANNs. For the ten currency pairs and 65 trading days, we obtained 650 comparisons in total. We also increased the amount of training epoch for each model to 320 to accommodate the much larger training dataset. Figure 3.4 is the distribution of the accuracy of universal TANN minus the accuracy of currency-specific TANN; a positive value for a certain pair, say EURUSD, means that patterns learnt from other pairs in the ensemble are used to generalize and predict EURUSD. We found our universal model is on average increase the prediction accuracy by 1.1%. This indicates there are common features across different currency pairs, which can be utilized by neural networks. This is rather surprising given that in the ensemble there are not implied exchange rate (i.e., pairs XY, YZ, XZ) and that this is a quote-based market. This might indicate that the universal model is learning the pricing model of the market maker itself, which could be of independent interest and worthy of further investigation. We also notice that the distribution implied by the histogram in Figure 3.4 is more heavy-tailed than the one produced for stocks in [123] – we believe this is due to the dataset being relatively small for training a universal model with only ten currency pairs. We also believe that the model structure and training process should be further optimized for obtaining more robust universal models.

3.5 Conclusions

In this chapter, we purpose a novel pipeline for predicting short-term price movement directions in the FX market. We combined technical indicators with a neural network model for processing high-frequency FX tick prices, which are considered very noisy. We perform the same classification task for five different classical machine learning models, as well as our

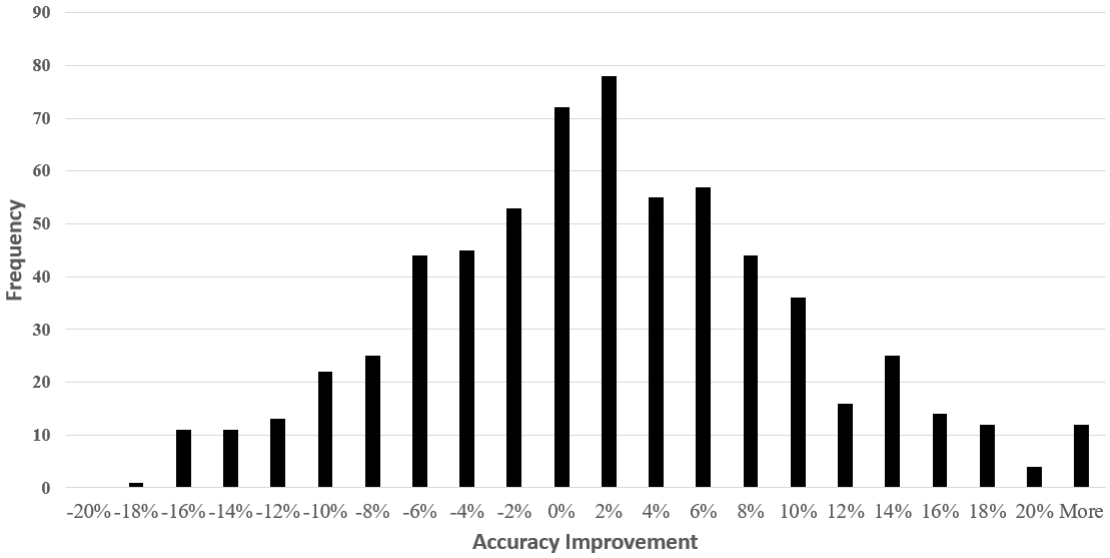


Figure 3.4: Comparison of Universal TANN and Currency-Specific TANNs

Technical Analysis Neural Network (TANN) pipeline. We found that every machine learning model performs differently for this task, and Linear Discriminate Analysis (LDA) is the best among those machine learning methods. Our TANN pipeline surpasses an LDA model with roughly 5% higher classification accuracy for three different prediction horizons (5 minutes, 10 minutes and 15 minutes). We also found that prediction of future price movement of smaller prediction horizon is a more challenging task for all machine learning and our neural network model because short-term FX prices is noisier. Our default setting is collecting one-day FX prices and test on the following trading day, and we also tried to collect longer historical periods (i.e. two days and three days), which discover that longer periods would only degenerate model performances.

We discussed how the investor's risk attitude would influence the balance of the labels in the dataset – recall that there are no natural classification labels associated with financial time series data, which is a key difference of computational finance problems with computer science tasks. If the investor's risk attitude is changing dynamically as the market

evolves, model performances could be improved. This suggests that market participants should always try to understand the evolution of market environment, and not blindly reply on computer algorithms for making investment decisions.

In addition, we also highlighted that universal features exist in high-frequency FX, and can be utilised by our neural network model to further improve classification results. The classification accuracy of the universal model is encouraging.

Chapter 4

Efficient Hedging Frontier

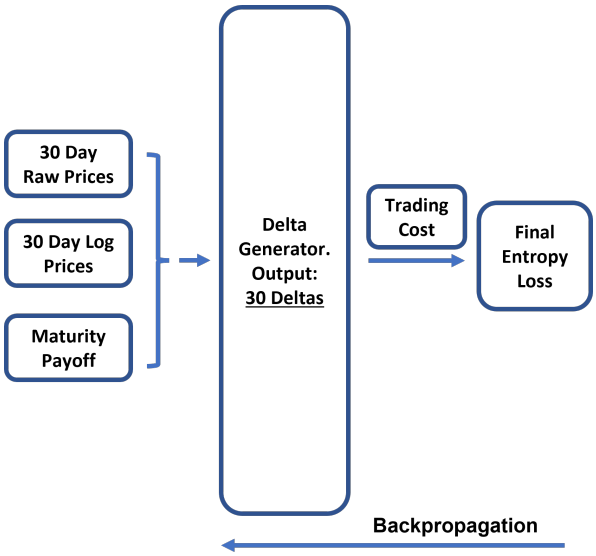
The trade off between risks and returns gives rise to multi-criteria optimisation problems that are well understood in finance, efficient frontiers being the tool to navigate their set of optimal solutions. Motivated by the recent advances in the use of deep neural networks in the context of hedging vanilla options when markets have frictions, we introduce the **Efficient Hedging Frontier** (EHF) by enriching the pipeline with a filtering step that allows to trade off costs and risks. This way, a trader's risk preference is matched with an expected hedging cost on the frontier, and the corresponding hedging strategy can be computed with a deep neural network. We experiment on the default Heston model of the underlying asset prices, as well as the rough Bergomi model where price increments are not independent.

We further develop our framework to improve the EHF and find better hedging strategies. By adding a random forest classifier to the pipeline to forecast market movements, we show how the frontier shifts towards lower costs and reduced risks, which indicates that the overall hedging performances have improved. In addition, by designing a new recurrent neural network, we also find strategies on the frontier where hedging costs are even lower.

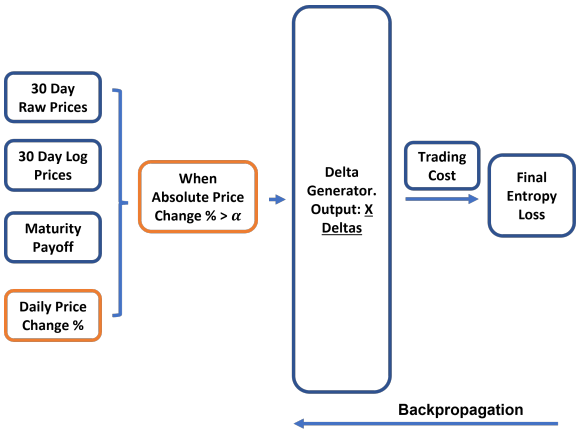
4.1 Introduction

In the past decades, the evolution of financial derivative markets has provided investors numerous opportunities for trading and, especially for managing risks associated with future commodity prices, stock prices, interest rates and exchange rates. The markets expanded massively in the past ten years [151]. As we discussed in Section 2.2.3, an investor utilises the option to benefit from a future price movement of the underlying asset which aligns with her expectation, and avoid the risk if the price moves in the opposite direction [152]. The option buyer pays an option premium to the issuer at the inception of the contract, and both parties could frequently trade the underlying asset to hedge their exposures to the price movements. Therefore, finding a better solution for working out an appropriate option premium and generating hedging strategies is fundamentally important for both parties to ensure fairness in the option contract and appropriately manage associated risks.

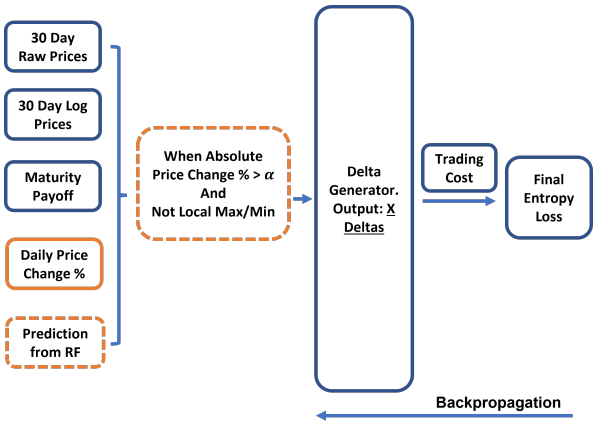
The default method to work out an option price and its hedging strategies is the classical Black-Scholes-Merton (BSM) method as we mentioned in Section 2.2.5, and there are quite a lot of articles discussing and proposing solutions to overcome the limitations of BSM framework. Jankova argued that constant volatiltiy assumption is the key issue of the BSM, and suggested to include stochastic volatility element [153]. The Heston and Bates process are two famous stochastic volatility models that introduce uncertainties in the behaviour of volatility, and consequently allow simulating the price evolution of financial assets more realistically [52][154]. Beginning with Hutchinson et al., neural network models were considered as a non-parametric solution to solve option pricing and hedging problems [129].From the universal approxiamtion theory disscussed in section 2.3.5, we know that a neural network model has the capability of arbitrarily approximating any nonlinear relationship [155] . Deep



(a) Original Deep Hedging



(b) Deep Hedging with Price Change Threshold



(c) Deep Hedging with Price Change Threshold and Random Forest

Figure 4.1: The Original and Amended Deep Hedging

Hedging is one of the most recent advances in this line of work — its technical pipeline is depicted in Figure 4.1a for an option with 30 days maturity [17]. The creators of Deep Hedging propose a framework to replicate conventional delta hedging strategies with learning networks. As we mentioned in Section 2.3.6, the Deep Hedging pipeline is a reinforcement solution for hedging of option contracts, so that delta generator is the policy generator in reinforcement learning terms, and the final entropy loss defines the rewards for the system. With Deep Hedging, traders could optimize models under different levels of transaction costs, as well as various risk measurements and risk appetites. It is also found in [17] that neural networks achieve better hedging performances than Black-Scholes-Merton model with real S&P500 index data when re-calibrated on a daily basis. There are also follow-ups stemming from this work [152][131][156][130].Each of these follow up provides an alternative or extended algorithm that generating hedging strategies for option contracts from neural network models. There are literatures focusing on implementing neural network hedging from the reinforcement learning aspect [131][152]. HedgeNet method focuses on minimize the hedging error instead of risk adjusted return or final utility [130].

However, all these papers assume the hedging trades happen at regular intervals (e.g. every day, every two days), but this restriction is not realistic. It is certainly not the case on the trading floor. Since continuous trading is impossible, traders often make decisions based on personal experience and knowledge. They decide the best timing to re-balance their hedging positions. For example, if it is believed that the underlying price is going to experience a V-shaped (or reverted V-shaped) pattern, then it is clearly a waste of money to sell some share and then, within a short period of time, buy it back as transaction costs would never be zero.

In this work, we tackle the problem of letting the algorithm self-decide when it is the best

moment to buy or sell the underlying asset. The decisions are based upon historical prices of the underlying, as well as the model's expectations about future prices movements. From the technical point of view, these two new inputs act as a *filtering* step for the deep hedging network.

We first introduce a price change threshold which restricts the model to perform trades only when the underlying prices experience significant movements, see Figure 4.1b. The price change threshold in this chapter is to decide whether hedging trades should happen or not, whereas in Chapter 3, we also have a price change threshold to decide the prediction labels. These two thresholds both could reflect a model user's risk attitude towards the market, but they are applied for different behaviour (trading and hedging). By changing the value of the price threshold α, we get different incomparable hedging strategies. The larger α we use, the smaller the number of trades used by the strategy; consequently, we have smaller hedging costs (given the lower transaction fees) but bigger risk of experiencing a large loss at maturity (given that we hedge less effectively). More formally, costs and risks here are measured in terms of the mean and variance of the termination loss over a large number of market paths, respectively. We call the **Efficient Hedging Frontier** (EHF) this curve of undominated strategies in the cost–risk space, inspired by the efficient frontiers defined in portfolio optimisation [157] and algorithmic execution [158]. From the EHF, a market participant can pick a trade-off strategy which satisfies her risk and return preferences, by choosing an adequate value of α.

Our second filtering step is depicted in Figure 4.1c. We additionally place a random forest classifier before the network to predict future movements of underlying prices. The classifier would instruct the neural network to hold its position if it believes a V-shaped pattern is coming. By adding the classifier, we could shift the EHF and obtain strategies where the

mean and variance of termination losses are reduced simultaneously.

Finally, we also experiment with the architecture of the hedging neural network and test the effectiveness of using recurrent architectures to leverage the temporal relationships in the price time series. We show how such a design choice can further shift the EHF towards even better strategies.

The remainder of this chapter is organised as follows. Section 4.2 gives an illustration of what is the generated hedging strategy from default deep hedging pipeline, so we could compare with our proposed method directly. Section 4.3 describes our first setting which uses a predefined price change threshold to constrain the model trading activities. Section 4.4 shows how a classifier could benefit a Deep Hedging neural network to reduce both hedging costs and risks. Section 4.5 presents the results from our experiments and compares them with Black-Scholes-Merton and Deep Hedging in default settings. In Section 4.6, we show our further experiments which replace dense networks with recurrent neural networks for Deep Hedging. Section 4.7 concludes this research chapter.

4.2 An illustration of Default Deep Hedging

We have given detailed discussions of the Deep Hedging model in Section 2.3.7, which includes its motivation, model structure, typical loss functions, and its strengths and weaknesses. In this section, we are going to give an example of the hedging strategies generated from the default Deep Hedging model in table 4.1

The limitations of default Deep Hedging pipeline is quite obvious. The delta generator (as shown in figure 4.1a) takes price information from every trading day, and outputs one best value of delta for that particular day. Sometimes, from one day to the next, the price change

Table 4.1: An illustration of the Default Deep Hedging

Day	Price	DH Delta	Buy/Sell	Trading Cost
0	100.00	0.4090	40.9042	2.0452
1	**100.13**	**0.4092**	**0.0178**	**0.0009**
2	106.12	0.4334	2.5711	0.1286
3	**106.34**	**0.4377**	**0.4471**	**0.0224**
4	***109.43***	***0.4559***	***1.9992***	***0.1000***
5	106.71	0.4704	1.5435	0.0772
6	102.52	0.4711	0.0684	0.0034
7	**102.28**	**0.5039**	**3.3557**	**0.1678**
8	101.99	0.4921	-1.2001	0.0600
9	***105.46***	***0.5205***	***2.9990***	***0.1500***
10	103.59	0.5114	-0.9491	0.0475

is negligible and the model buys or sells little amount of underlying asset. More importantly, when there is a V-shaped movement of underlying in two consecutive days, the model would sell some underlying and then buy them back, which causes unnecessary trading costs. This is illustrated in Table 4.1. At day 1, 3 and 7, the price changes are small comparing with the previous days. At day 4 and day 9, they are peak values of underlying prices. It is reasonable for a trader takes no actions in these days; in this example, this will save roughly 15% of trading costs in a 10-day period.

4.3 Deep Hedging with a Price Change Threshold

We first try to limit Deep Hedging to generate deltas only when the underlying price changes significantly from one day to another. Therefore, we introduce an additional input feature to the delta generator. As from above, this amended Deep Hedging pipeline is shown in Figure 4.1b, where orange denotes our novel filtering.

The absolute percentage changes of daily prices are calculated from simulated trajectories, and only the days with absolute price changes higher than a predefined threshold α will be considered by the neural network. For the other days, the deltas will remain unchanged, and therefore no buy or sell actions are taken. For the default Deep Hedging algorithm, the model always outputs 30 deltas for each input path. By adding the threshold α, the trading frequency for each path reduces from 30 (daily trading) to roughly 0 (no trading) as α increase from 0 to 0.2 for Heston simulation with high volatility settings. We simulated 120,000 paths for our experiments, and Table 4.2 shows the average number of trading days for one trajectory when the value of α varies. For example, if α is set to 0.04, there would be only 9.53 trades performed during the 30-day period.

Table 4.2: Trading frequency reduction as α increases

Threshold (α)	Average Frequency
0.00	30.00
0.02	16.64
0.04	9.53
0.06	5.20
0.08	2.73
0.10	1.40
0.12	0.71
0.14	0.36
0.20	0.05

4.4 Deep Hedging with a Classifier

A classifier is a model used to divide non-labelled data into different categories. It is very commonly applied with financial time series to predict future movements of asset prices. A decision tree is one of the most fundamental supervised classification model, which can be used to discover features and extract patterns for discrimination and predictive modelling [134]. The idea is basically to break up a complex task into many simpler decisions, and for each decision, the algorithm tries to increase the homogeneity of each classification category. As discussed in section 2.4, Random forest (RF) is a popular ensemble model of many decision trees, where each tree is trained with a sub-sample of the training dataset. The output is generated from votes of the trees and therefore could improve the predictive accuracy and control over-fitting [159]. Our random forest classifier takes log of normalised

stock price (S_t / S_0) in the past 2 days as input, and output one classification label. It utilizes fifty tress in the ensemble and implements Gini Impurity as the loss function.

Before running the Deep Hedging network, we first label our simulated daily prices with two labels. If one day's precentage change of the underlying price is higher (lower) than yesterday and lower (higher) than tomorrow by some threshold β, we label it as zero, otherwise we label it as one. Basically, zero means do not trade one that day, because today's profit (loss) will be recovered tomorrow. We set β to be 0.05 for our experiments. The value of β is generally reflecting the size of the V-shaped movement of stock price, which we want to avoid. We then take log-normalised prices from the previous two days and train a random forest classifier to classify every daily price into category zero or category one. Because we use synthetic data, the classification accuracy is relatively high, with more than 90% for test data. Subsequently, we start training the Deep Hedging network, and add the labels predicted from the random forest classifier as an extra feature. These labels will instruct our neural network to skip the days, where underlying prices are local maxima or local minima, see Figure 4.1c.

4.5 Experimental Setting and Results

We simulated 120,000 trajectories for our experiments, split in 100,000 for training and 20,000 for testing. We train the network to optimise the issuer's accumulated cost at the maturity of an option contract, which we refer to as termination loss. The paths are simulated using the Heston model and the rough Bergomi (rBergomi) model, and the detailed formulation of these two approaches have been discussed in Section 2.2.6 and Section 2.2.7 respectively.

Table 4.3: Heston parameters used in our experiments

Market Scenario	v_0	θ	κ	μ	σ	ρ	S_0
Low Volatility	0.4	0.4	1	0.01	4	-0.7	100
High Volatility	0.8	0.8	1	0.01	4	-0.7	100

We use two sets of Heston parameters for our experiments, which try to simulate market scenarios with different levels of fluctuation. The values for those parameters are shown in Table 4.3. For the rBergomi simulations, the parameters S_0 and ρ are the same as Heston model. We also set $\eta = 1.9$ and $V_0 = 0.05$. We change the values of H to demonstrate how the EHF react to the correlation of price changes, which will be discussed further. These paths are simulated using the popular quantitive analysis software package of QuantLib [160].

Our neural network models are implemented in Python with Tensorflow [71], and the neural network structure is the same as default deep hedging, with 2 fully connected layers (delta generator) and 30 neurons at each layer. The model is trained with Adam optimizer with learning rate of 0.0001 and batch size of 128 [72]. The loss function is the entropy function purposed by default deep hedging with risk measurement parameter λ. The random forest classifier utilised is from scikit-learn package [147].

In this section, we first present our experiments under high market volatility scenario with Heston simulations, and then follow by discussing different market conditions. After that, we will also show the EHF with the rBergomi simulations.

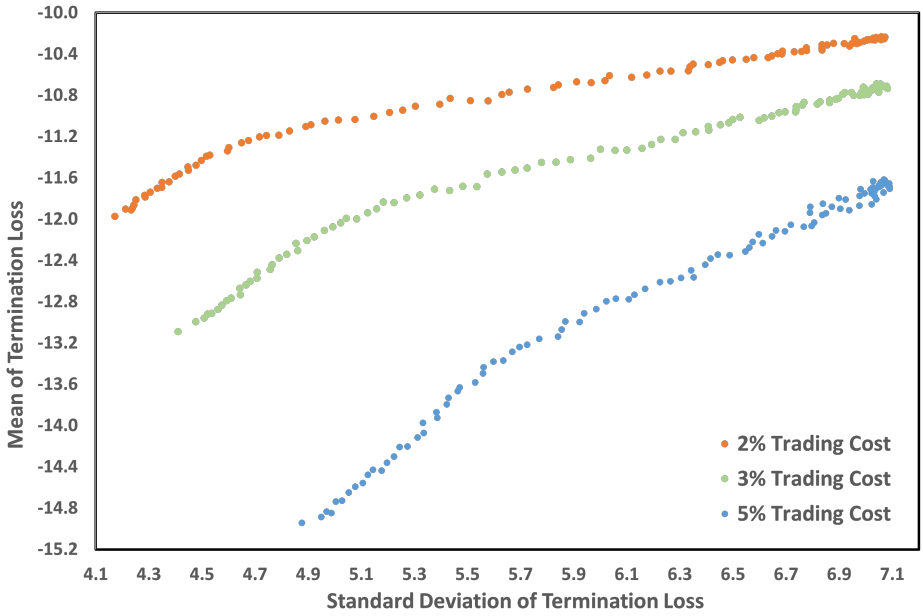


Figure 4.2: The Heston EHFs for different trading costs ($\lambda = 0.5$)

4.5.1 Heston Simulation with Various Trading Costs

As mentioned in Section 4.1, our ultimate objective is to reduce unnecessary trading for our Deep Hedging system. Using the approach discussed in Section 4.3 we first attempt to force the network to only focus on trading days where there is a significant price changes. Comparing with the default Deep Hedging [17], there is one additional input feature of the daily price change percentage. The price change threshold α could reduce the average termination losses for our 120,000 simulated paths, but also increase the standard deviation. Therefore, by tuning the value of α we could obtain the EHF under high volatility market assumption as shown in Figure 4.2.

There are three colours in Figure 4.2 indicating different market trading cost assumptions. Market costs are assumed to be proportional to the cash amount spent for buying/selling the underlying assets. There are 100 points for each line, and each point represents one particular price change threshold α selected evenly from 0 to 0.2. The Y-coordinate of a

point in Figure 4.2 is the mean of 20,000 termination losses from testing trajectories for a given value of α. The X-coordinate is the relevant standard deviation of these losses. As α increases, the points move from left to right. Therefore, the bottom-left point is the standard Deep Hedging that trades every single day ($\alpha = 0$). At the top-right point of each line, where $\alpha = 0.2$, the system is making only 0.05 trades during the 30-day period (see Table 4.2). The average loss over 20,000 test paths gets very small (i.e. no trading cost), but the standard deviation of losses is significant (i.e. no hedging). It is also worthwhile to observe that at right end of each line, there are clusters of points. The explanation is that when α makes small changes at high values (e.g., from 0.196 to 0.198), the algorithm could not filter out many extra trading days, so the results are faltering because of the randomness nature of neural networks.

At 5% trading cost and $\alpha \in [0, 0.1]$, we can calculate the average of mean termination losses as well as the average of standard deviations of termination losses from those points in in Figure 4.2. The statistics are -13.628 and 5.578 respectively. If the hedging strategy is calculated with the Black-Scholes-Merton method instead, and average of means and average of standard deviations are -14.790 and 6.978 with the same values of α. This also proves Deep Hedging outperforms delta hedging by a clear margin with Heston simulations.

Clearly, adding a price change threshold is not actually improving Deep Hedging but provides a new prospective for trading-off risks and returns. An investor could decide a point on the efficient hedging frontiers to represent her risk appetite and then make the appropriate hedging strategy decisions.

Table 4.4: Improved Deep Hedging with Random Forest

Day	Stock Price	DH Delta	Buy/Sell	Trading Cost
0	100.00	0.4334	43.3373	0.8667
1	97.09	0.4346	0.1144	0.0023
2	93.72	0.4300	-0.4301	0.0086
3	**101.45**	**0.4300**	**0.0000**	**0.0000**
4	93.91	0.4331	0.2969	0.0059
5	80.61	0.3064	-10.2177	0.2044
6	82.60	0.3274	1.7344	0.0347
7	89.02	0.3803	4.7137	0.0943
8	**96.33**	**0.3803**	**0.0000**	**0.0000**
9	84.12	0.3122	-5.7299	0.1146
10	83.97	0.3129	0.0603	0.0012

Table 4.5: Improvement through RF classifier ($\lambda = 0.5, \alpha \in [0, 0.1]$)

	Mean of Losses			Standard Deviation of Losses		
	2% Cost	3% Cost	5% Cost	2% Cost	3% Cost	5% Cost
DH	-11.253	-11.898	-13.628	4.887	5.143	5.578
DH with RF	-10.718	-10.784	-11.683	4.209	4.353	4.543
Improvement	4.75%	9.36%	14.27%	13.87%	15.37%	18.54%

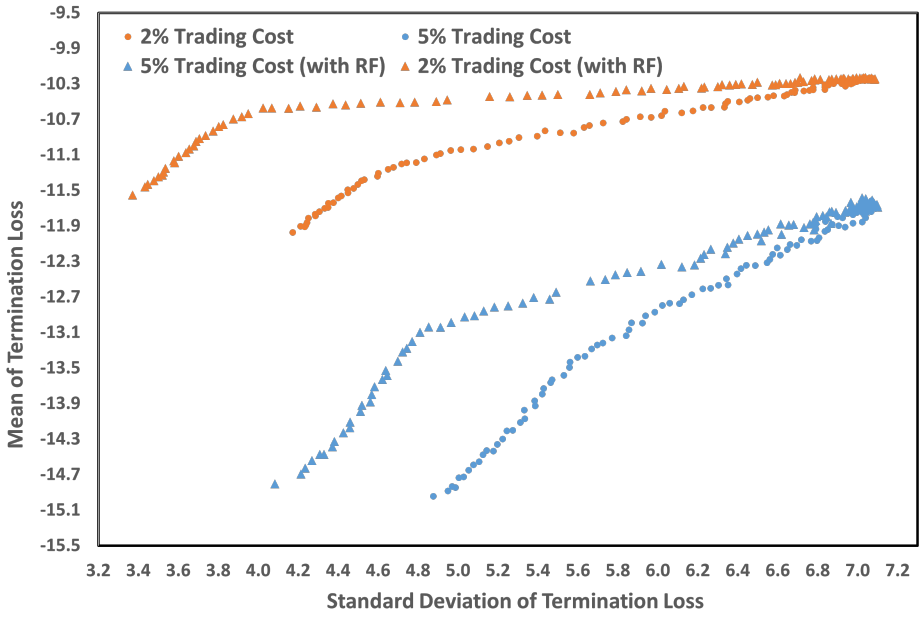


Figure 4.3: The Heston EHFs with Random Forest forecast ($\lambda = 0.5$)

4.5.2 Heston Simulation with Random Forest Classifier

Our next step is to combine the Deep Hedging algorithm with a random forest classifier, as shown in Figure 4.1c. The classification labels generated from the random forest is treated equivalently to the trader's expectations of the future movements of underlying prices. We show that if the classification task is solved sufficiently well (or, equivalently, the trader has good knowledge of the market) the hedging losses and risks could be reduced simultaneously. For high volatility market scenario, the result is shown in Figure 4.3. There are two groups of lines, which represent two trading cost assumptions. There are two lines in each colour group. The higher line shows the performances of Deep Hedging with the help of random forest classifier. At low values of α (i.e. left end), the gap between performances of Deep Hedging with and without random forest is larger than at high values of α (i.e. right end). When α is really large, the two models exhibit similar performances, as α is filtering out most trading days and good predictions could not make much contributions.

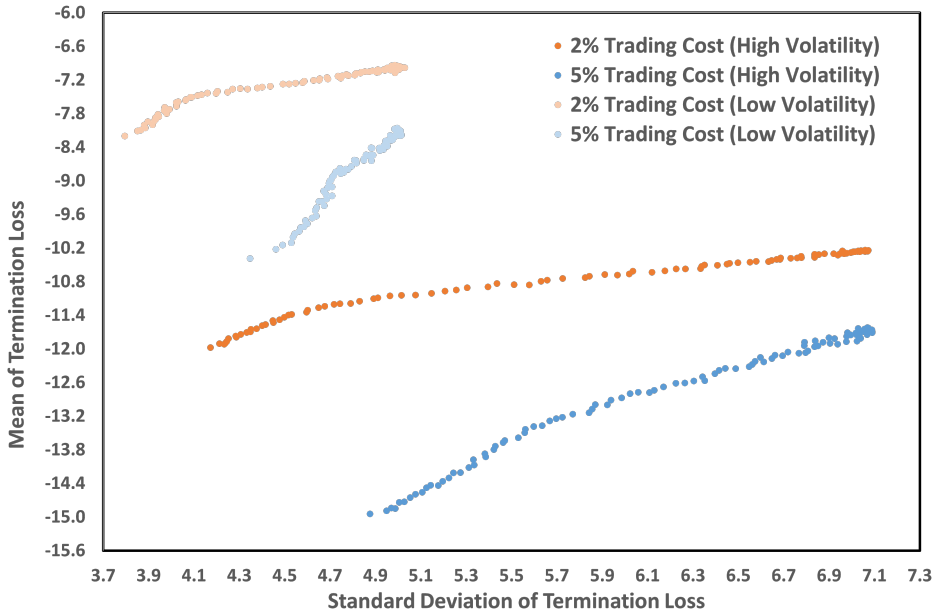


Figure 4.4: The Heston EHFs under different market conditions ($\lambda = 0.5$)

Table 4.4 gives an illustration of how the combined system makes hedging decisions. The forecasts from the random forest algorithm instructs the neural network that day 3 and day 8 are local maximum points for the underlying, according to its threshold (5%), therefore the hedging generator skipped both days. The numerical comparisons of Deep Hedging with and without random forest classifier are shown in Table 4.5. Note we only take values of α from 0 to 0.1 for calculating the averages, as larger values limit the trading frequencies too much. Overall, the improvement for standard deviations of termination losses is higher than for means of termination losses, which can also be visually observed in Figure 4.3.

4.5.3 Heston Simulations with Different Market Conditions

We also test the model in the low volatility market condition; the EHFs are displayed in Figure 4.4. The price change thresholds α considered are still in the interval from 0 to 0.2 and evenly distributed. It is noticed that when the underlying asset is less volatile, both

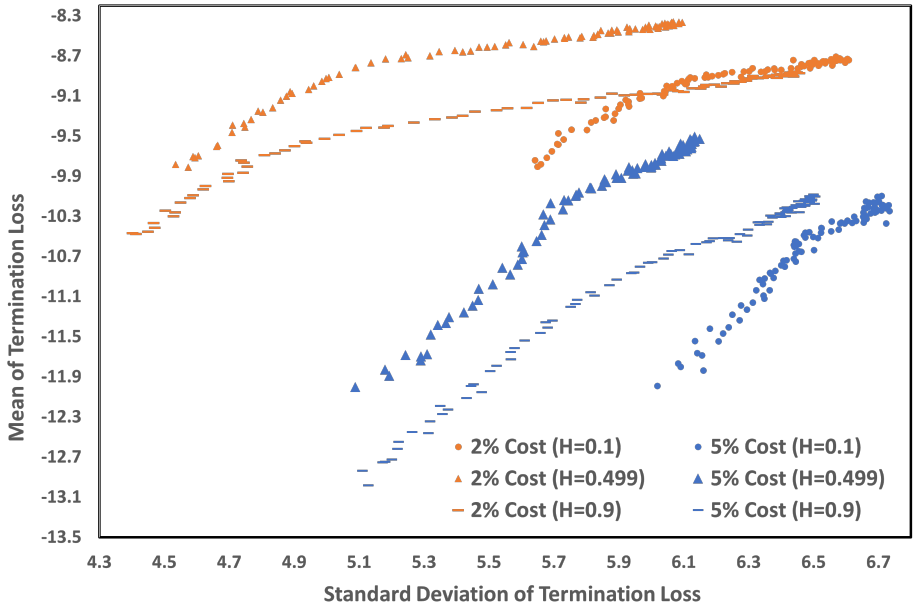


Figure 4.5: The rBergomi EHFs for different trading costs and H ($\lambda = 0.5$)

mean and standard deviation of termination losses are reduced as expected. In addition, the length of the EHF is getting shorter and the points are more compactly distributed. It is also observed that the slope of the frontier is smaller with 2% trading cost than with 5% in both market conditions.

4.5.4 Rough Bergomi Simulation with Different Hurst Parameters

If the underlying prices are simulated from a rBergomi model, there are similar patterns with the Heston simulations. Figure 4.5 shows that the EHFs with low trading costs have smaller slope than those with high costs. For any value of H larger or smaller than 0.5, both mean and standard deviation of termination loss are increased. It is also noticed that for $H = 0.1$, the mean and standard deviation have the minimum ranges.

The above experiments are carried out with entropy risk measure parameter $\lambda = 0.5$. If λ changes to 0.7, the frontier will shift slightly to the right and when it is 0.2, the frontier is

Table 4.6: Comparing neural network architectures ($\lambda = 0.5, \alpha \in [0, 0.1]$)

	Mean of Losses			Standard Deviation of Losses		
	2% Cost	3% Cost	5% Cost	2% Cost	3% Cost	5% Cost
DH	-11.253	-11.898	-13.628	4.887	5.143	5.578
DH with GRU	-10.938	-10.685	-13.094	5.073	5.259	5.597
Improvement	2.79%	1.79%	3.92%	-3.18%	-2.27%	-0.35%

slightly to the left. This is expected since the EHF moves in the same direction of the trader's risk aversion.

4.6 Updating the Neural Network

As discussed above, the default Deep Hedging model utilises two fully connected layers for the delta generator, and the input is only one daily price. Therefore, it does not consider the temporal relationships of underlying time series. It is very common and intuitive to choose recurrent neurons instead of dense connections for this problem.

We tested the use of Gated Recurrent Unit (GRU) as the recurrent element and re-designed the Deep Hedging pipeline. As illustrated in Figure 4.6, we use a vector (instead of a single number) to input historical prices from Heston simulations in the past 3 days to the GRU layer. The delta generator consists of two recurrent layers each with ten recurrent units and one dense layer to output a single number, which means the optimal amount to hold the underlying asset. We need to point out that in the first two days for a trajectory, there are not enough past prices for constructing the vector, so for those we still incorporate dense layers as the default Deep Hedging.

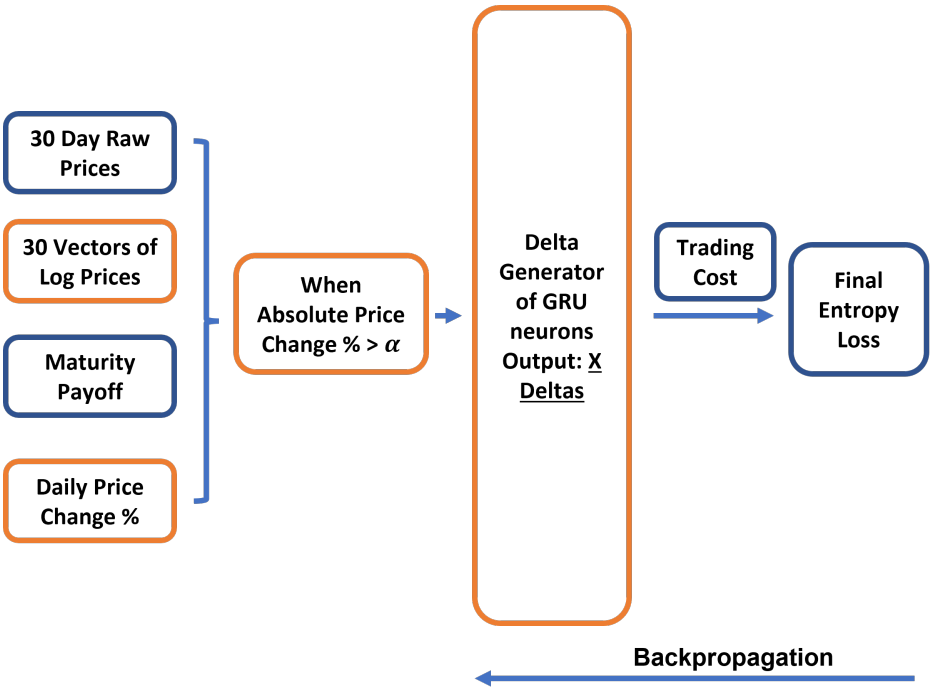


Figure 4.6: Deep Hedging using Gated Recurrent Network

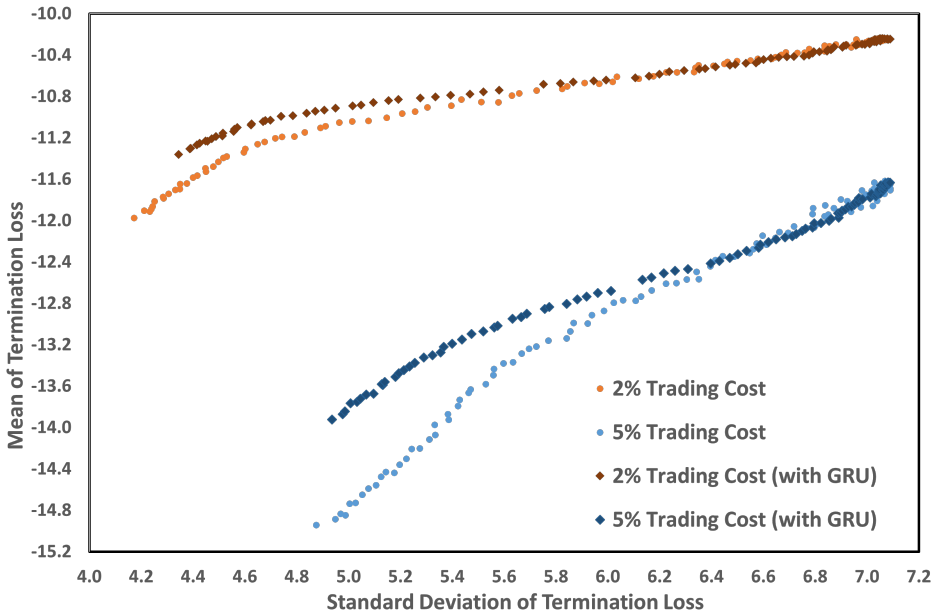


Figure 4.7: The Heston EHFs with GRU neural network ($\lambda = 0.5$)

Comparing the EHFs of the standard Deep Hedging with GRU version, we can conclude from Figure 4.7 that while the mean of termination losses are reduced with the GRU architecture for small values of α, the expected deviation of losses increases. The two lines overlap rather quickly as price change threshold α gets larger. The numerical results are shown in Table 4.6.

4.7 Conclusions

We wanted to limit the trading activities of a Deep Hedging model so that unnecessary trading costs could be saved. By adding a price change threshold, which filters out trading days with insignificant price movements, we could generate an efficient hedging frontier. On the frontier, a market participant could intuitively balance his/her position between risk tolerances and expecting losses when hedging a European call option, and generate appropriate strategies accordingly. We experimented with various trading costs and market volatility assumptions, as well as different values of λ for entropy risk measures. We could also improve the efficient hedging frontier by incorporating a random forest classifier with the Deep Hedging neural network. Outputs from the classifier are treated as prior knowledge of how the underlying price will evolve in the near future, which helps the delta generator network to avoid trading against V-shaped movements. Experiments with rBergomi show similarities with Heston simulations, as well as differences when the H parameter changes. In addition, our experiments also proved that replacing dense layers with GRU layers could reduce the expected mean of termination losses for Deep Hedging, but increase the standard deviations.

Chapter 5

A New Encoding for Implied Volatility Surfaces

In financial terms, an implied volatility surface can be described by its term structure, its skewness and its overall volatility level. We use a PCA variational auto-encoder model to perfectly represent these descriptors into a latent space of three dimensions. Our new encoding brings significant benefits for synthetic surface generation, in that (i) scenario generation is more interpretable; (ii) volatility extrapolation achieve better accuracy; and, (iii) we propose a solution to infer implied volatility surfaces of a stock from an index to which it belongs directly by modelling their relationship on the latent space of the encoding. All these applications, and the latter in particular, have the potential to improve risk management of financial derivatives whenever data is scarce.

5.1 Introduction

The Black-Scholes-Merton (BSM) model is the benchmark for stock option pricing and valuation [47][48]. As mentioned in Section 2.2.5 and section 2.2.9, one major weakness of the standard BSM method is the assumption that volatility of the underlying stock is a constant, and not related to the moneyness or the remaining term of the option contract. This does not conform with our observations in the market.

If we equate the BSM formula with market option prices and solve for the volatility parameter, we get the implied volatility value. Implied volatilities have a typical characteristic, which is, the value of volatility gets smaller as the option moneyness move close to one and gets larger as it moves away from one. This phenomenon is referred to as the implied Volatility Smile or Volatility Skewness [161]. If we take the remaining term of an option contract as another dimension into the volatility smile, we get the Implied Volatility Surface.

Implied volatility surface is the most important variable to consider if anyone wants to design, trade or evaluate financial derivatives. The most straightforward use case is to calculate the option prices of non-existent term and moneyness combinations of a particular underlying stock [162] on a particular trading day. This involves the process of predicting an extended subset of volatility values on a surface from some existing values that can be observed from the market. This task is commonly referred to as the implied volatility extrapolation. Once the extended volatility values are extrapolated, they can be put into the BSM formula to calculate the required prices of financial derivatives of the special term and moneyness. The most common scenario is to extrapolate from products with short-term expiration to products with long-term expiration. The profit or loss of issuing such products are largely depended on the accuracy of the extrapolated implied volatility values.

Moving one step further, we could not only predict the extended term and moneyness combinations, but also predict implied volatility surfaces of a specific stock from another related stocks or stock index. We consider a scenario where, on a particular trading day, we have zero information about a stock's implied volatility surface, and therefore we cannot perform extrapolation. However, we have the historical implied volatility surfaces of this stock and some other related stocks or stock indexes. We need to identify the relationships between different stocks/indexes from their empirical volatility surfaces and predict future implied volatility surfaces from scratch. This task is most challenging, and to the best of our knowledge, there is no established method to solve this problem.

It is also believed that the shape of an implied volatility surface reflects the current market perception of risk and return, and the implied volatility values indicate the demand and supply for different combinations of moneyness and term structure [163]. Ideally, market participants could generate many implied volatility surfaces with similar shape but not identical values, all representing typical market environments, and use this batch of synthetic surfaces for evaluating their derivative trading engines. Because stress scenarios are rarely represented in the historical data, the ability to generate synthetic implied volatility surfaces of good quality and in an interpretable manner is crucial.

In this paper, we present a new encoding mechanism of implied volatility surfaces using a PCA variational auto-encoder model. The new encoding significantly improves the state of the art for the three problems mentioned above.

There is limited literature on the modelling of implied volatility surfaces with neural network approaches [164][165][15][166]. Their work did not clarify the question of what features of the implied volatility surfaces are exactly encoded or learnt by the neural network model. This is a crucial issue, as we want to generate (a subset or the whole of) a volatility surface to

perform the aforementioned tasks. If we do not know how the neural network model encodes the market data or what is represented in the neural network model, we cannot interpret the generated synthetic surfaces or infer the implied volatility surface from an index to a stock.

The remainder of this chapter is organised as follows. In section 5.2, we explain the PCA variational auto-encoder model, which is used to encode the implied volatility surface differently for performing the tasks mentioned above. Section 5.3 introduces the data we used for our experiments and, more importantly, how we evaluate the usefulness of our generated synthetic implied volatility surfaces. Section 5.4 introduces our method of encoding implied volatility surfaces; we compare our approach with an alternative solution highlighting the differences on the training process, as well as the unique characteristics of the encoded latent space. Section 5.5 shows how we can leverage the new encoding to efficiently solve the three challenges on synthetic surfaces generation; interpretable scenario-based generation, implied volatility extrapolation and stock-specific generation. Section 5.6 concludes our work for this chapter.

5.2 PCA Variational Auto-Encoder

To address the problems discussed above, we upgrade the variational auto-encoder (VAE) model introduced in section 2.3.4 to a PCA variational auto-encoder.

As discussed in section 2.3.1, the functionality of a neural network model is effectively controlled by its loss function. Different loss function instructs the model to capture different features from the training data.

The PCA auto-encoder model was first developed by Ladjal et al. [16]. They made an important change to the loss function of a classic auto-encoder by adding an extra term L_{cov},

which measures the covariance between all pairs of latent dimensions. They try to reduce the value of L_{cov}, which effectively forces the latent dimensions to be independent to each other. The formula for L_{cov} is:

$$L_{cov} = \sum_{i=1}^{D} \sum_{j=i+1}^{D} \left[\frac{1}{B} \sum_{t=1}^{B} (Z_{t,i} Z_{t,j}) - \frac{1}{B^2} \sum_{t=1}^{B} Z_{t,i} \sum_{t=1}^{B} Z_{t,j} \right] \tag{5.1}$$

where B is the number of data point in a training batch, and we combine the concept of PCA auto-encoder with the variational auto-encoder models. The final loss function for training our PCA variational auto-encoder becomes:

$$L = L_{recon} + \lambda_{kl} \cdot L_{kl} + \lambda_{cov} \cdot L_{cov}. \tag{5.2}$$

The value of λ_{cov} controls how much restriction we want to enforce on the model due to the covariance. On one hand, if λ_{cov} is too small, the latent dimensions are not independent. On the other hand, the model loses the ability to generate surfaces if λ_{cov} is too large. By trial and error, we found the optimal value of λ_{cov} is 0.1 for training a PCA variational auto-encoder on implied volatility surfaces.

We will be using the VAE model structure together with this loss function to train the PCA variational auto-encoder models. The encoder and decoder neural networks are identical, with each network has 2 fully connected hidden layers and 32 neurons for each layer. The activation function is Relu function for all layers, and the model is trained with Adam optimizer with learning rate of 0.0001.

We are going to demonstrate their unique characteristics for the model training process and encoded latent space, and their strengths for dealing with the three synthetic surface generation challenges mentioned in the introduction.

5.3 Dataset and Evaluation Criteria

Our dataset consists of daily implied volatility surfaces for the period from 4 October 2016 to 4 October 2021. We collected data from 44 European listed stocks, as well as the stock index of STOXX50[1]. We also have the daily high, low, open, close prices of these stocks and index over the same period of time. The volatility surfaces computed before 4 October 2020 are marked as training datasets, and those after that date are testing datasets.

Among the 44 stocks, we take 6 stocks out of the dataset for performing the volatility surface extrapolation and specific stock generation tasks. We want to emphasize that the PCA variational auto-encoder model is capable of learning the general features of volatility surfaces, which not only exist among the temporal dimensions but also between difference stocks.

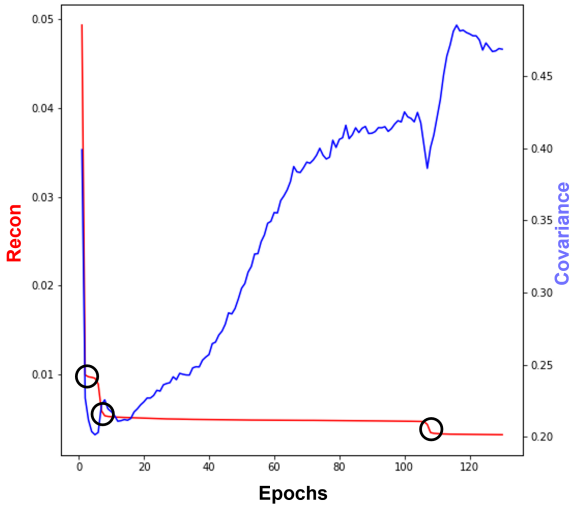
After we generate synthetic implied volatility surfaces from our model, we wish to measure how realistic or useful they are. To evaluate a model, one often calculates the numerical difference, as, e.g., the mean absolute error (MAE), between every point on a real surface and the generated surface. The conclusion would be that if MAE values are smaller, the model performs better. However, this seems unsatisfactory for two reasons. Firstly, the financial dataset sampled from the market contains a lot of noise. Consequently, we should not force the model to learn every piece of the data and the aforementioned evaluation method would more easily lead to overfitting. The MAE and other similar measurements are more adequate for computer science tasks, where the data has much less misleading information. Secondly, if we generated a surface that is identical to the market surface (i.e., with zero MAE) then it would not be useful in our context. We want our synthetic surfaces to be similar

[1] STOXX50 is a stock index covers 50 stocks from the Eurozone

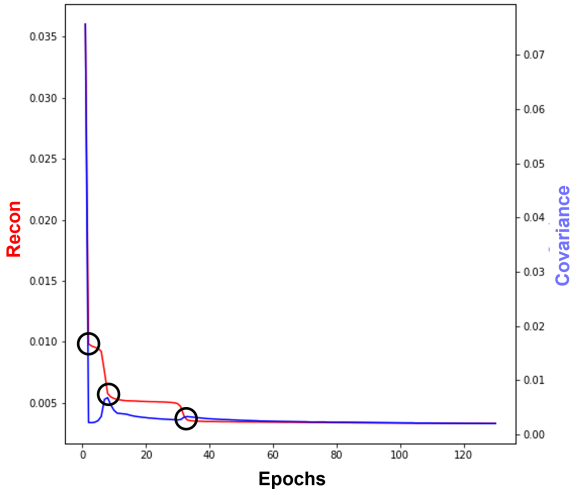
Table 5.1: Evaluation Thresholds for Implied Volatility Surfaces

	$M \in (0, 0.9]$	$M \in (0.9, 1.05]$	$M \in (1.05, \infty)$
$\tau \in (0, 3]$	1.49%	1.83%	1.69%
$\tau \in (3, 9]$	0.88%	1.18%	1.05%
$\tau \in (9, \infty)$	0.90%	0.98%	1.09%

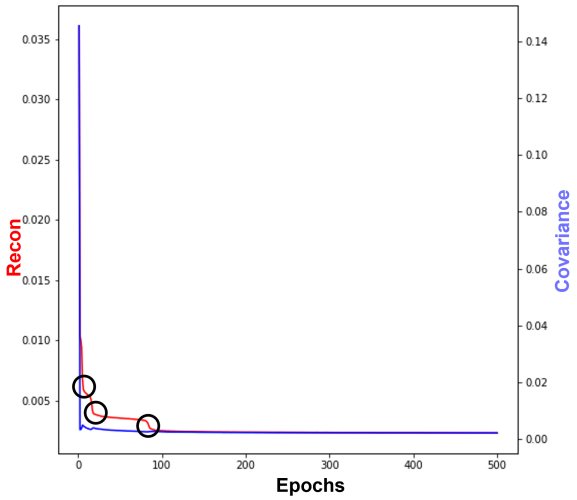
but not identical to the market, and the similarity is within a reasonable range (e.g. bid/offer spread), so that the synthetic surfaces are considered useful. In order to obtain a measure of usefulness, we collected 9082 market quotes for options on European stocks submitted by front office traders in September 2021. Among these quotes, bid and ask quotes are roughly equally distributed. We calculate the implied volatility value for every submitted quote using the quote price and the reverse of equation 2.18, and obtain the implied volatility values for bid and ask quotes separately. We average the distances between bid and ask implied volatility values for different ranges of term and moneyness, and we come up with the evaluation threshold as listed in Table 5.1. The table needs to be read in the following way. If, for example, the absolute difference between the market implied volatility and generated implied volatility $|\sigma_{3,0.9} - \sigma'_{3,0.9}|$ is smaller than 0.0149, then we consider the generated $\sigma'_{3,0.9}$ as a satisfactory implied volatility point. We calculate the percentage of satisfactory points within all the points on all the surfaces we generate, and call this the *satisfaction rate*. In other words, if the generated implied volatility is within the range of bid offer spread from the market value, we consider it as satisfactory.



(a) Classical Variational Auto-encoder (3 Latent)



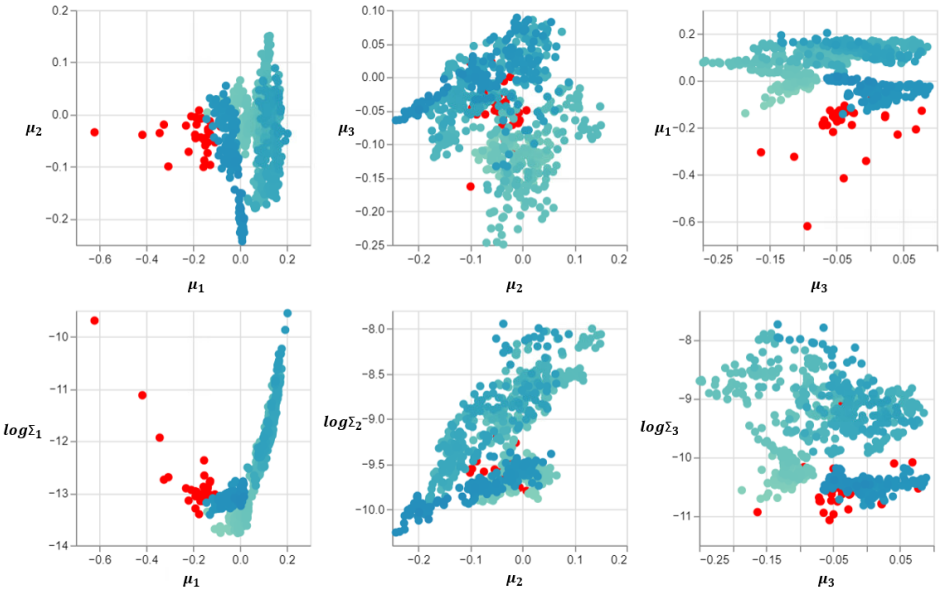
(b) PCA Variational Auto-encoder (3 Latent)



(c) PCA Variational Auto-encoder (4 Latent)

Figure 5.1: Training of Different Variational Auto-Encoders

Figure 5.2: Encoded Latent Space for STOXX50



5.4 Encoding Implied Volatility Surface

In this section, we are going to compare the training process of a PCA variational auto-encoder with a classic variational auto-encoder model, and discuss our discoveries from the trajectories of the training losses. Once the model is trained, we can obtain the plots for the encoded values of each implied volatility surfaces. As our model is a PCA variational auto-encoder in which input data are encoded into distributions, we are going to plot the mean μ and logarithm of the variance Σ.

5.4.1 Training the PCA Variational Auto-encoder

As mentioned in Section 5.1, the PCA variational auto-encoder model incorporates covariance measurements for each pair of latent dimensions in the loss function. By restricting the value of covariance during model training, independent features of volatility surfaces are embedded into latent dimensions. This modification not only largely improves practicabil-

ity and interpretability of auto-encoder for surface related applications, but also significantly changes the trajectory for the reconstruction losses during model training.

In this section, we focus on the model training process. It is undoubtedly important to understand what is exactly learnt by the neural network during the training, and this will be a solid foundation for various volatility surface generation applications to be discussed in Section 5.5.

There are three plots in Figure 5.1, which represent the evolution of reconstruction loss L_{recon} in (5.2) and covariance loss L_{cov} in (5.1) during the training of different variational auto-encoder models. The plots are for a classic variational auto-encoder model, a PCA variational auto-encoder model with three latent dimensions and a PCA variational auto-encoder with four latent dimensions, respectively. The x-axis contains the number of training epochs, whereas the y-axis measures the loss values.

Figure 5.1a shows the training process of a classic variational auto-encoder model ($i.e., \lambda_{cov} = 0$) for implied volatility surfaces. We can clearly observe three turning points where the reconstruction loss is significantly reduced; they are at epoch number 2, 7 and 108. Every time the reconstruction loss has a large reduction, the covariance loss is reduced as well, but it quickly grows afterwards. The final value of covariance loss is roughly 0.468.

The training process of a PCA variational auto-encoder (with $\lambda_{cov} = 0.1$) is shown in Figure 5.1b. It is evident that by adding a covariance constraint to the loss function, the training epochs needed to reach the minimum reconstruction error is smaller (reduced from 108 to 35 epochs). The explanation is that since the covariance is limited, the information encoded in the first two latent dimensions are largely reduced so that there is more information to be captured by the third dimension, and the model can quickly learn the feature. On the contrary, for the case of the classic variational auto-encoder, there is little information left for

the third dimension, so the model struggles to learn the marginal information. It is important to note that in Figure 5.1b we show then training until 130 epochs for comparison with Figure 5.1a. The actual model we use for our experiments is trained after 40 epochs to avoid over-fitting.

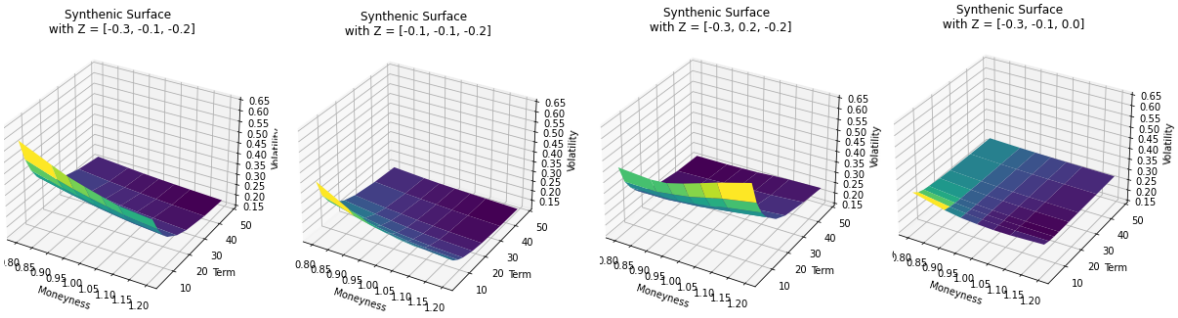
Bergron et al. argue that implied volatility surfaces can be captured by variational auto-encoder using as few as two latent dimensions [15]. From our experiments of training PCA variational auto-encoder with 2 latent dimensions, we found that even though the reconstruction (MAE) error is small after two dimensions are learnt, the covariance loss is not small enough. More importantly, we did not observe the independence of the latent numbers when we build the interactive generation tool, which will be presented in Section 5.5.1.

We want to explore further the training process by adding one more latent dimension into the PCA variational auto-encoder model; the training process is shown in Figure 5.1c. We keep training the model until 500 epochs and cannot observe any further significant reduction in the reconstruction loss. This indicates that there is no further independent feature to be modelled by the PCA variational auto-encoder model. From these observations, we can confirm that three latent dimensions are the only reasonable choice for encoding implied volatility surfaces with PCA variational auto-encoder models. Fewer dimensions will not promote independent features encoded latent dimensions, and adding further dimensions will not encode any extra meaningful features; this will also be emphasised in Section 5.5.1.

5.4.2 Encoded Latent Space

After a PCA variational auto-encoder is trained, we can generate the latent encoding of our training data and examine how the model transfers the implied volatility surfaces into the latent space.

Figure 5.3: Generated Synthetic Surfaces



We plot the values of μ and $log\Sigma$ for all the STOXX50 index implied volatility surfaces, see Figure 5.2. We marked all the surfaces for the trading days in March 2020 and April 2020 as stress scenarios and coloured them in red, as the COVID-19 pandemic caused much turbulence in the market during that period.

In Figure 5.2, we first observe from the top graphs that μ_1, μ_2 and μ_3 are independent to each other, which is the unique characteristic for PCA variational auto-encoders. If the model is a classic variational auto-encoder, there will be some degree of linear relationship between them. We also discover that the stress scenarios for March and April 2020 are caused by abnormally small values of μ_1, which also means that Z_1 is small, as Z_1 is sampled from a normal distribution of mean μ_1. The stress scenario is not much related to the second or third latent dimensions.

In addition, from the bottom left graph in Figure 5.2, we found that when μ_1 is approximately -0.1, Σ_1 is minimum. As μ_1 moves away from -0.1, Σ_1 increases rapidly. From subsequent analysis, which will be discussed in Section 5.5.1, we know that the first dimension controls the overall volatility level of a surface. It makes sense that extreme cases are more volatile in the market.

5.5 Generating Synthetic Surface

We are going to show several important applications of our PCA variational auto-encoder model. The importance and difficulties of these tasks have already been discussed in Section 5.1. We will compare our results with [15] whenever applicable.

5.5.1 Scenario-Based Generation

We mentioned several times that PCA variation auto-encoder is able to encode difference features of implied volatility surfaces independently into three latent numbers. We are going to present how visually this is going to influence the shape of generated synthetic surfaces. There are four synthetic volatility surfaces in Figure 5.3, together with the values of latent vector Z which is used to generate each of these surfaces. The first surface on the left is a reference surface, whereas others are generated by changing only one latent values of the reference surface.

From Figure 5.3, we can discover that Z_1 only controls the overall volatility level, Z_2 only changes the skewness and Z_3 affects only on the term structure of a volatility surface. Therefore, we have a powerful method to generate synthetic surfaces for any specific description of the scenario that is requested. For example, if we need a surface of overall high volatility level, small difference of volatility among moneyness and large spread between short and long term, we can use small values for Z_1 and Z_3 and a large value of Z_2 to generate such a surface. This is not possible with the classical variation auto-encoder model.

Table 5.2: Volatility Surface Extrapolation with Classic VAE Model

Stock	MAE (Known Points)	MAE (Unknown Points)	Satisfaction
DPWGn.DE	0.0044	0.0211	0.6571
DTEGn.DE	0.0154	0.0297	0.5161
LVMH.PA	0.0029	0.0159	0.6857
MUVGn.DE	0.0049	0.0288	0.5366
SGEF.PA	0.0040	0.0232	0.5098
SIEGn.DE	0.0065	0.0299	0.6268
Average	0.0063	0.0248	0.5887

Table 5.3: Volatility Surface Extrapolation with PCA VAE Model

Stock	MAE (Known Points)	MAE (Unknown Points)	Satisfaction
DPWGn.DE	0.0066	0.0259	0.8143
DTEGn.DE	0.0051	0.0202	0.6598
LVMH.PA	0.0032	0.0117	0.7411
MUVGn.DE	0.0070	0.0336	0.6643
SGEF.PA	0.0236	0.0600	0.4902
SIEGn.DE	0.0165	0.0290	0.7491
Average	0.0103	0.0301	0.6865

5.5.2 Implied Volatility Extrapolation

We have briefly introduced the volatility extrapolation problem in Section 5.1. The task is to predict (extrapolate) the full implied volatility surface from only a subset of points on the same surface.

The extrapolation challenge can be interpreted as a two-step problem. The first step is to make sure that the encoding vector calculated from the known subset of a surface contains enough information to extrapolate the unknown points. This depends on what model is used to generate the encoded latent space. The second step is to use an optimisation method to efficiently find the points on the latent space which represent the known subset of a surface.

Bergeron at el. suggested to use classic variational auto-encoder to encode the latent space and the L-BFGS algorithm to find the optimal latent encoding vector which minimises the difference on the known subset of a implied volatility surface [15]. To to generate the unknown subset of the surface, they provide this vector to the pre-trained decoder neural network.

In our dataset, each implied volatility surface has eight terms (3, 6, 9, 12, 18, 24, 36, 48 months) and 7 moneyness settings (0.80, 0.90, 0.95, 1.00, 1.05, 1.10, 1.20). For our extrapolation experiments, we assume the known subset of a volatility surface is of short term (month value of 3, 6, 9 and 12) and close to being at-the-money (moneyness value of 0.95, 1.00 and 1.05), which consists of 12 points. The remaining 44 points are to be extrapolated, and we compare the full generated surface with the full true surface to evaluate the performance of the extrapolation methods using the satisfaction criteria introduced in Section 5.3.

We found that for the extrapolation task, PCA variational auto-encoder model is a better

solution to produce the encoded latent space. In Section 5.5.1, we showed that the model captures the correlation between short- and long-term volatility through latent dimension Z_3, which indicates that the latent vector found from only short-term information also contain the relevant shape information for the long-term. Because volatility surfaces have to satisfy the no-arbitrage conditions, this connection is pretty stable across stocks and time-horizons. This is also the case for Z_2, which presents the volatility skewness. This is an important advantage of PCA variational auto-encoder, which leads to better extrapolation performances, as shown in Table 5.2 and Table 5.3.

We calculate the mean absolute errors on the known subset of surfaces (12 points) and unknown subset (44 points) separately in Table 5.2 and Table 5.3, as well as the satisfaction rates among all the 56 points on the surfaces. We can observe that even though both MAE errors are higher for PCA variational auto-encoder, the satisfaction rate achieved is 10% higher. This indicates that PCA variational auto-encoder is encoding the shape of surfaces instead of individual values of volatility. We also found that the differences of MAEs between two subsets are smaller, which indicates that our model is better to infer the long-term volatility from the short-term volatility as discussed above. However, we think L-BFGS is not a perfect method for optimising on the encoded space because of the level of over-fitting; finding better solutions could be the object of further research.

5.5.3 Stock Specific Generation

The third practical question we want to solve is to infer the implied volatility surfaces for a particular stock from the stock index, by modelling their historical relationships on the latent space produced by a PCA variational auto-encoder model.

For every trading day, we have 39 volatility surfaces, which represent 38 stocks and 1

stock index (STOXX50). We encode these volatility surfaces using a trained PCA variational auto-encoder, and plot for every stock and trading day, its correspondent first latent number (Z_1) of the stock volatility and STOXX50 volatility surfaces. The plot is shown in Figure 5.4.

Every point plotted in Figure 5.4 represents a particular stock on a single trading day. Different colours represent different stocks; we also plot the line $y = x$ as a reference.

We can see From figure 5.4 that there is a linear relationship between Z_1 of each stock and Z_1 of STOXX50. This relationship exists for all the stocks we plotted in the figure. If we draw lines to models these relationships, we see the lines should have slopes close to one, which indicates if the index volatility surface shifts, the stock volatility surface moves roughly by the same amount. We also notice that most stocks have an intercept smaller than one, meaning that Z_1 of stocks are generally smaller than Z_1 of STOXX50. If we refer to volatility surface generation mentioned in Section 5.5.1, this should reflect that the volatility surface of a single stock is usually higher than the surface of the STOXX50 index on the same day, a known stylised fact of the market. Similar linear relationships hold true for Z_2 and Z_3 as well.

Based on these observations, we use linear regression models to capture the relationship between a particular stock and the STOXX50 index in the encoded space. For every stock on a trading day, we train three linear regression models to predict its three latent numbers separately. The independent variables for linear regression models are the latent number of STOXX50 index and the stock's long-term price volatility. Every linear regression model has two independent variables. The models are trained in a moving windows manner, as illustrated in Figure 5.5.

For every trading day in the test period, we use three linear regression models to obtain the latent encoding of the implied volatility surface, which represents , and put the encoding vector into our pre-trained PCA variational auto-encoder. We then compare the generated

Figure 5.4: Z_1 of single stocks and STOXX50

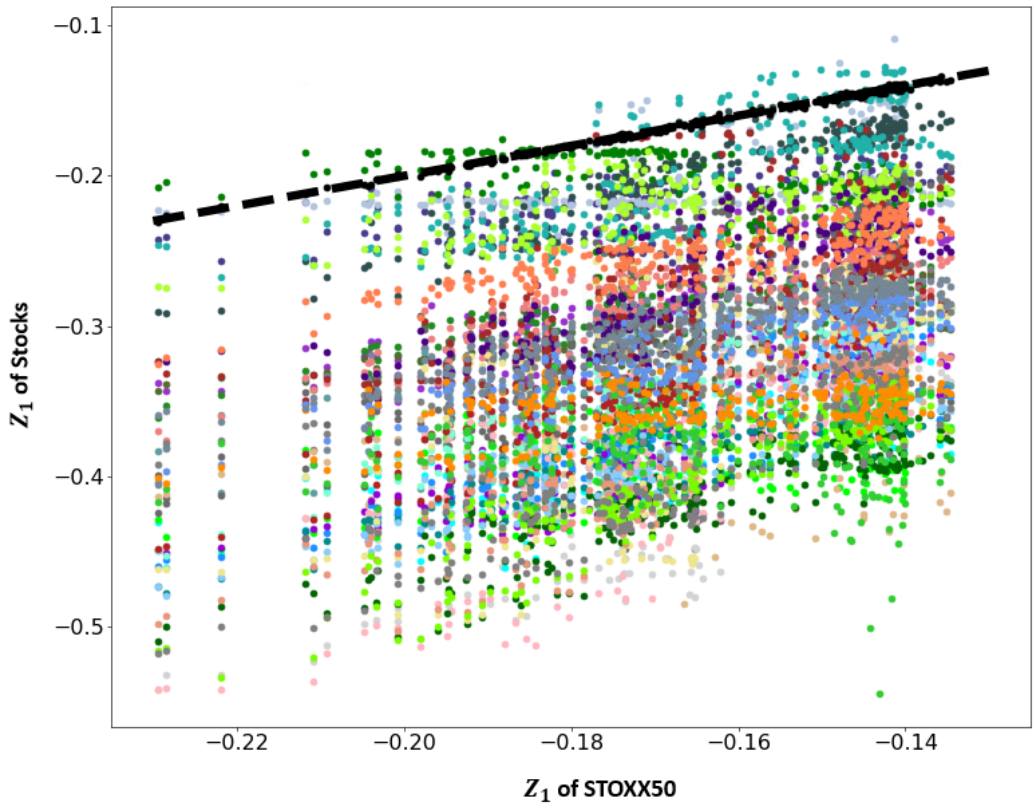


Figure 5.5: Z1 of MUVGn.DE and STOXX50

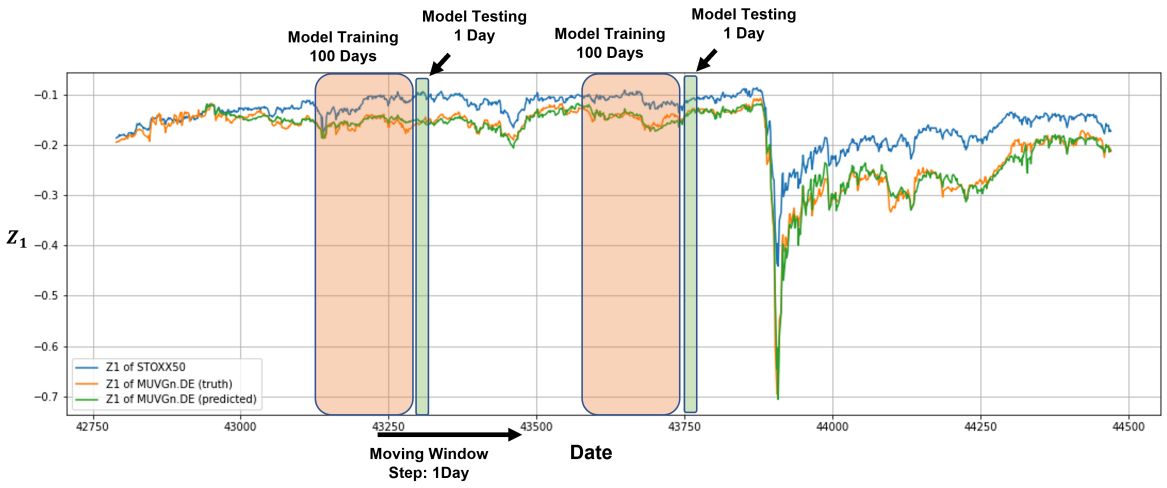


Table 5.4: Predict Stock Volatility Surface Using STOXX50

Stock	Z_0 Error	Z_1 Error	Z_2 Error	Satisfaction
DPWGn.DE	0.0072	0.0112	0.0066	0.7611
DTEGn.DE	0.0088	0.0098	0.0052	0.6981
LVMH.PA	0.0080	0.0086	0.0062	0.8672
MUVGn.DE	0.0086	0.0092	0.0047	0.7131
SGEF.PA	0.0096	0.0090	0.0081	0.6984
SIEGn.DE	0.0069	0.0086	0.0050	0.8383
Average	0.0082	0.0094	0.0060	0.7627

surface with the true surface in our dataset and obtain the results shown in Table 5.4. The errors in Table 5.4 are calculated as the MAE between the true Z value and the predicted one. We see the average satisfaction rate is 76%, which indicates most predicted implied volatility values are within the bid offer spread of the market.

5.6 Conclusion

We use a new way to encode implied volatility surfaces into a latent space with a PCA variational auto-encoder neural network. The extra covariance measurement in the loss function for model training ensures independence of the three latent dimensions, which brings significant benefits for different synthetic surface generation applications.

The scenario-based generation process becomes intuitive and interpretable. The three latent dimensions could represent the overall volatility level, the volatility term structure and the volatility skewness separately. The description of a volatility surface could be interpreted

into numerical values of latent encodings directly.

The volatility extrapolation performance is largely improved and over-fitting is reduced, because of the independence of three latent dimensions. Comparing with classic VAE model, PCA VAE model significantly reduces the differences on mean absolute error for the known area of a volatility surface and the unknown area of the surface.

We also developed a novel solution to infer a single stock volatility surface from the stock index volatility surface, based on our observations that first latent encoding of a stock implied volatility surfaces and its relevant stock index has a linear relationship on every trading day. We used a dynamic linear regression model to present this relationship, and adjust the parameters according to the market environment movements. The prediction processing is efficient, and as far as we know, it is the first model to prediction stock volatility surfaces directly from index volatility surfaces.

Chapter 6

Conclusion

This chapter is the conclusion chapter for this thesis. It provides an overall summary of the research work and points out the achievements and contributions. It also suggests further research directions for applying deep learning for trading and hedging in the financial markets.

6.1 Summary

As indicated by its title, this thesis is focused on applying different types of deep neural network models to the financial markets to tackle real-life problems associated with trading and hedging financial assets. By studying a wide range of pieces of literature about financial mathematics, we understand that there are many sophisticated nonlinear relationships in the financial markets for the foreign exchange rates, stock prices and derivative contracts. Conventionally, people use mathematical equations to represent these relationships and help market participants set up trading and hedging strategies. But those conventional methods have many limitations. Some cannot utilise sufficient data generated from the market, some

do not provide satisfactory performances, and some are based on idealised assumptions. We also learnt, from many influential neural network models that emerged recently from computer vision and natural language processing files, that deep neural network models are universal approximators of nonlinear relationships. It has been successfully applied to capture relationships for different types of information. For example, the relationship between one image and its categorical label or the relationship between one sentence and its translation in another language. Therefore, We first ask if neural networks could also capture relationships inherited in the nature of financial assets. In addition to that, standing in the shoes of financial people, we know that the financial industry is fundamentally different to the computer science industry, where human knowledge has much more in-depth involvement in day-to-day decision-making, it is not only to increase efficiencies and accuracies but also to benefit the interpretability of the methods, which is required by the highly regulated nature of the industry. ‘

Based on these considerations, we carry out three studies in this thesis to understand the capability of neural network models to process data from financial markets. These signals are considered much noisier than other areas where neural network models have already been well studied. We also explored combining neural network models with partial financial knowledge and requirements that investors or regulators specify to provide a more insightful data analysis so that the whole pipeline is more suitable for practical use in dynamic financial markets.

The first study is presented in chapter 3. We collected high-frequency foreign exchange rates from the market and proposed a pipeline for forecasting short-term price changes in the FX market.

Inspired by several recent models proposed for the stock market, we combine technical

indicators with a convolutional neural network to predict future price movements in the foreign exchange market. Five technical indicators are calculated to filter out noise information from the high-frequency FX data, and they are also arranged to focus on eleven different historical window sizes. We capture market information which is most relevant to short-term price movements. We use a convolutional neural network with a recurrent layer to scan the preprocessed matrix and classify the high-frequency price information into three categories representing three directions of future FX price changes. Our proposed Technical Analysis Neural Network (TANN) model captures the relationship between high-frequency ticks in the past and the price movement direction for the short-term future. Our prediction accuracy is, on average, about 5% higher than the best conventional machine learning we tested, the LDA method. Furthermore, using high-frequency ticks from the previous day for training the neural network and testing the following day gives the best performance, indicating the model could learn short-term market conventions, and it helps predict the market. In general, we show that prediction power exists in high-frequency FX ticks if there is an appropriate method to formulate and process the information. Our results also emphasised that financial time series prediction is possible without the support of typical microstructure for the market, such as limit order books (LOB).

We have built a neural network model to process market information at irregular intervals, and this model can help investors better understand the trend in the FX spot market. We want to explore further the possibility of fitting neural network models closer to a market participant's risk tolerances, so the model could better utilise its capabilities and be more practically useful. We tackle one essential shortcoming of the popular deep hedging pipeline, which is to perform trading of the underlying assets at regular intervals, regardless of the user's risk preference or expectation. We upgrade the default deep hedging pipeline

to include a filter element before the hedging generator in deep hedging. The filter is controlled by the model user's preset risk tolerance, and it decides that trading is only allowed when the price change of the underlying asset exceeds the risk tolerance. This way, when the risk tolerance increases, the average final loss of the 20000 simulations reduces as reduced trading frequency saves trading costs. However, the standard deviation of the final loss increases because only significant price movements are dealt with. By varying the risk tolerance threshold, we could generate a line to illustrate the trade-off between the mean and standard deviation of the final hedging loss. We call it the Efficient Hedging Frontiers (EHF). This work is discussed in chapter 4.

EHF is the tool for a model user to trade off between expected risk and expected return from the hedging strategies generated by the deep hedging model. It incorporates the user's risk appetite into the neural network based pipeline, which brings the popular model one step closer to real-life scenarios. We also use a random forest classifier to forecast the future prices of the simulated paths and instruct the strategy generator to avoid V-shaped movements of share prices. In this way, the overall performance of the deep hedging model could be improved, which means the EHF lines shift to the upper left direction. For a given level of risk (standard deviation of termination losses), the model user could achieve a better return (larger mean of termination losses). We also experiment using GRU layers in the neural network, which give slightly better trading profits than the default fully connected version.

We explored and developed new methods regarding trading and hedging in the financial market for the first two studies. After that, we are going to look at the pricing side.

Implied volatility surfaces are the pricing devices for various financial derivatives, which means implied volatility is the most crucial variable for the BSM option pricing model. We

apply the loss function from a PCA autoencoder model to the classic variation autoencoder to obtain the encoded vectors of implied volatility surfaces of 44 European stocks and a European stock index, in which three independent latent dimensions are obtained. We found that each latent dimension represents a unique characteristic of the overall volatility level, volatility skewness and volatility term structure for an implied volatility surface. These independent latent encodings introduce significant benefits to the three synthetic surface generation tasks, which are most attractive to academics and industries. First, the synthetic surface generation process is much more interpretable, as the descriptions of required implied volatility surfaces could be directly translated to the values of the three latent numbers. The model user would have clear expectations of the generated surface and how the model produced these synthetic surfaces. Second, volatility extrapolation achieves better accuracy. Because one latent number explicitly controls the relationship between the short-term subset and the long-term subset of an implied volatility surface, it is easier to extrapolate long-term from short-term. This is also true for extrapolating out-of-the-money from in-the-money volatilities, which is the skewness of an implied volatility surface. Third, we proposed a novel method to predict volatility surface from a stock index to a specific stock based on their relationships on the independent latent spaced generation by our PCA variational autoencoder model. The method could be beneficial in determining the price of financial derivatives of illiquid stocks where its implied volatility surfaces are not directly observable from the market.

6.2 Contribution

This thesis explores various applications of deep neural network models for trading and hedging financial assets. The major contributions of this thesis are as follows:

1. We propose the Technical Analysis Neural Network (TANN) pipeline for processing high-frequency tick prices from the quote-based foreign exchange market and forecasting foreign exchange rate movements in future 5, 10 and 15 minutes. The method surpasses conventional machine learning methods by a clear margin. Furthermore, the performance of the TANN method indicates noisy financial market data could be utilised by a neural network as long as there is an appropriate way to extract crucial information from the market. There is a relationship between historical ticks and future price movement direction in a short time, and neural networks can model the relationship.

2. We upgraded the default deep hedging pipeline, which could trigger hedging actions based on actual market situations and model users' risk preferences, instead of trading blindly on a regular daily basis. In this way, we establish the relationship between risk (standard deviation of contract termination losses) and return (mean of contract termination losses) using the Efficient Hedging Frontiers. The frontiers could assist the model user in deciding the optimal hedging risk tolerance level, which brings deep hedging one step closer to real-life scenarios.

3. We also demonstrate that using extra anticipations of future price movement directions for the underlying asset, such as price predictions obtained from random forest models, could shift the frontiers to the upper left direction so that the overall profits achieved at the end of an option contract is higher, for a given level of risk tolerance.

4. We create a new three-dimensional encoding space for the implied volatility surfaces of forty-four stocks and a stock index, using a PCA variational autoencoder model. The significance is that three latent dimensions are forced to encode independent fea-

tures of implied volatility surfaces, which are verbally interpretable. The novel encoding space benefits the fundamental tasks of synthetic implied volatility surface generation. The generation process becomes intuitive and explainable. The accuracy of implied volatility extrapolation is increased.

5. We propose a novel method to predict the volatility surfaces of a typical stock from the stock index surfaces based on our observation of the relationship between the implied volatility surfaces of stocks and the stock index on the latent encoding space. The method utilised a regression model and the decoder of a pre-trained autoencoder neural network. The novel solution provides benefits for pricing derivatives of illiquid stocks when their implied volatility surfaces are not observable from the market directly.

6.3 Future work

This thesis discusses several areas where deep learning could be applied for the trading and hedging of financial assets. It proves that even the noisiest high-frequency tick data could be utilised to analyse market trends and predict future prices. It also proposes novel methods to incorporate investors' knowledge and expectation with start-of-art neural network models to make more explainable hedging decisions. The pricing of financial derivatives is also improved with the new encoded space of implied volatility surfaces.

The Technical Analysis Neural Network (TANN) pipeline discussed for predicting short-term future foreign exchange rate movement could first be upgraded by finding possible alternative neural network model structure which generates better results. The proposed model structure is similar to the DeepLOB model for stock price prediction, but there are novel solutions to optimise neural network structures using genetic programming and other

similar methods. There is a possibility that the neural network model structure is related to the data type or data quantity to some degree. Another direction for improvement is to look at the technical indicators, and it is possible using other indicators or even create new indicators to achieve better prediction accuracies. The third direction could be to develop a downstream method for trading the foreign exchange and test the profitability. Good prediction does not guarantee a good profit in trading. There are plenty of other aspects to consider. For example, the capital allocation, the timing to place an order, the trading cost and latency, etc.

We propose using the Efficient Hedging Frontiers (EHF) to understand better the hedging decisions generated by the deep hedging neural network model. It is possible to extend the method to more complex derivatives types. For example, the American option allows the holder to exercise the right anytime before termination instead of only at the termination. It is also possible to consider a portfolio of option contracts at different expiration or different underlying stocks. In this case, when deciding the optimal dates for trading, we have to consider correlations in the portfolio. Not only the risk preference or trading frequency. Other constraints may also exist in reality, for example, the allocation of capital. We could work towards incorporating them into the deep hedging neural network and finding out how these constraints would change the balance between the standard deviation of termination losses and the mean of termination losses.

The newly encoded latent space of implied volatility surfaces has plenty of potential research directions. First, check the non-arbitrage conditions for the generated surfaces and the existence and level of arbitrage values on the encoded latent space. If arbitrage values exist, a modified version of the loss function should force the model to generate arbitrage-free surfaces. The latent encoding values are assumed to have standard normal distributions for the PCA variational autoencoder model, but other distributions may also be tested. The

choices of distribution may be related to the market scenario in general. It is also possible to extend the proposed regression model for stock-specific implied volatility surface generation for the scenario that there are no historical surfaces for the stock at all. We should also try to modify the neural network layers from fully connected to convolutional layers because it may detect other meaningful features which specifically target a specific area in the surface, for example, the long-term out of the money volatilities, which are more critical for some types of financial derivative contracts. Recurrent layers are also a candidate for this neural network model because it may specifically target the term structure of implied volatilities.

Bibliography

[1] Paramjit Kaur, Kewal Krishan, Suresh K Sharma, and Tanuj Kanchan. Facial-recognition algorithms: A literature review. *Medicine, Science and the Law*, 60(2):131–139, 2020.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[4] Abolfazl Zargari Khuzani, Morteza Heidari, and S Ali Shariati. Covid-classifier: An automated machine learning model to assist in the diagnosis of covid-19 infection in chest x-ray images. *Scientific Reports*, 11(1):1–6, 2021.

[5] Omer Berat Sezer and Ahmet Murat Ozbayoglu. Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. *Applied Soft Computing*, 70:525 – 538, 2018.

[6] Ritika Singh and Shashi Srivastava. Stock prediction using deep learning. *Multimedia Tools and Applications*, 76(18):18569–18584, 2017.

[7] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[8] Francesco Costantino, Giulio Di Gravio, and Fabio Nonino. Project selection in project portfolio management: An artificial neural network model based on critical success factors. *International Journal of Project Management*, 33(8):1744–1754, 2015.

[9] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*, 2017.

[10] Eliana Angelini, Giacomo Di Tollo, and Andrea Roli. A neural network approach for credit risk evaluation. *The quarterly review of economics and finance*, 48(4):733–755, 2008.

[11] Vincenzo Pacelli, Michele Azzollini, et al. An artificial neural network approach for credit risk management. *Journal of Intelligent Learning Systems and Applications*, 3(02):103, 2011.

[12] Jingtao Yao and Chew Lim Tan. A case study on using neural networks to perform technical forecasting of forex. *Neurocomputing*, 34(1-4):79–98, 2000.

[13] An-Sing Chen and Mark T Leung. Regression neural network for error correction in foreign exchange forecasting and trading. *Computers & Operations Research*, 31(7):1049–1068, 2004.

[14] Ashok K Nag and Amit Mitra. Forecasting daily foreign exchange rates using genetically optimized neural networks. *Journal of Forecasting*, 21(7):501–511, 2002.

[15] Maxime Bergeron, Nicholas Fung, John Hull, Zissis Poulos, and Andreas Veneris. Variational autoencoders: A hands-off approach to volatility. *The Journal of Financial Data Science*, 4(2):125–138, 2022.

[16] Saïd Ladjal, Alasdair Newson, and Chi-Hieu Pham. A pca-like autoencoder. *arXiv preprint arXiv:1904.01277*, 2019.

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

[18] Xiaotao Zhang, Ziqiao Wang, Jing Hao, and Feng He. Price limit and stock market quality: Evidence from a quasi-natural experiment in the chinese stock market. *Pacific-Basin Finance Journal*, page 101778, 2022.

[19] Otc foreign exchange turnover in april 2022. `https://www.bis.org/statistics/rpfx22_fx.htm`. Accessed: 2023-07-20.

[20] Ioannis Souropanis. *Essays on exchange rate forecasting*. PhD thesis, University of Kent, 4 2019. Available at `https://kar.kent.ac.uk/73470/`.

[21] Martin D Gould, Mason A Porter, Stacy Williams, Mark McDonald, Daniel J Fenn, and Sam D Howison. Limit order books. *Quantitative Finance*, 13(11):1709–1742, 2013.

[22] Angelo Ranaldo. Order aggressiveness in limit order book markets. *Journal of Financial Markets*, 7(1):53–74, 2004.

[23] Francesco Rundo. Deep lstm with reinforcement learning layer for financial trend prediction in fx high frequency trading systems. *Applied Sciences*, 9(20), 2019.

[24] Md Samsul Alam, Syed Jawad Hussain Shahzad, and Román Ferrer. Causal flows between oil and forex markets using high-frequency data: Asymmetries from good and bad volatility. *Energy Economics*, 84:104513, 2019.

[25] Yi-Chieh Kao, Hung-An Chen, and Hsi-Pin Ma. An fpga-based high-frequency trading system for 10 gigabit ethernet with a latency of 433 ns. In *2022 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–4. IEEE, 2022.

[26] Martin Martens and Jason Zein. Predicting financial volatility: High-frequency time-series forecasts vis-à-vis implied volatility. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 24(11):1005–1028, 2004.

[27] Abdalla Kablan and Wing Lon Ng. Intraday high-frequency fx trading with adaptive neuro-fuzzy inference systems. *International Journal of Financial Markets and Derivatives*, 2(1-2):68–87, 2011.

[28] Taufiq Choudhry, Frank McGroarty, Ke Peng, and Shiyun Wang. High-frequency exchange-rate prediction with an artificial neural network. *Intelligent Systems in Accounting, Finance and Management*, 19(3):170–178, 2012.

[29] S Villa and Fabio Stella. A continuous time bayesian network classifier for intraday fx prediction. *Quantitative Finance*, 14(12):2079–2092, 2014.

[30] Burton G. Malkiel. The efficient market hypothesis and its critics. *Journal of Economic Perspectives*, 17(1):59–82, 3 2003.

[31] Sunil Poshakwale. Evidence on weak form efficiency and day of the week effect in the indian stock market. *Finance India*, 10(3):605–616, 1996.

[32] Raymond M Leuthold and Peter A Hartmann. A semi-strong form evaluation of the efficiency of the hog futures market. *American Journal of Agricultural Economics*, 61(3):482–489, 1979.

[33] Salman Syed Ali, Khalid Mustafa, and Asad Zaman. Testing semi-strong form efficiency of stock market [with comments]. *The Pakistan Development Review*, pages 651–674, 2001.

[34] Joseph E Finnerty. Insiders and market efficiency. *The journal of finance*, 31(4):1141–1148, 1976.

[35] Steven B Achelis. Technical analysis from a to z, 2001.

[36] Andrew W Lo and A Craig MacKinlay. Stock market prices do not follow random walks: Evidence from a simple specification test. *The review of financial studies*, 1(1):41–66, 1988.

[37] Alireza Sadeghi, Amir Daneshvar, and Mahdi Madanchi Zaj. Combined ensemble multi-class svm and fuzzy nsga-ii for trend forecasting and trading in forex markets. *Expert Systems with Applications*, 185:115566, 2021.

[38] He Ni and Hujun Yin. Exchange rate prediction using hybrid neural networks and trading indicators. *Neurocomputing*, 72(13):2815 – 2823, 2009.

[39] Saeede Anbaee Farimani, Majid Vafaei Jahan, Amin Milani Fard, and Seyed Reza Kamel Tabbakh. Investigating the informativeness of technical indicators and news sentiment in financial market price prediction. *Knowledge-Based Systems*, 247:108742, 2022.

[40] John C Hull. *Options futures and other derivatives*. Pearson Education India, 2003.

[41] Ali Hirsa and Salih N Neftci. *An introduction to the mathematics of financial derivatives*. Academic press, 2013.

[42] Yue-Kuen Kwok. Introduction to derivative instruments. *Mathematical Models of Financial Derivatives*, pages 1–34, 2008.

[43] Robert Brown. Xxvii. a brief account of microscopical observations made in the months of june, july and august 1827, on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies. *The philosophical magazine*, 4(21):161–173, 1828.

[44] Krishna Reddy and Vaughan Clinton. Simulating stock prices using geometric brownian motion: Evidence from australian companies. *Australasian Accounting, Business and Finance Journal*, 10(3):23–47, 2016.

[45] Kiyosi Itô. 109. stochastic integral. *Proceedings of the Imperial Academy*, 20(8):519–524, 1944.

[46] Rahul R Marathe and Sarah M Ryan. On the validity of the geometric brownian motion assumption. *The Engineering Economist*, 50(2):159–192, 2005.

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

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

[49] Xisheng Yu and Xiaoke Xie. On derivations of black-scholes greek letters. *Research Journal of Finance and Accounting*, 4(6):80–85, 2013.

[50] Viktor Stojkoski, Trifce Sandev, Lasko Basnarkov, Ljupco Kocarev, and Ralf Metzler. Generalised geometric brownian motion: Theory and applications to option pricing. *Entropy*, 22(12):1432, 2020.

[51] Svetlana Boyarchenko and Sergei Z Levendorskii. *Non-Gaussian Merton-Black-Scholes Theory*, volume 9. World Scientific, 2002.

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

[53] Ole E Barndorff-Nielsen, Mikko S Pakkanen, and Jürgen Schmiegel. Assessing relative volatility/intermittency/energy dissipation. *Electronic Journal of Statistics*, 8(2):1996–2021, 2014.

[54] Ole E Barndorff-Nielsen, Fred Espen Benth, and Almut ED Veraart. Modelling energy spot prices by volatility modulated lévy-driven volterra processes. *Bernoulli*, 19(3):803–845, 2013.

[55] Jim Gatheral, Thibault Jaisson, and Mathieu Rosenbaum. Volatility is rough. *Quantitative Finance*, 18(6):933–949, 2018.

[56] Christian Bayer, Peter Friz, and Jim Gatheral. Pricing under rough volatility. *Quantitative Finance*, 16(6):887–904, 2016.

[57] Mikkel Bennedsen, Asger Lunde, and Mikko S Pakkanen. Hybrid scheme for brownian semistationary processes. *Finance and Stochastics*, 21(4):931–965, 2017.

[58] Taras Bodnar and Wolfgang Schmid. Econometrical analysis of the sample efficient frontier. *The European journal of finance*, 15(3):317–335, 2009.

[59] Mark Rubinstein. Markowitz's" portfolio selection": A fifty-year retrospective. *The Journal of finance*, 57(3):1041–1045, 2002.

[60] Robert C Merton. An analytic derivation of the efficient portfolio frontier. *Journal of financial and quantitative analysis*, 7(4):1851–1872, 1972.

[61] Henri Berestycki, Jérôme Busca, and Igor Florent. Computing the implied volatility in stochastic volatility models. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(10):1352–1373, 2004.

[62] Jin E Zhang and Yi Xiang. The implied volatility smirk. *Quantitative Finance*, 8(3):263–284, 2008.

[63] Jyh-Woei Lin. Artificial neural network related to biological neuron network: a review. *Advanced Studies in Medical Sciences*, 5(1):55–62, 2017.

[64] Tomasz Szandała. Review and comparison of commonly used activation functions for deep neural networks. In *Bio-inspired neurocomputing*, pages 203–224. Springer, 2021.

[65] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

[66] Dabal Pedamonti. Comparison of non-linear activation functions for deep neural networks on mnist classification task. *arXiv preprint arXiv:1804.02763*, 2018.

[67] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[68] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for neural networks for image processing. *arXiv preprint arXiv:1511.08861*, 2015.

[69] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[70] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[71] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[72] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[73] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE international conference on neural networks*, pages 586–591. IEEE, 1993.

[74] Eric W Weisstein. Convolution. *https://mathworld. wolfram. com/*, 2003.

[75] Ali N Akansu, Richard A Haddad, and Paul A Haddad. *Multiresolution signal decomposition: transforms, subbands, and wavelets*. Academic press, 2001.

[76] William D Warner and Cyril Leung. Ofdm/fm frame synchronization for mobile radio data communication. *IEEE Transactions on vehicular technology*, 42(3):302–313, 1993.

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

[78] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[79] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.

[80] Mohammad Sabokrou, Mohsen Fayyaz, Mahmood Fathy, Zahra. Moayed, and Reinhard Klette. Deep-anomaly: Fully convolutional neural network for fast anomaly detection in crowded scenes. *Computer Vision and Image Understanding*, 172:88 – 97, 2018.

[81] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115—118, 2 2017.

[82] Hongwen Yan, Qingliang Cui, and Zhenyu Liu. Pig face identification based on improved alexnet model. *INMATEH-Agricultural Engineering*, 2020.

[83] Hao Wu and Jinsong Zhao. Deep convolutional neural network model based chemical process fault diagnosis. *Computers & chemical engineering*, 115:185–197, 2018.

[84] Sid Ghoshal and Stephen Roberts. Thresholded convnet ensembles: neural networks for technical forecasting. *Neural Computing and Applications*, 04 2020.

[85] Srijan Sood, Zhen Zeng, Naftali Cohen, Tucker Balch, and Manuela Veloso. Visual time series forecasting: an image-driven approach. In *Proceedings of the Second ACM International Conference on AI in Finance*, pages 1–9, 2021.

[86] Jiasheng Cao and Jinghan Wang. Stock price forecasting model based on modified convolution neural network and financial time series analysis. *International Journal of Communication Systems*, 32(12):e3987, 2019.

[87] Lounnapha Sayavong, Zhongdong Wu, and Sookasame Chalita. Research on stock price prediction method based on convolutional neural network. In *2019 international conference on virtual reality and intelligent systems (ICVRIS)*, pages 173–176. IEEE, 2019.

[88] Can Yang, Junjie Zhai, Guihua Tao, et al. Deep learning for price movement prediction using convolutional neural network and long short-term memory. *Mathematical Problems in Engineering*, 2020, 2020.

[89] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, 2019.

[90] Omer Berat Sezer and Ahmet Murat Ozbayoglu. Financial trading model with stock bar chart image time series with deep convolutional neural networks. *arXiv preprint arXiv:1903.04610*, 2019.

[91] Xiurui Hou, Kai Wang, Cheng Zhong, and Zhi Wei. St-trader: A spatial-temporal deep neural network for modeling stock market movement. *IEEE/CAA Journal of Automatica Sinica*, 8(5):1015–1024, 2021.

[92] S Kumar Chandar. Convolutional neural network for stock trading using technical indicators. *Automated Software Engineering*, 29:1–14, 2022.

[93] Lina Ni, Yujie Li, Xiao Wang, Jinquan Zhang, Jiguo Yu, and Chengming Qi. Forecasting of forex time series data based on deep learning. *Procedia computer science*, 147:647–652, 2019.

[94] Rian Rassetiadi and Suharjito Suharjito. Foreign exchange prediction based on indices and commodities price using convolutional neural network. *Indonesian Journal of Electrical Engineering and Computer Science*, 18(1):494–501, 2019.

[95] Chen Liu, Weiyan Hou, and Deyin Liu. Foreign exchange rates forecasting with convolutional neural network. *Neural Processing Letters*, 46:1095–1119, 2017.

[96] Lkhagvadorj Munkhdalai, Tsendsuren Munkhdalai, Kwang Ho Park, Heon Gyu Lee, Meijing Li, and Keun Ho Ryu. Mixture of activation functions with extended min-max normalization for forex market prediction. *IEEE Access*, 7:183680–183691, 2019.

[97] Kevin Chantona, Ronsen Purba, and Arwin Halim. News sentiment analysis in forex trading using r-cnn on deep recurrent q-network. In *2020 Fifth International Conference on Informatics and Computing (ICIC)*, pages 1–7. IEEE, 2020.

[98] Che-Yu Lee and Von-Wun Soo. Predict stock price with financial news based on recurrent convolutional neural networks. In *2017 conference on technologies and applications of artificial intelligence (TAAI)*, pages 160–165. IEEE, 2017.

[99] Joy Iong-Zong Chen and Kong-Long Lai. Deep convolution neural network model for credit-card fraud detection and alert. *Journal of Artificial Intelligence and Capsule Networks*, 3(2):101–112, 2021.

[100] Kang Fu, Dawei Cheng, Yi Tu, and Liqing Zhang. Credit card fraud detection using convolutional neural networks. In *Neural Information Processing: 23rd International Conference, ICONIP 2016, Kyoto, Japan, October 16–21, 2016, Proceedings, Part III 23*, pages 483–490. Springer, 2016.

[101] Abolfazl Mehbodniya, Izhar Alam, Sagar Pande, Rahul Neware, Kantilal Pitambar Rane, Mohammad Shabaz, and Mangena Venu Madhavan. Financial fraud detection in healthcare using machine learning and deep learning techniques. *Security and Communication Networks*, 2021:1–8, 2021.

[102] Tadaaki Hosaka. Bankruptcy prediction using imaged financial ratios and convolutional neural networks. *Expert systems with applications*, 117:287–299, 2019.

[103] Yilin Ma, Ruizhu Han, and Weizhong Wang. Prediction-based portfolio optimization models using deep neural networks. *Ieee Access*, 8:115393–115405, 2020.

[104] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deep learning for portfolio optimization. *The Journal of Financial Data Science*, 2020.

[105] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[106] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[107] Roger Grosse. Lecture 15: Exploding and vanishing gradients. *University of Toronto Computer Science*, 2017.

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

[109] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, 10 2014. Association for Computational Linguistics.

[110] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[111] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015.

[112] Zengjian Liu, Ming Yang, Xiaolong Wang, Qingcai Chen, Buzhou Tang, Zhe Wang, and Hua Xu. Entity recognition from clinical texts via recurrent neural network. *BMC medical informatics and decision making*, 17(Suppl 2):67, 7 2017.

[113] Binbin Yan, Memon Aasma, et al. A novel deep learning framework: Prediction and analysis of financial time series using ceemd and lstm. *Expert systems with applications*, 159:113609, 2020.

[114] Yifei Zhang. A better autoencoder for image: Convolutional autoencoder. In *ICONIP17-DCEC*, 2018.

[115] Yu Chen and Mohammed J Zaki. Kate: K-competitive autoencoder for text. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 85–94, 2017.

[116] Cheng-Yuan Liou, Wei-Chen Cheng, Jiun-Wei Liou, and Daw-Ran Liou. Autoencoder for words. *Neurocomputing*, 139:84–96, 2014.

[117] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[118] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

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

[120] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[121] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[122] Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7304–7308. IEEE, 2013.

[123] Justin Sirignano and Rama Cont. Universal features of price formation in financial markets: perspectives from deep learning. *Quantitative Finance*, 19(9):1449–1459, 2019.

[124] Fan Fang, Waichung Chung, Carmine Ventre, Michail Basios, Leslie Kanthan, Lingbo Li, and Fan Wu. Ascertaining price formation in cryptocurrency markets with deeplearning. *CoRR*, abs/2003.00803, 2020.

[125] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[126] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Os-

trovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[127] Yuh-Jong Hu and Shang-Jen Lin. Deep reinforcement learning for optimizing finance portfolio management. In *2019 amity international conference on artificial intelligence (AICAI)*, pages 14–20. IEEE, 2019.

[128] Frensi Zejnullahu, Maurice Moser, and Joerg Osterrieder. Applications of reinforcement learning in finance–trading with a double deep q-network. *arXiv preprint arXiv:2206.14267*, 2022.

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

[130] Johannes Ruf and Weiguan Wang. Hedging with linear regressions and neural networks. *Journal of Business and Economic Statistics*, 2020.

[131] Jay Cao, Jacky Chen, John Hull, and Zissis Poulos. Deep hedging of derivatives using reinforcement learning. *The Journal of Financial Data Science*, 3(1):10–27, 2021.

[132] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[133] Jorma Laaksonen and Erkki Oja. Classification with learning k-nearest neighbors. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 3, pages 1480–1483. IEEE, 1996.

[134] Anthony J Myles, Robert N Feudale, Yang Liu, Nathaniel A Woody, and Steven D Brown. An introduction to decision tree modeling. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 18(6):275–285, 2004.

[135] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[136] Hasna Haifa Zahrah, Siti Saâ, Rita Rismala, et al. The foreign exchange rate prediction using long-short term memory. *International Journal on Information and Communication Technology (IJoICT)*, 6(2):94–105, 2020.

[137] Richard Meese and Kenneth Rogoff. Empirical exchange rate models of the seventies: Do they fit out of sample? *Journal of International Economics*, 14:3–24, 1983.

[138] Robert P. Flood and Andrew K. Rose. Fixing exchange rates a virtual quest for fundamentals. *Journal of Monetary Economics*, 36(1):3 – 37, 1995.

[139] Andrew Lilley, Matteo Maggiori, Brent Neiman, and Jesse Schreger. Exchange rate reconnect. *Review of Economics and Statistics*, 104(4):845–855, 2022.

[140] Oleg Itskhoki and Dmitry Mukhin. Exchange rate disconnect in general equilibrium. *Journal of Political Economy*, 129(8):2183–2232, 2021.

[141] Ramazan Gençay, Michel Dacorogna, Ulrich A Muller, Olivier Pictet, and Richard Olsen. *An introduction to high-frequency finance*. Elsevier, 2001.

[142] James B Glattfelder, Alexandre Dupuis, and Richard B Olsen. Patterns in high-frequency fx data: discovery of 12 empirical scaling laws. *Quantitative Finance*, 11(4):599–614, 2011.

[143] Shengnan Li, Edward PK Tsang, and John O'Hara. Measuring relative volatility in high-frequency data under the directional change approach. *Intelligent Systems in Accounting, Finance and Management*, 29(2):86–102, 2022.

[144] Irene Aldridge. *High-frequency trading: a practical guide to algorithmic strategies and trading systems*, volume 604. John Wiley & Sons, 2013.

[145] Alexis Stenfors and Masayuki Susai. Liquidity withdrawal in the fx spot market: A cross-country study using high-frequency data. *Journal of International Financial Markets, Institutions and Money*, 59:36–57, 2019.

[146] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[147] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[148] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv e-prints*, page arXiv:1312.4400, 12 2013.

[149] Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap

and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

[150] Menzie D. Chinn and Richard A. Meese. Banking on currency forecasts: How predictable is change in money? *Journal of International Economics*, 38(1):161 – 178, 1995.

[151] Xing Huan and Antonio Parbonetti. Financial derivatives and bank risk: evidence from eighteen developed markets. *Accounting and Business Research*, 49(7):847–874, 2019.

[152] Edoardo Vittori, Michele Trapletti, and Marcello Restelli. Option hedging with risk averse reinforcement learning. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8, 2020.

[153] Zuzana Janková. Drawbacks and limitations of black-scholes model for options pricing. *Journal of Financial Studies and Research*, 2018:1–7, 2018.

[154] David S. Bates. Jumps and stochastic volatility: Exchange rate processes implicit in deutsche mark options. *The Review of Financial Studies*, 9(1):69–107, 1996.

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

[156] Ji Hyun Jang, Jisang Yoon, Jungeun Kim, Jinmo Gu, and Ha Young Kim. Deepoption: A novel option pricing framework based on deep learning with fused distilled data from multiple parametric methods. *Information Fusion*, 70:43–59, 2021.

[157] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[158] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, pages 5–39, 2000.

[159] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[160] Luigi Ballabio. Implementing quantlib, 2005.

[161] Leif Andersen and Jesper Andreasen. Jump-diffusion processes: Volatility smile fitting and numerical methods for option pricing. *Review of derivatives research*, 4(3):231–262, 2000.

[162] Cristian Homescu. Implied volatility surface: Construction methodologies and characteristics. *arXiv preprint arXiv:1107.1834*, 2011.

[163] Janne Äijö. Implied volatility term structure linkages between vdax, vsmi and vstoxx volatility indices. *Global Finance Journal*, 18(3):290–302, 2008.

[164] Jay Cao, Jacky Chen, and John Hull. A neural network approach to understanding implied volatility movements. *Quantitative Finance*, 20(9):1405–1413, 2020.

[165] Damien Ackerer, Natasa Tagasovska, and Thibault Vatter. Deep smoothing of the implied volatility surface. *Advances in Neural Information Processing Systems*, 33:11552–11563, 2020.

[166] Yu Zheng, Yongxin Yang, and Bowei Chen. Gated deep neural networks for implied volatility surfaces. *arXiv preprint arXiv:1904.12834*, 7, 2019.