ROBUST FINANCIAL TRADING SYSTEM WITH DEEP Q NETWORK (DQN)

Sutta Sornmayura

A Dissertation Submitted in Partial
Fulfillment of the Requirements for the Degree of
Doctor of Philosophy (Management)
International College,
National Institute of Development Administration
2017

ROBUST FINANCIAL TRADING SYSTEM WITH DEEP Q NETWORK (DQN)
Sutta Sornmayura
International College,

..... Major Advisor
(Associate Professor Vesarach Aumeboonsuke, Ph.D.)

The Examining Committee Approved This Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy (Management).

..... Committee Chairperson
(Assistant Professor Sid Suntrayuth, Ph.D.)

..... Committee
(Associate Professor Vesarach Aumeboonsuke, Ph.D.)

..... Committee
(Assistant Professor Nopphon Tangjitprom, Ph.D.)

..... Dean
(Associate Professor Piboon Puriveth, Ph.D.)

_____/_____/_____

ABSTRACT

Title of Dissertation	ROBUST FINANCIAL TRADING SYSTEM WITH DEEP Q NETWORK (DQN)
Author	Sutta Sornmayura
Degree	Doctor of Philosophy (Management)
Year	2017

Forex trading is one of the most attractive areas in finance. However, developing the profitable trading system is not an easy task because it requires extensive knowledge in several areas such as quantitative analysis, financial skills, and computer programming. Trading system expert, as a human, also bring in their own bias to develop the system. The trading system developers will prefer some markets over others, prefer some indicators over others, and prefer some trading time frame over others. Moreover, developing the trading system will also be prone to data-snooping and look-ahead bias. Developing trading system is the never-ending task and requires numerous experiments with several parameters.

Random walk and EMH theories support the assumption for choosing buy-and-hold as the best alternative when choosing the strategy for trading. However, there are numerous studies which contradict both theories. Those studies support the idea of using technical analysis as a predictive tool to find the hidden profitable pattern in the market. However, technical analysis is also prone to bias of the users as well.

The problem of developing the robust financial trading system is challenging. In terms of developing cost, time and effort. It must be a new method to efficiently develop the trading system. Simultaneously, this method should eliminate all biases from system developers.

The most attractive way to develop the system is to use cutting-edge technology such as artificial intelligence (AI) technology. This new method of developing the trading strategy needs to benchmark with buy-and-hold (Random walk and EMH assumption) and with the trading experts who are commodity trading advisor (CTA).

This study tried to compare the performance of AI to buy-and-hold strategy and performance of AI to the expert trader. The tested markets were Forex (EURUSD, USDJPY) and Gold (XAUUSD) market, data obtaining from Dukascopy Bank SA

Switzerland (15 years data). Both hypotheses were tested with Paired t-Test at the significance level of 0.05. The findings showed that AI could significantly beat buy-and-hold strategy for FOREX in both 2 currency pairs (EURUSD, USDJPY), and AI could also significantly outperform Commodity Trading Advisor (CTA) for trading EURUSD. However, AI could not significantly outperform CTA for USDJPY trading. For Gold (XAUUSD) market, AI could not significantly outperform buy-and-hold and CTA. Limitation, contribution, and further research were also recommended.

ACKNOWLEDGEMENTS

First, I would like to express my sincerest gratitude to my advisor, Associate Professor Dr. Vesarach Aumeboonsuke, for the continuous support of my Ph.D. study and research, for her patience, motivation, enthusiasm, and immense knowledge. On the academic level, she taught me fundamentals of conducting scientific research in the financial area. Under her supervision, I learned how to define a research problem, find a solution to it, and finally publish the results. On a personal level, she inspired me with her hardworking and passionate attitude. To summarize, I am truly grateful for never-ending attention, mercy, and support throughout my time as her advisee.

Besides my advisor, I would like to thank the rest of my dissertation committee members (Assistant Professor Dr. Sid Suntrayuth and Assistant Professor Dr. Nopphon Tangjitprom) for their great support, invaluable advice, and insightful comments.

I would like to thank all my lovely friends and classmates at international college, NIDA and my friends at Assumption University for their support and consideration.

My sincere thanks also go to Ms. Marisa Marsiglietti who always support me and stand beside me during my difficult time working out how to fix the bug in the code and problems with word processing.

Big thank also goes to my brother, Dr. Pattana Sornmayura, who always encourage me to get through my hard time.

Last but not least, I would like to express my deepest gratitude to my parents, Mr. Sompothi Sornmayura and Mrs. Penchan Sornmayura. This dissertation would not have been possible without their warm love, continued patience, and endless support.

Sutta Sornmayura

June 2018

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

1.1 Background

Financial trading is one of the most interesting, challenging, and promising areas in finance. Developing profitable trading systems is a hard-working task for professional traders or trading system developers. However, the payoff for such a dedicated endeavor may yield high rewards. Currently, there are two schools of thought when one wants to develop trading systems, based on the underlying assumption of the value in the long term fundamental factors called fundamental analysis or based on future price movement that possesses predictive power called technical analysis; both are followings:

1. *Fundamental analysis*: the attempt to predict the price of instruments based on fundamental factors such as P/E, profits, EBITDA, EBIT.
2. *Technical analysis*: the attempt to predict the future price of instruments based on historical price, volume, volatility, technical indicator.

The Trading system can be developed purely from technical analysis such as price action, technical indicators, or only fundamental analysis such as critical financial metrics like P/E ratio, P/BV ratio, profit growth, or the hybrid system that combine both technical analysis and fundamental analysis concepts. The Trading system can also be as simple as using historical price data as an indicator or as complex as using AI to make complicated trading decisions.

Several literatures substantiate the idea that technical analysis, if appropriately applied, can be used to develop the profitable system and possess some statistical edges to trade the market. Dourra et al. (2002) had studied the application of technical analysis and fuzzy logic with three technical indicators which were ROC, Stochastic, and support/resistance to study four stocks. The result found to be excellent and

surpassed S&P 500 performance (Dourra & Siy, 2002). Chan et al. (1995) had studied the application of neural network with three technical indicators which were SMA, stochastic, momentum. Chan used the neural network to predict the trading signal before the crowd got into the trade. The result found to be more profitable than using traditional technical signal. This study showed that if the traders could predict the trading signal before the majority found out, they would be able to make money, and the neural network could predict before the crowd (Chan & Teong, 1995). The researcher found that there is some predictive power in the technical analysis that used price and technical indicators as the key to unlock the future price movement and profitable opportunities. Therefore, the purpose of this study is to describe how to use the price and technical indicators as the valuable pieces of information for the computer to uncover the hidden profitable patterns and develop the trading systems.

Currently, the majority of trading systems are automated, also known as algorithmic trading strategies. The systems have been developed based on intensive quantitative analysis including mathematics, statistics, linear algebra, machine learning, artificial intelligence, and even the laws of physics. Developing the tradable algorithmic trading strategies is very challenging because the robust systems need to survive and adapt itself through several of market conditions. However, after carefully researched, no such a trading strategy can be profitable in all market conditions. The market is continuously changing and evolving due to non-linear, stochastic nature of the market. The researcher found that developing the robust trading system need critical features such as adaptive capability and synchronicity with the market. Therefore, the concept of machine learning and artificial intelligence that the computer will learn to trade from the data is very challenging to study.

The research motivation in this paper comes from two domains of knowledge which are artificial intelligence (machine learning) and financial trading. The critical feature for AI is the ability to learn from a significant amount of data to find the hidden patterns. For financial trading, the endeavor is developing the profitable trading system which is currently an inefficient task that needs to be automated.

In artificial intelligence (AI) area, there was a big leap shown by the company named "Deepmind". They showed that the computer could learn how to play the Atari game without human supervision. Later, in March 2016, there was a major

breakthrough when the computer learned how to play the board game go, had first-time beaten human professional go player. The significant development was leading to the search for general intelligence, the computer that performs self-learning process to do any task. All these discoveries intrigued the researcher to explore more about the application of AI or machine learning in the area of financial trading.

In financial trading or quantitative trading areas, trading system developers are required to apply quantitative analysis concept to develop trading strategies. The key idea of quantitative trading strategies is to find the statistical edge in the specific types of the market conditions and apply these developed trading systems in that markets. However, developing trading system process is time-consuming, error-prone, and inefficient. Therefore, the majority of trading systems will be developed by computers instead of humans. Currently, for the past decade, algorithmic trading and high-frequency trading have been growing until becoming majority strategies in the trading market. Most of the trading activities are mainly executed by the computers (Seth, 2015).

1.2 Problem Statement

Developing the trading system is a rigorous task. It is required to have several skills from system developers such as statistics, computer programming, and finance. However, the developed system has no guarantee that performance of the trading system will be similar to the backtesting results. Developing quantitative trading strategy is a time-consuming, inefficient task due to the need to identification of following factors:

- What instruments to trade (stock, ETF, forex, futures, and commodity).
- How many stocks, lots, and contracts should we open for each position (position sizing or money management)?
- What timeframe should we trade such as 1hour, 4 hours, daily, weekly, and monthly?
- What side of the position to take? (go long or go short)
- Position management (what to do when position already opened).
- When to exit?

Making those above decisions are all depend on the constraints of the traders (available capital, risk tolerance, transaction cost, available time, and trading objective).

The traditional approach for trading system development will be the variation of the following process:

1. Define the objective function
2. Decide what to trade and how to trade it
3. Design the trading system
4. Determine the in-sample period
5. Determine the out-of-sample period
6. Decide what to optimize
7. Perform walk forward runs
8. Evaluate out-of-sample results
9. Trade the system
10. Monitor the results

Majority of trading systems belong to the categories of forecasting system. System developers decide to choose inputs and choose the indicators to predict the up and down of the markets. If there is an error the process of parameter, adjustment will take place until the performance is satisfactory. Next step, defining the trading rules with adjusted parameters will follow. Finally, the trading decision will be made.

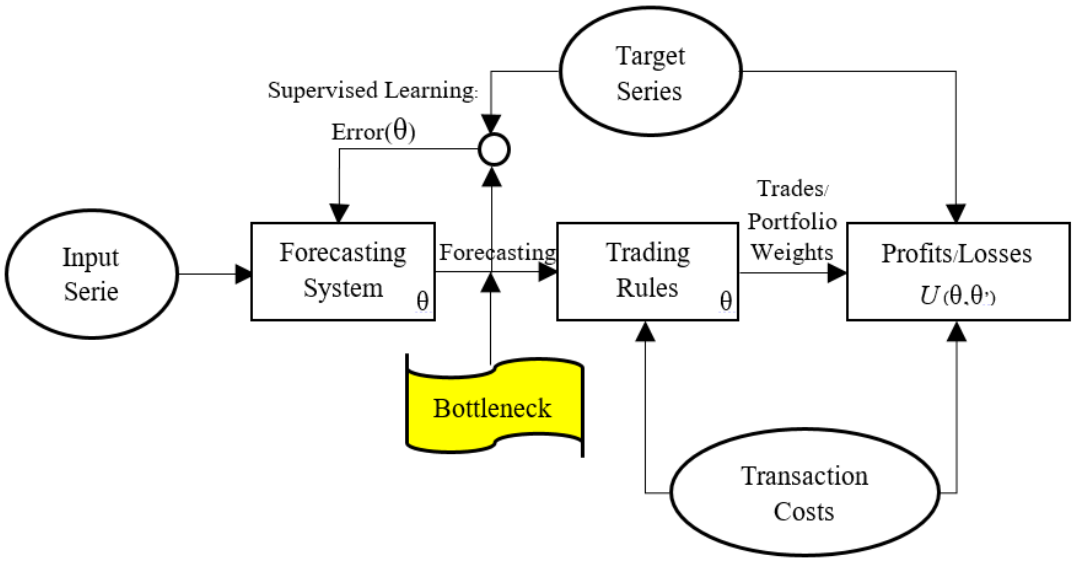


Figure 1.1 Trading System Based on Forecast
Source: Moody (2001)

Moreover, There are several of drawbacks when one developing the trading system such as many biases toward some preferred instruments (i.e., some stock over the others), bias to choose favorite indicators, bias to choose the in-sample period, bias toward optimization. All biases are mostly brought to the development process by the trading system developer. Moreover, there are also some major biases found in backtest period which are followings:

1.1 Data-snooping bias. During the backtesting process, introducing more parameters to gain higher performance would be running the risk of "curve-fitting". When trading lives, the performance will be very much different from the backtesting result.

1.2 Look-Ahead bias. The future data is accidentally included in the backtest of the system. For example, if the backtest is running day 0 to day 30, but backtesting data include the information from day 31 onto the future, this would be called Look-ahead bias.

1.3 Survivorship bias. It could happen when we arbitrarily test only on the instrument or the stocks that we know it had survived to the current period of testing.

A system trader could end up with the trading system that can trade profitably at the specific market condition and be confident to trade live. However, the trading system will fail when the market and trading system both are not in synchronicity as we can see in the following picture that the system performs well in the in-sample period, but fail miserably in the out-of-sample period.

Figure 1.2 Backtest Result vs. Live Trade
Source: Wiecki (2015b)

From above picture, the backtesting result shows uptrend of cumulative return in green color. However, when trade lives, the cumulative return turned out to be very much different from the backtesting result due to several reasons such as the stochastic nature of the market, announcement of the news, central bank intervention, etc. Moreover, some systems perform well both in in-sample and out-of-sample period, but when trade lives it fail as well. The researchers have done extensive studies and found that no any strategy will prosper and perform well through time, no one size fits all for the trading system or another word “no holy grail trading system”. The market changes will fail the system. How do we create the profitable system? The profitable system must be a self-adaptive system that changes according to the market regime.

After careful studies, the researcher found that traditional ways for developing trading system may no longer be efficient. Development processes that take years and months to develop, but no guarantee the result is not acceptable when time factor is critical. Therefore, the researcher put together all concerns for developing the trading system as based on the quest for followings answer:

1. How do we create a profitable system with the shortest time possible?
2. How do we adjust the trading system through the change of market conditions?
3. How do we optimize all parameters regarding each trading system?
4. How do we develop, back-test, validate and production with the shortest time?

Computer not only can execute trading order according to what we program them to do, but also be able to find the above solutions as well. Machine learning is the novel concept that can be applied in financial trading problems. The researcher found that developing computer-generated strategies is much more efficient ways. Computer-generated strategies can be performed by several approaches such as evolutionary algorithm, standard machine learning algorithm such as random forest, support vector machine.

Several types of research in the past mostly attempting to apply machine learning for stock prediction using linear regression and classification. For regression-type research, most research papers tried to forecast and calculate MSE to evaluate the error. This type of study tried to predict the price rather than developing a trading system. For classification-type research, most tried to predict next day up or down, measuring hit rate to evaluate accuracy. Both types of the studies were not enough to develop a full-blown trading system that can trade live. However, there is a better way we can develop the trading system by just feeding the computer with all information they need to make a decision and let them learn from that information. This method is one type of machine learning called reinforcement learning.

To the best of researcher knowledge, no study uses advanced area of machine learning and deep learning which is the combination of deep learning (Kalmus, Trojan, Mott, & Strampfer, 1987) and reinforcement learning (RL) to develop a trading system for the Foreign Exchange Market (FOREX). Upon this concept,

Computers can develop their strategy by learning from data, no need to tell them the fixed rules about how to buy, how to sell, when to buy, when to sell, how much to buy and how much to sell. The computer will be given the data and make their own decision.

1.3 Purpose of the Study

The purpose of this paper is to explore the possibility to create an automated and robust trading system using combined knowledge from machine learning, quantitative finance, and big data. We try to answer two research questions:

1. Can we teach a computer to develop a trading system that beat buy-and-hold strategy (B&H)?
2. Can the machine trading performance surpass the experienced trader?

EMH (efficient market hypothesis) state that it is impossible to beat the market by timing the market consistently. Therefore, the best strategy for the EMH supporter is buy-and-hold (B&H). However, if the machine can see the repeatable, profitable patterns in the market that human cannot see, it is possible that the machine can detect and time the market correctly. Finally, it will be possible to make consistent profits and provide the performance which is better than buy-and-hold. We could also use the benchmark from BarclayHedge of currency index fund as a performance measurement for experienced traders to answer if the machine is better than a human expert (BarclayHedge, 2017).

1.4 Contribution

Findings from this paper will contribute to the investment and asset management industry especially the quantitative hedge fund to explore more about using the machine learning or artificial intelligence to develop trading strategies efficiently. The result will inspire trading system developer, investment advisor, quantitative trader, quantitative researcher to explore more in the area of machine learning and artificial intelligence. Forex brokerage can develop machine learning model to service their customers with limitless possibilities. Bank trader will also learn more of application of artificial intelligence for foreign exchange trading from

the desk. A hedge fund can develop more robust strategies for higher profitable opportunities. An institutional investor will find the way to rebalance the portfolio in such a way that more diversified. The private or individual investor will be beneficial from another alternative investment.

Moreover, this study will support those studies in the past that contradict to the financial theories such as EMH and Random Walk Model in that it is possible to make consistent returns from the speculative market by identifying the hidden pattern in the market by the method of machine learning. It is the novel method to trade the market profitably rather than buy-and-hold strategy.

1.5 Implication

The implication of this study, the possible future will be more of robots or the machines to make a crucial decision such as trading decision. The most obvious advantages of the machine over the human is the computing power and discipline. A human can become too emotional when they are in the trade so when they need to decide at a crucial time, they tend to make a suboptimal decision. However, there will be a need for highly skilled system developers who possess the skill in several areas such as programming, statistic, financial trading. Moreover, Machine can be more suitable to the task that need not of wisdom such as trading task. In the very near future, traders and financial analyst might be replaced by the robots.

The paper consists of 6 Chapters, Chapter 2 and 3 are a literature review and research methodology, respectively. Empirical results and data analysis are reported in Chapter4. The discussion and findings are in Chapter 5. Lastly, Chapter 6 is the conclusion, limitation, and future research.

CHAPTER 2

LITERATURE REVIEW

2.1 Random Walk Theory

The Random Walk theory believes that successive price changes are independent of each other so that it is impossible to consistently make profits from the market by using the technical analysis or fundamental analysis (Fama, 1995). The model explains that the stock price is purely random and unpredictable; however, this paper will contradict to this model in that AI can learn the hidden patterns in historical data and make a profitable decision based on these patterns.

2.2 Efficient Market Hypothesis

Efficient market hypothesis (EMH) state that the price of security fairly and fully reflects all available information. A direct implication of EMH is that the accurate timing to buy and sell from the market is purely random due to the random walk of the stock price. EMH also explain the speed and quality of the price adjustment according to the information. If the market is very efficient also refer to as strong form, there will be no one can earn a consistent abnormal return from trading.

It has been recognized that there are three forms of EMH which are weak form, semi-strong form, and strong form. The weak form of efficient market hypothesis indicates that no investors will make abnormal returns because the historical price data will always reflect the full information (Technical Analysis will not be useful). Semi-strong form of efficient market hypothesis indicates that current price fully reflect both historical price data and available public information so no one can make abnormal returns based on this information (Both Technical Analysis and Fundamental analysis will not be useful). Strong form of efficient market hypothesis indicates that current price will already reflect all information that all participant will

be able to access even investor with inside information still cannot make abnormal returns (Technical analysis, Fundamental analysis, and Inside Information are not useful) (Malkiel, 1989).

Fama had laid the foundation about the efficient market hypothesis that all investors can easily access to the same public information, finally, nobody will be able to earn abnormal returns consistently. Profitable trades can be made from time to time could be possibly a fluke. According to EMH, The investors will react to market instantly so the profit opportunity will disappear (Malkiel & Fama, 1970). Proponents of the EMH, hence, suggest that the most appropriate strategy to trade the market is to buy-and-hold.

This study contradicts to what EMH have been proposed. If the machine/AI can learn how to make a trading decision from the available historical data and make consistent returns, it implies that the best strategy is not buy-and-hold.

2.3 Financial Trading System

The trading system is the set of specific rules, together with parameters, that determine the entry and exit signal for the trade. Generally, the signal will be marked on the chart (Investopedia, 2017). The financial trading system has been increasingly evolving into the algorithmic trading system, the trading method that uses a computer to execute the orders. Farjam et al. (2018) stated that the current market now is hybrid, the market which is composed of both human and computer trader. However, there is still inconclusive about the effect of the increase in the algorithmic trading system relating to the increase in volatility and price discovery (Farjam & Kirchkamp, 2018).

Some algorithmic trading systems are based on the trading rules, some are based on statistical learning, and some are developed from machine learning. These algorithmic trading systems have been developed from historical data such as price, volume and order flow. All data are an underlying foundation for higher level mathematical transformation such as trading formula and machine learning. Wang et al. (2009) have summarized that there are five types of algorithmic trading model which are price, time, shortfall, volume participation, and smart order routing

algorithm. Moreover, these can be categorized as four most popular strategies which are 1) volume-weighted average price 2) time-weighted average price, stock index futures arbitrage, and statistical arbitrage. Almost all algorithmic trading systems are developed based on historical data for the implication that these data has predictive power for developing a profitable trading system.

Wang et al. (2009) have also stated that the evolution of trading system resulted from the expansion of the market, relaxation of government regulation, the speed of financial transaction to execute the orders, and new advanced technology. Therefore, the trading system developer will try to stay ahead of the competition to gain the advantage over the competitors.

Computers have been increasingly used to develop, execute, and automate the trading system. Nowadays, the trading system will become more complex. Exploration of using a computer and advanced computation become a more attractive method to develop the new trading strategy. This study is also trying to leverage the advanced technology to gain the advantage in developing the profitable trading system.

2.4 Technical Analysis

Taylor et al. (1992) stated that technical analysis is the analysis of financial market that attempts to forecast the future price movement based on the past historical price, volume, and volatility. Price and volume can be further transformed into several of indicators which can help them identify market opportunities. Several studies explained how to apply the technical analysis to develop the trading system. Farias Nazário (2017) has conducted extensive studies for technical analysis literature reviews and proposed that there are three significant indicators that most frequently used which are stochastic, relative strength index and moving average. Moreover, there are also some tools which were used to study for technical analysis in the literatures such as econometric model, evolutionary algorithm, genetic algorithm, statistical analysis, neural network, and others(Farias Nazário, Silva, Sobreiro, & Kimura, 2017).

Historical data and technical analysis are the basic inputs to this study because when computers learn to develop trading decisions, we need to feed raw price data and technical indicators (all of them are considered to be the main ingredients to our study) into the system to generate trading decision such as buy and sell.

2.5 Gold Market

There are some studies that linked technical analysis and speculation of the gold price. Batten (2018) has studied the intraday predictive power of 3 technical indicators which are simple moving average (SMA), weighted moving average (WMA), and exponential moving average (Dempster & Leemans, 2006) and found that there is a predictive power for some combination of parameters (Batten, Lucey, McGroarty, Peat, & Urquhart, 2018). It also indicates the possibility to use technical analysis to develop profitable trading systems. Moreover, Baur (2015) also conducted the speculative trading in the gold market and found that there is a pattern for bubble-like characteristics which is predictable. It can substantiate the underlying assumption that technical analysis which is derived from historical data possess some predictive power for the gold market (Baur & Glover, 2015).

2.6 Foreign Exchange Market

The foreign exchange market (forex, FX, or currency market) is a global decentralized market for the trading of currencies. It includes all aspects of buying, selling and exchanging currencies at current or determined prices. Regarding the volume of trading, it is by far the largest market in the world, followed by the credit market. The primary participants in this market are the larger international banks. Financial centers around the world function as anchors of trading between a wide range of multiple types of buyers and sellers around the clock, except weekends. The foreign exchange market does not determine the relative values of different currencies but sets the current market price of the value of one currency as demanded against another. In this paper, the researcher found advantages to choose forex market over stock market due to the following reasons:

1. No corporate action (data cleaning is much more convenient)
2. 24-hour market
3. The largest financial market in the world
4. Data access is prevalent
5. The most liquid market in the world

2.6.1 Product

The world currencies are the products that will be traded on the foreign exchange market. There are currencies called “Major pair”, governing all the majority of foreign exchange market transaction which are United State Dollar (USD), Euro (EUR), Great British Pound (GBP), Swiss Franc (CHF) and Japanese Yen (JPY).

2.6.2 Market Participants

International Trade Transaction

When there are transactions for export and import between countries that use a different currency, it will be transactions on the foreign exchange market. If the company in the UK would like to buy the product in the US, UK Company has to buy US Dollar and simultaneously sell British Pound to get the products. On the other hand, if UK Company wants to sell the product to the US, they will need to convert the US Dollar back to British Pound by Buying British Pound and Sell US Dollar.

Hedgers

Hedging is the attempt to minimize the risk of the position by financial transaction execution (Levinson, 2014). The obvious example of the hedger is the business party that is obligated to pay foreign currency in the future. They do not know if the currency will go against them or not. They may use futures or options contract as tools to minimize their risk.

Speculators

The majority of trading volumes in foreign exchange market comes from speculative trades. Speculators take the position based on the expectation of the price direction. For example, when the hedger tries to minimize their risk, the speculators will undertake the risk in exchange for the opportunity to earn profits.

Foreign Exchange Dealer and Brokers

Foreign exchange dealer usually earns profits by offering both bid and ask price, the profits are the difference between the bid and ask called spread. However, brokers are different from a dealer in that they do not take the opposite side, but earn profits from brokerage fee.

Central banks and treasuries

Central bank objective to participate in the foreign market is to achieve the target rate which comes from the policy. The target will be controlled due to the policy to encourage export and import for the trade in each country.

Bank and non-bank institution.

Major activity of the banks that they usually participate in forex market are mostly in the wholesale market or interbank market. Most activities in interbank market are done by their own account called proprietary desk.

Non-bank institutions which are an insurance company, multi-national Corporation, financial institution, hedge fund, mutual fund and provident fund, etc. also participate in the wholesale market to minimize or to diversify risk for their portfolio.

Individuals and firms conducting commercial or investment transactions

A large firm like multi-national corporation also participates in the foreign exchange market. Their objective is to hedge against the volatility of foreign exchange market, to minimize the risk, and to make an investment for future profits.

2.7 Technical Analysis for Trading System Contradict to EMH

Several studies have supported that technical analysis can be utilized to develop the profitable trading systems. Technical analysis in literature, mainly applied in the speculative markets such as foreign exchange market and the stock market, had

been studied by several of researchers. Many studies found that this technique can be applied profitably in the markets. Taylor et al. (1992) conducted the questionnaire survey the professional foreign exchange dealer in Hong Kong and found that technical analysis is more useful when used in shorter-term and most of them use to forecasting the trend and turning point (Taylor & Allen, 1992). Neely et al. (1997) used genetic programming technique to find technical trading rules, and found strong evidence to support out-of-sample excess returns for six exchange rate from 1981-1995 (Neely, Weller, & Dittmar, 1997). Moreover, Lu et al. (2012) investigated the application of candlestick reversal pattern which is the relationship of open, high, low, and the closing price of stock in Taiwan during 2002-2008. All three bullish reversal patterns was profitable (Lu, Shiu, & Liu, 2012). Vajda (2014) had conducted the test for trading forex market with MACD and found that it is possible to earn profits from forex market with technical indicators like MACD in timeframe 1 hour (H) with optimized stop loss and target profit (Vajda, 2014).

Moreover, there is a study to provide empirical evidence from FOREX market that contradicts to the efficient market hypothesis. Alonso et al. (2015) had conducted the study of automated trading in Forex market. The study was conducted for six currency pair which are EUR/USD, GBP/USD, USD/CAD, USD/JPY, USD/CHF, AUD/USD, the optimized period start form 20001-2008), the testing period from 2008-2011, the indicator used for generating signal was MACD. The study showed satisfactory results for all currencies. Results from all currencies showed positive returns whereas ETF showed negative returns in some years. This study contradicts to the efficient market hypothesis. (Alonso-González, Peris-Ortiz, & Almenar-Llongo, 2015). Coakley et al. (2016) had conducted the trading study of 22 currencies quoted in US dollar, 113,148 technical trading system had been tested. The result showed that the robust trading rules declined overtime especially the traditional trading rules which used a moving average. However, the newer technical trading rules which used Bollinger Band, MACD and RSI showed robust, profitable results over the testing period started from 1997-2015 (Coakley, Marzano, & Nankervis, 2016). The paper showed that it is possible to earn abnormal returns in the forex market by using technical analysis developed from historical price data.

Table 2.1 Summary for trading foreign exchange market study with Technical analysis

Authors	Article Title	Currency & period	Indicators used	Results
Neely, Weller & Dittmar, 1997	Is technical analysis in foreign exchange market profitable?	EURUSD, GBPUSD, GBPJPY, USDCAD, USDCHF, AUDUSD 1981-1995	Genetic Algorithm	Profitable
Vajda, 2014	Could a trader using only "old" technical indicator be successful at the Forex market?	EURUSD (2000-2003)	MACD	Profitable if optimized SL/TP
Alonso-González, 2015	Providing empirical evidence		MACD	Comparison to ETF in the same period, all currencies are profitable
Coakley et al., 2016	How profitable are FX technical trading rules?	22 Currencies based on US dollar (1997-2015)	Bollinger Band, MACD, RSI	Step SPA-Test show robust evidence for a robust, profitable trading system

2.8 Artificial Intelligence (AI)

AI is the ability of digital computers or computer-controlled robot to solve the problem that is typically associated with the higher intellectual processing capability of a human (Ertel, 2018). Artificial intelligence is the program that sense, think, act, and adapt to the environment. Machine learning is the algorithm of the machine to learn from data. Deep learning is the new subset of machine learning which become popular because of the performance and diverse applications.

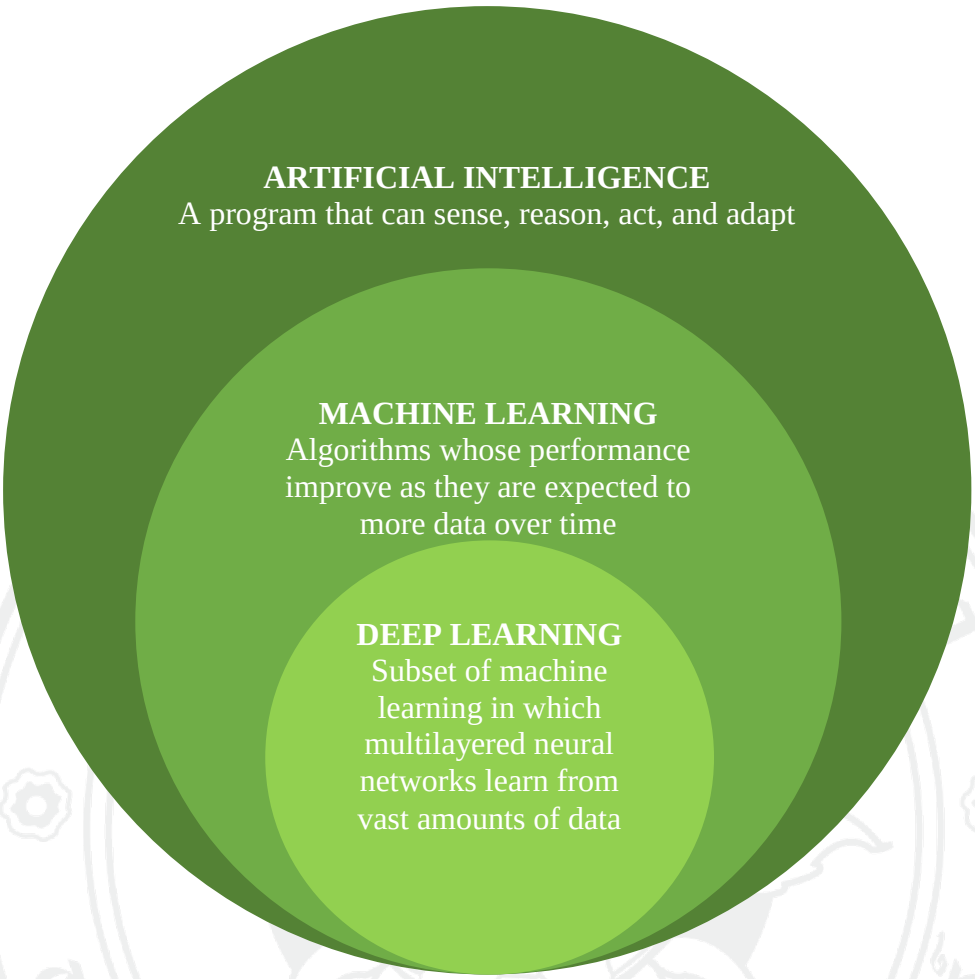


Figure 2.1 Relationships between Artificial Intelligence, Machine learning, and Deep learning

2.9 Commodity Trading Advisor (CTA)

Nasdaq provides the definition of CTA which is "An investment manager that focuses on long and short trading in the futures markets. The trades are often intraday trades. Sometimes referred to as Managed Futures" (Nasdaq, 2018).

2.10 Machine Learning

Machine Learning (ML) is a fascinating field of artificial intelligence (AI) research and practice where we investigate how computer agents can improve their perception, cognition, and action with experience. Machine Learning is about machines improving data, knowledge, experience, and interaction.

Machine learning techniques to intelligently handle large and complex amounts of information build upon foundations in many disciplines, including statistics, knowledge representation, planning and control, databases, causal inference, computer systems, machine vision, and natural language processing (Veloso, 2017).

Machine learning application in finance is the new area due to the advancement of computing power, latest algorithm to solve the complex problem that never happens before, especially in the area of deep learning that is being applied in computer vision, voice recognition, machine translation, etc. Machine learning can be categorized into three groups based on how machine learning from data which are

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning.

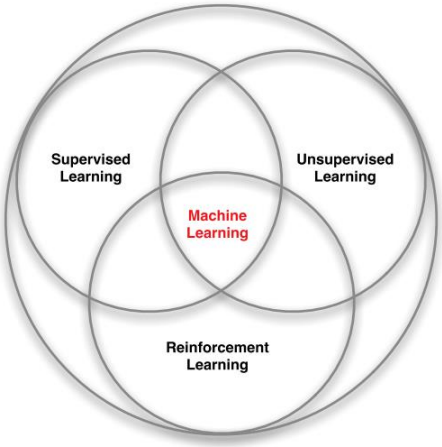


Figure 2.2 Three Major Types of Machine Learning

2.11 Supervised Learning

Supervised learning is a type of machine learning algorithm that identifies the function from labeled data. The supervised learning mostly will use for classification and regression problems, such as prediction of the fraudulent online transaction, disease prediction, time series forecasting.

2.12 Unsupervised Learning

Unsupervised learning is a type of machine learning algorithm that identifies the inference of the data set without labeling (do not know the ground truth) . For example, unsupervised learning is used to find the group of similar features, such as K-mean clustering,

2.13 Reinforcement Learning

Concept and terminology

Reinforcement learning is one of the approaches in machine learning that machine can learn sequential decision-making process from data. There are composed of environment, states (what features the agent can sense from the environment) , action, and reward. The agent will learn to find the optimal policy (what action to take in each specific state) that maximize the cumulative future reward. Agents sometimes are called learner or decision maker.

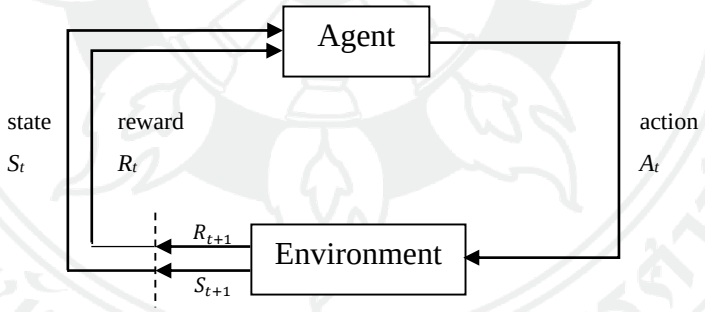


Figure 2.3 Reinforcement Learning Framework

State $s_t \in S$ where is all possible states

Action $a_t \in A(S_t)$ where $A(S_t)$ is all action in each state t

Final/terminal states: The states that have no available actions are final/terminal states.

Episode: An episode is a complete play from one of the initial state to a final state.

For example, the agent randomly starts in one state (s), then choose an action (a) to earn an immediate reward (r) and end up at next state (s´) and the process keep repeating as Markov decision process (MDP)

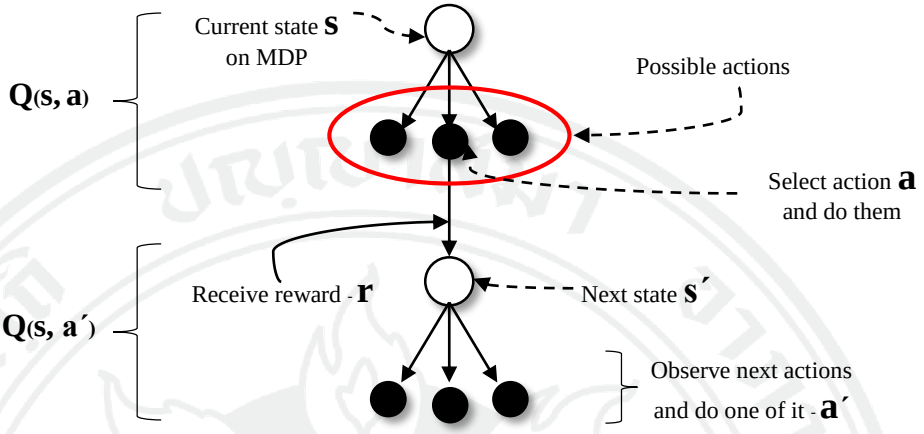


Figure 2.4 Q-Value in Sequential Decision Process
Source: Vladimir and Kabysh

Experience tuple we can get the information of series as called experience tuple <s,a,s,r´> to iterate over time. For example, we can start with s_0, we take action a_0, we get the reward r_0, and then we end up with state s_1, and then repeat again so we can have the series of

$$S_0 , A_0 , R_0 , S_1, A_1, R_1,\ldots\ldots\ldots\ldots, S_n \qquad (1)$$

Where

S_0 is a numerical representation of state zero

A_0 is a numerical representation of action at state zero

R_0 is a numerical representation of reward at state zero

S_1 is a numerical representation of state one

A_1 is a numerical representation of action at state one

R_1 is a numerical representation of reward at state one

Cumulative reward: The cumulative reward is the discounted sum of reward accumulated throughout an episode:

$$R = R_0 + \gamma R_1 + \gamma^2 R_2 + \ldots\ldots\ldots \qquad (2)$$

Where

R_i= Reward at state i

γ = Discounted rate

Policy: A Policy is the agent's strategy/ behavior to choose an action in each state. It is noted by π. The policy is different from the plan in that the policy is stochastic based on probabilistic state transition, but the plan is deterministic.

Policy in each state can be

Deterministic policy $\pi(s) = a$ (3)

Stochastic policy $\pi(a|s) = P[a|s]$ (4)

Where

$\pi(s)$ = policy in any state

a = action

$\pi(a|s)$ = policy to choose action given state s

$P[a|s]$ = probability of state transition

Value function: a prediction of future reward. Q- Value function is expected a future reward

Optimal policy: The optimal policy is the theoretical policy that maximizes the expectation of cumulative reward. From the definition of expectation and the law of large numbers, this policy has the highest average cumulative rewards given sufficient episode.

The objective of reinforcement learning is to train an agent such that his policy converges to the theoretical optimal policy.

There are three approaches to reinforcement learning (Silver, 2015).

1. Value-based RL: Find optimal value function to find the optimal policy
2. Policy-based RL: Find the policy directly
3. Model-based RL: Build a model of the environment

In this paper, we will focus on Value-based RL which is Q learning and combine with the concept of the deep neural network to become deep Q network or deep Q-learning.

2.14 Q Learning

Q learning is one of the most widely accepted algorithms in reinforcement learning. Q function is value function based approach which the agent will learn value function and choose the action based on the highest Q-value. There are two value function that needs to be specified first which are followings:

1. State-value function V(s): this value tell how good it is to be in any state
2. Action- value function Q(s, a): this value tells the quality of certain action in any state.

From above information, we can create state-value function V(s) that represent how good the agent is in any state used the Bellman Equation as following:

$$V(s) = \max_a(R(s, a) + \gamma V(s')) \quad (5)$$

Where

V(s) is value function of any state

R(s,a) is reward function of any state and action

γ is a discounted rate

V(s′) is value function of state S′

From (5), in any state, the value of each state is equal to the sum of Reward plus γ (discounted rate) multiplied with the value of the next state given that choose the action that maximizes the sum. If discounted factor (γ) = 0, that means V(s) = max(R(s, a)) or we do not care about the future state that we will end up with we care only the reward. If discounted factor (γ =1), that means V(s) = max(R(s, a) +V(s′)), or we do care about the future value of the next state (γ is the parameter to adjust depending on how much we care about future).

From above Bellman equation, If we take account of Markov decision process (MDP) which is a stochastic process of the random sequence (if the agent performs a

certain action there are probability distribution to end up in any states), we could rewrite the above equation to following in the recursive form as below:

$$V(S) = \max(R(s, a)) + \gamma \sum_{s'} p(s, a, s')V(s') \quad (6)$$

Where

V(s) is value function of any state

R(s,a) is reward function of any state and action

γ is a discounted rate

P(s,a,s′) is the probability of transition to end up in any states

V(s′) is value function of state S′

Now, we know how good to be in any state. The next step, we will find the quality of each specific action in any state which is Q. We use Q function or Q(s, a) to represent the quality of each action in any given state. Therefore, we will get the equation for Q(s, a) as below:

$$Q(s, a) = R(s, a) + \gamma(\sum_{s'} p(s, a, s')V(s')) \quad (7)$$

Where

Q(s,a) is Q function of any state and action

R(s,a) is reward function of any state and action

γ is a discounted rate

P(s,a,s′) is the probability of transition to end up in any states

V(s′) is value function of state S′

There is a link between V(s) and Q(s, a) that the term in the bracket in (6) is equal to (7), so the value in (6) is to choose a maximum of all possible Q values.

From (7), we can replace the last term of V(s′) with $\max_{a'}(Q(s',a'))$ to get the recursive form of the Q function so we would get the following Bellman equation in the form of Q:

$$Q(s, a) = R(s, a) + \gamma(\sum_{s'} p(s, a, s') \max_{a'}(Q(s',a')) \quad (8)$$

Where

Q(s,a) is Q function of any state and action

R(s,a) is reward function of any state and action

γ is a discounted rate

p(s,a,s′) is the probability of transition to end up in any states

Q(s′, a′) is value function of state S′

From (8) we can simplify into to the deterministic Q(s,a) as following:

$$Q(s,a) = R(s,a) + \gamma \max_{a'}(Q(s',a')) \quad (9)$$

Where

Q(s,a) is Q function of any state and action

R(s,a) is reward function of any state and action

γ is a discounted rate

Q(s′, a′) is value function of state S′

2.14.1 Temporal Difference

Compare Q(s,a) before action to Q(s, a) after the action in any state to find Temporal Difference

$$TD_t(a, s) = [R(s, a) + \gamma \max_{a'}(Q(s',a'))] - Q_{t-1}(s, a) \quad (10)$$

Where

$TD_t(a,s)$ is Temporal Different function of any state and action

R(s,a) is reward function of any state and action

γ is a discounted rate

Q(s′, a′) is Q function of state s′ and action a′

$Q_{t-1}(s, a)$ is Q function of state s and action a at time t-1

So, we can get the update rules

$$Q(s, a) = Q_{t-1}(s, a) + \alpha TD_t(a, s) \quad (11)$$

Where

Q(s, a) is Q function of state s′ and action a′

$TD_t(a,s)$ is Temporal Different function of any state and action

$Q_{t-1}(s, a)$ is Q function of state s and action a at time t-1

α is learning rate ($0 \leq \alpha \leq 1$)

$$\pi(s) = \text{argmax}_a Q(s,a) \quad (12)$$

Where

$\pi(s)$ is policy at state s

Q(s,a) is Q function of state s and action a

2.14.2 Q-learning Algorithm

The main idea of Q-learning is that we can iteratively repeat the process of the update until it converges as the following algorithm:

Initialize Q[num_states, num_actions] arbitrarily

Observe initial state s

Repeat

Select and carry out an action a

Observe reward r and new state s′

$$Q[s,a] = Q[s,a] + \alpha (r + \gamma \max_{a'} Q[s', a'] - Q[s,a])$$

Until terminate

This procedural approach can be translated into simple steps as follows:

1. Initialize the Q-values table, Q(s, a).
2. Observe the current state, s.
3. Choose an action, a, for that state based on one of the action selection policies.

4. Take action, and observe the reward, r, as well as the new state, s'.
5. Update the Q-value for the state using the observed reward and the maximum reward possible for the next state. The updating is done according to the formula and parameters described above.
6. Set the state to the new state, and repeat the process until a terminal state is reached.

2.15 Reinforcement Learning in Financial Trading

Moody et al. (1998) studied the application of RRL (recurrent reinforcement learning) in 3 empirical studies 1) Trader simulation 2) Portfolio management formulation 3) S&P500 and T-bill asset allocation system. For trader simulation, they test 2 RRL in one simulation stock price (one with maximizing profit, one with maximizing differential Sharpe ratio compare to forecast model and found that RRL performed better. For Portfolio management formulation, RRL trained to maximize differential Sharpe ratio perform better than maximize profits. For S&P 500 and T-bill asset allocation system, it showed the predictive power from 1970 to 1994 (Moody, Wu, Liao, & Saffell, 1998).

Moody et al. (2001) introduced the direct reinforcement learning using differential Sharpe ratio as performance function to be optimized. They found that direct reinforcement learning performs better than Q learning for asset allocation problem in S&P500 T-bill portfolio (Moody & Saffell, 2001).

Gold (2003) studied RRL (recurrent reinforcement learning) to explore the effect of training parameters on the performance of FX trading (Gold, 2003).

Dempster el al. (2006) also studied to deal with the usable, fully automated intelligent system. The system based on three layers 1. Machine learning algorithm 2. Risk management layer 3. Dynamic optimization layer and together called Adaptive reinforcement learning which is based on RRL. The added features make the model more flexible for different risk tolerance leave. The study used EUR/USD 1 min (from 2000 to 2002). It showed absolute profits in pips (5104) or approximately 26% p.a. , compared to Buy and hold (8% loss or 1636 pips loss) (Dempster & Leemans, 2006).

Du et al. (2009) studied reinforcement learning method between RRL and Q learning in asset allocation problem between risky and riskless asset. The study, used simulation, showed that RRL outperforms Q learning regarding stability when exposed to the noisy dataset. Q-learning is sensitive to the selection of value function. On the other hand, RRL is more flexible to choose objective function (Du, Zhai, & Lv, 2016).

2.16 Deep Learning

Deep learning belongs to the broader family of the machine learning method. Deep learning is composed of the artificial neural network (ANN) with multi-hidden layers.

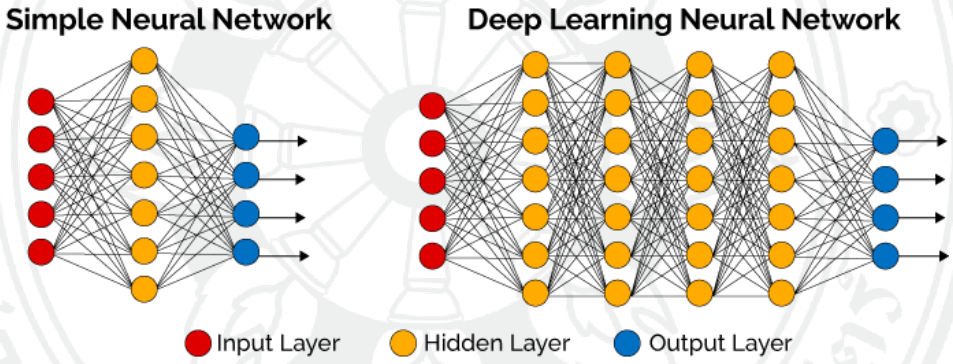


Figure 2.5 Neural Network Architecture

From above picture, single neural network compares to deep learning neural network with four hidden layers. In the context of this paper, there are several uses of deep learning depend on training methods (supervised, unsupervised, and partially supervised), in this paper, we will use deep learning neural network as a function approximation. We will provide the state as input and try to predict Q; then we will input the next state (S′) and predict new Q. We will repeat this step to update Q according to Deep Q learning algorithm explain later.

2.16.1 Convolutional Neural Network (CNN)

Convolutional neural network (CNN) is one type of the deep learning network which works best for the image task. This network has been widely used in the tasks like image classification, image recognition, and computer vision. For training AI to play the game, CNN also is used to convert pixel to signal, and the output will be classification types of signal.

CNN is composed of two types of layers which are a convolutional layer, subsampling layer. These types of the layer will connect successively. In convolutional layer, the convolution operation will be performed; the outcome will be passed on to the next layer. In the subsampling layer, representation size and parameter will be reduced until the data become one-dimensional vector (Sezer & Ozbayoglu, 2018).

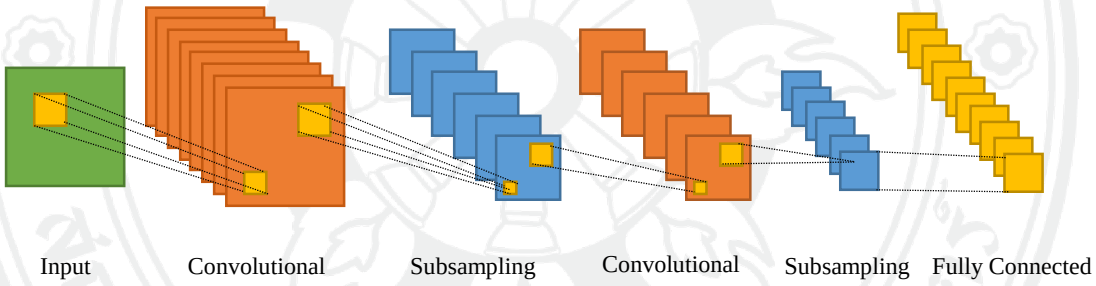


Figure 2.6 Generalized Convolutional Neural Network Structure

2.17 Deep Q Learning/ Deep Q Network

It has been known that deep-learning network is good at learning hierarchical of patterns of data, and also good at representation of noisy data, invariant, disturbance data. Thus, we can use Deep Q-Learning as a function approximation to find Q(s, a).

Learning:

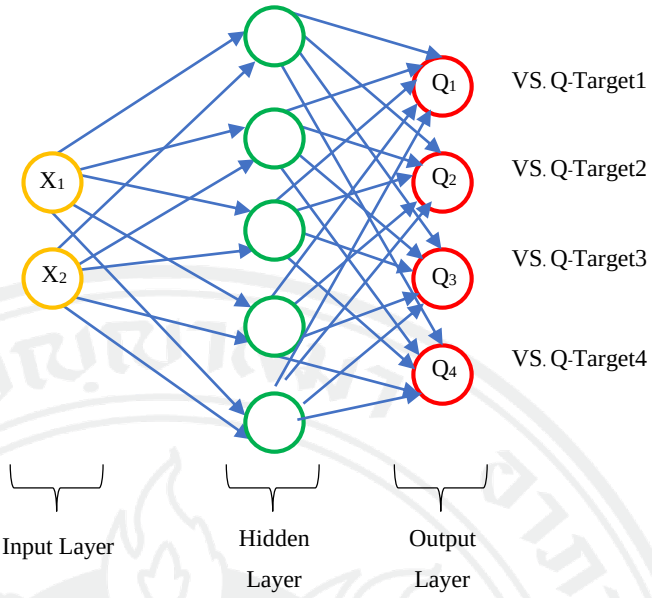


Figure 2.7 Deep Neural Network for Q learning

The above picture shows that we can feed the input to the network (state) and calculate the predicted Q using the deep neural network. The predicted Q will be compared to the target for each specific action (in this example, there are four actions so we can have four Q values).

Loss function will be calculated as below:

$$\boldsymbol{L = \frac{1}{2}[\underbrace{r + max_{a'} Q(s', a')}_{\text{Target}} - \underbrace{Q(s, a)}_{\text{Prediction}}]^2} \quad (13)$$

Where

L is loss function

r is reward

Q(s, a) is Q function of state s and action a

Q(s′, a′) is Q function of state s′ and action a′

Given a transition < *s*, *a*, *r*, *s'* >, the Q-table update rule for Q-learning in the previous algorithm must be modified when applying deep neural network with the following:

1. Do a feedforward pass for the current state *s* to get predicted Q-values for all actions.
2. Do a feedforward pass for the next state s' and calculate maximum overall network outputs $max_{a'} Q(s', a')$.
3. Set Q-value as a target for action to $r + \gamma max_{a'} Q(s', a')$ (use the max Q-values calculated in step 2). For all other actions, set the Q-value target to the same as initially returned from step 1, making the error 0 for those outputs.
4. Update the weights using backpropagation.

Experience Replay: Due to the use of neural network to solve Q-learning, we would end up with diverges due to the correlation between samples and Non-stationary target. To solve this problem, we keep previous experiences in memory and after each action taken we draw a mini-batch of experience from that memory to perform the update step.

2.17.1 Deep Q-Network (DQN): Experience Replay

To remove correlations, build data-set from agent's own experience

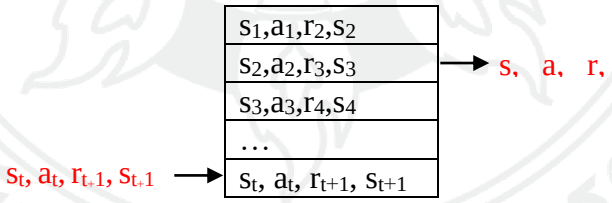


Figure 2.8 Sample Experiences from Data-Set and Apply Update

$$l = \left(r + \gamma \frac{max}{a'} Q(s', a', \bar{\boldsymbol{w}}) - Q(s, a, \boldsymbol{w})\right)^2 \quad (14)$$

Where

l is loss

r is reward

γ is a discounted rate

To deal with non-stationarity, target parameters $\bar{w}$ are held fixed

2.17.2 Action Selection Policy & Exploration/ Exploitation

In reinforcement learning, the agent needs to learn in all possible state to take action. The agent needs some experiences to learn how to make a good action. How the agent knows if it the good action. The agent needs to explore as much as possible. However, if the agent thinks that he already had a good action, sometimes he does not want to explore for more. Therefore, there is a trade-off between exploration and exploitation.

1. Greedy Approach: The agent takes the action to exploiting the knowledge that he knows from current states to choose the action. This makes him does not want to explore for more actions the problems of this approach will arrive with a suboptimal solution.
2. Random Approach: This approach is opposite to greedy approach the agent will always select the random action to explore more.
3. ϵ Greedy Approach: This approach is the combination of Greedy approach and random approach. When starting with the unknown environment, the agent will act randomly and then adjust to more exploitation.

2.17.3 Advantages of reinforcement learning to traditional quantitative trading

1. The agents can learn trading strategy from financial data. It is not necessary to input the preferred indicators or prefer instruments.

2. It is different from supervised learning that one could develop the system that performs well in the past, but fail in the future. Due to the volatility, unexpected event, short-term noise, the better system should be self-adaptive. Thus online learning approach that quickly adapts to such a change will be practically necessary.

3. The agents will learn the optimal policy based on performance function (reward) to maximize the future cumulative reward so that we can adjust the performance function for the agent based on the critical metric of system performance.

2.18 Deep Q Learning In Financial Trading

Deng et al. (2017) studied the performance of trading in a different method which are FDDR, DDR, SCOT, DRL, and BH. The study used three instruments (IF, AG, SU) and used TP and SR as performance function. It found that FDDR show the most attractive results (Deng, Bao, Kong, Ren, & Dai, 2017).

Wang et al. (2016) have researched developing an algorithmic trading system based on DQN which can automatically determine the signal to buy, sell, or hold with each trading time stamp (Wang, 2016).

After rigorously studied, the researcher found that there is still lack of Deep q network learning to trade the forex market. After the success of Alpha go, deep q network (combined deep learning with reinforcement learning) has been applied in several areas including finance. To the best of researcher knowledge, we believe that this paper will be the first to explore this lucrative, most liquid market in the world.

Table 2.2 The summary literature that uses reinforcement learning in financial trading

Literature	Machine Learning Model	Features	Market	Evaluation	Outcome
Moody et al., 1998	RRL	Simulation data	Simulation data	Differential Sharpe ratio MSE Cumulative profits	Compare Max DSR of Long/Short better than of min MSE
Moody & Saffell, 2001	RRL	Simulation data	Forex	Comparison between DRL and Q learning	Portfolio of RRL DRP performance is higher than Q learning

Literature	Machine Learning Model	Features	Market	Evaluation	Outcome
M. a. H. Dempster & Jones, 2001	Genetic Algorithm	Technical indicators	Forex	Sharpe	Trading performance
Gold, 2003	Recurrent Reinforcement Learning	Price & M/S ratio (price movement / spreads)	Forex	Profits and Sharpe ratio	Single layer is better than two layers
M. A. H. Dempster & Leemans, 2006	Reinforcement Learning	Risk parameters / price	Forex	Sharpe/ differential Sharpe ratio	Trading performance
Du et al., 2016	Reinforcement Learning	Price of risky asset	Simulated Data	Interval profit/ Sharpe ratio/	Policy search outperform value search
Deng et al., 2017	FDDR, DDR, SCOT, DRL	Price	Stock	TP/SR	FDDR is the most attractive result
Wang et al.	DQN	Delta of the close price	Index price	Accumulated Wealth, Sharpe ratio, Max DD	DQN is better than BH and RRL

CHAPTER 3

METHODOLOGY

3.1 Mapping Reinforcement Learning to Financial Trading

To solve the trading problems, we need to start mapping trading problem into reinforcement problems. We have to identify the following components:

1. *Set of states:*

   The set of states can be OHLC, indicators, and other features of three instruments (EURUSD, USDJPY, and XAUUSD). This set of states represent the perception that the agent can perceive the world.

2. *Set of Actions*:

   The set of actions can be the actions to take in each state. In this case, there are four actions: {Hold, Buy, Sell, Close}

   The agent will open only one position at a time.

3. *Reward function/Performance function:*

   The reward function is the reward that the agent will earn after performing the action in each state. The reward function can be the function of Cumulative profits (in pips), Sharpe ratio, total profits, and reward to risk.

4. *Experience tuple*,

   Experience tuple is the experience of the agent store in the memory buffer. in this part is the experience of the agent that learn from the data which is $< S,A,R,S' >$ this part will be used as experience replay.

When we all have above 1-4, we will be able to find the optimal policy π by using Deep-Q Learning algorithm.

In this paper, we will test 3 assets/ instruments separately; we will test each currency pair as long/ short trader to compare if we can use Deep-Q Network to develop the model for all instruments.

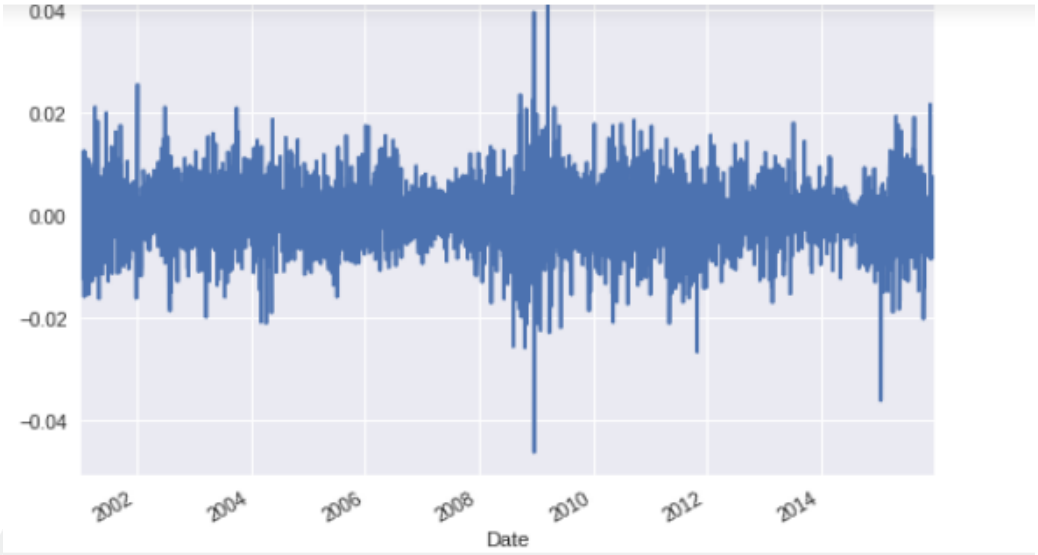
3.2 Data

The study uses 15 years of historical data obtained from the prominent Swiss broker, Dukascopy Bank, Switzerland. The data in our experiment will be daily data from (01/ 01/ 2001 – 12/ 31/ 2015) . Data is downloaded from Dukascopy website (https://www.dukascopy.com/swiss/english/marketwatch/historical/).

We use the daily data obtained from free historical feed data source (Dukascopy, 2017) . Data pre-processing (Cleaning) is necessary to ensure the reliability of the data. Followings are steps for data pre-processing.

1. Checking for missing data: if there is a missing data and replace with the average of 2 previous days.
2. Checking data format to ensure that the data are in a correct format such as numeric value rather than string.
3. Standardize data to ensure that significant numbers of one feature will not outweigh other features with a smaller number. We will use Z-score to the standard the data.

The data was split into a training set (01/01/2001-12/31/2003) and test data set (01/01/2005 -12/31/2015) using the function in scikit-learn python library to split the data automatically. We will randomly initialize the model and let the AI learn in batch during 2001-2004 so the AI will initially learn from small amount of data (2001-2003) and then AI will learn online (learn and update as it get more data) from the beginning (2001) again until it get smarter when it gets more data until 2015.




```
count    3839.000000
mean        0.000057
std         0.006290
min        -0.046251
25%        -0.003393
50%         0.000197
75%         0.003645
max         0.041203
Name: ret, dtype: float64
```

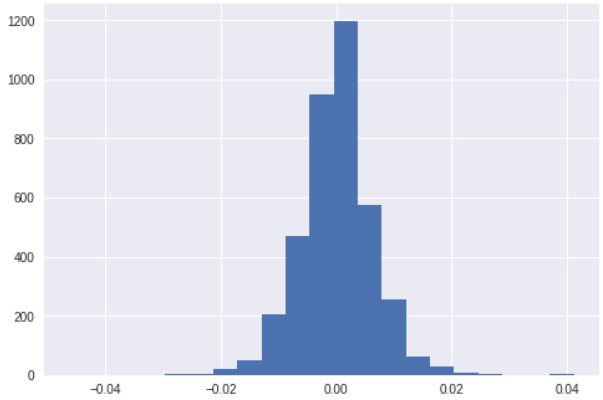


Figure 3.1 Descriptive Statistic of EURUSD Daily Returns Data


```
mean         0.000020
std          0.004653
min         -0.036115
25%         -0.001728
50%          0.000000
75%          0.001771
max          0.031060
Name: ret, dtype: float64
```

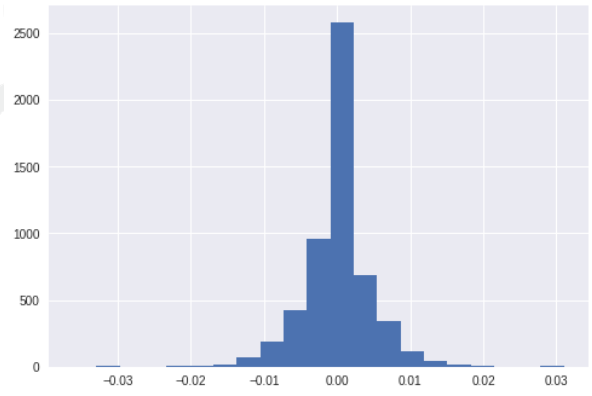


Figure 3.2 Descriptive Statistics of USDJPY Daily Returns Data

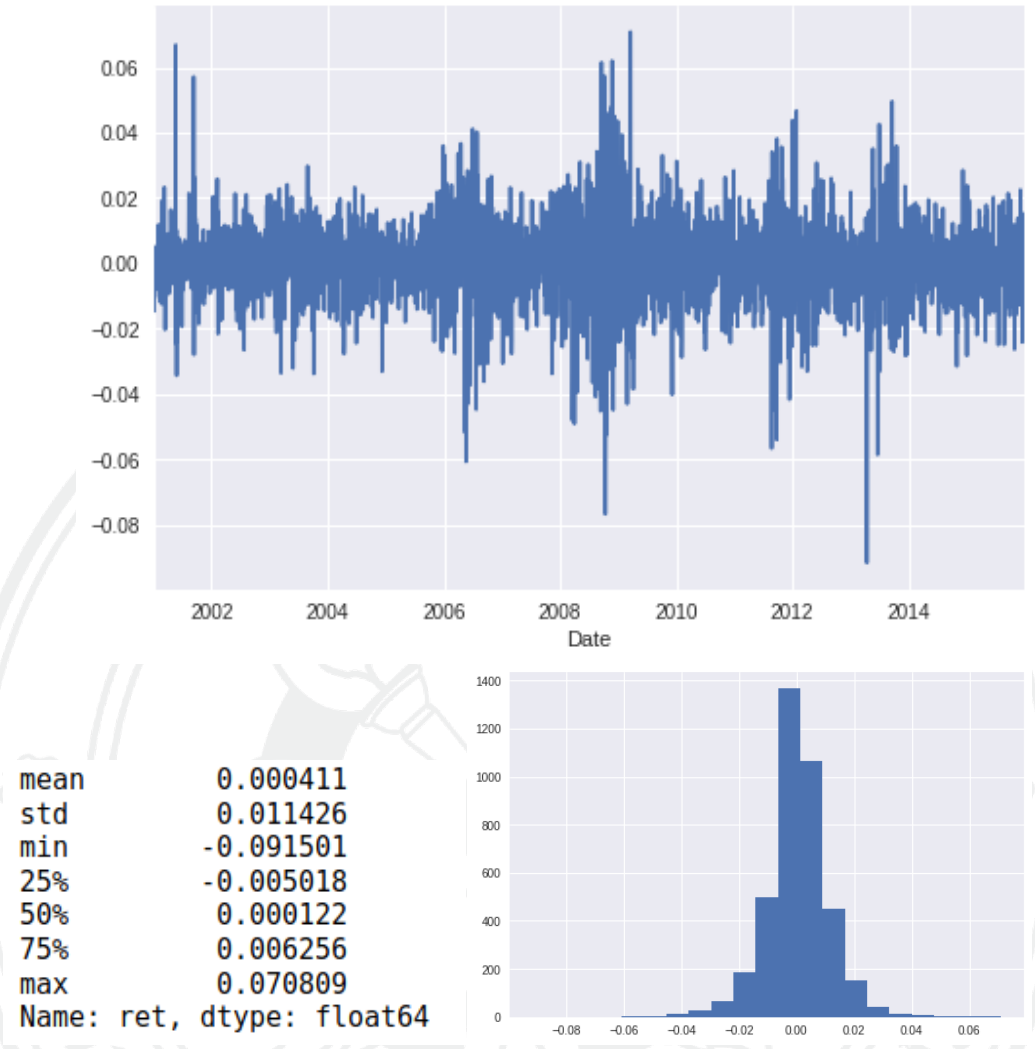


Figure 3.3 Descriptive Statistics of XAUUSD Daily Returns Data

3.3 Online Learning

We will train the agent online with stream data. The Q learning algorithm will update itself incrementally each of time-step. With this algorithm, the agent will learn more when getting more data.

3.4 Paired T-Test

A paired t-test is used as a statistical tool when we want to compare two population means, and both two samples can be paired together. For example, one

sample with the series of returns with one strategy and another sample is series of returns for another strategy.

Procedures for carrying out paired t-test

Let A is the data series for model 1, and B is the data series for model 2. To test the null hypothesis that the mean difference is zero, we need to calculate as followings:

1. Calculate the difference between A series and B Series on each pair.
2. Calculate the mean of the difference.
3. Calculate the standard deviation of the difference.
4. Calculate t-statistic , T = mean difference/ standard deviation of difference.
5. Use the table of t-distribution to compare T value from calculation to T value of (n-1 degree of freedom).

3.5 Experimental Process

1. Data prepossessing: in this step we need to do data cleaning and prepare the database for all currency pair for the daily timeframe. We will replace missing value with the average price for two previous days. We will check the format of the data.
2. Feature engineering: we need to create all relevant features which have the predictive power of price movement this includes all relevant indicators. We can do this by using the python library called Talib to calculate other features such as technical indicators. After we get the OHLC historical data from step 1, we will calculate all necessary indicators. After that, we will standardize all features.
3. Split data into train set and test set using the python library called Scikit-Learn. We will feed data into the training data set (2001-2003). We will train with a small amount of batch learning during the trading phase.
4. Feed data to Deep Q-Network to learn.
5. Hyper-parameter tuning during the training phase: tune the parameters for the architecture of the neural network.

6. Feed data to Deep Q-Network again to learn online. AI will learn again from the beginning and update gradient to get smarter over time.
7. Perform Hypothesis testing using Paired t-Test.
8. Evaluate the results.

3.6 Key Performance Matrices for Trading System

3.6.1 Sharpe ratio

The following relation defines the Sharpe Ratio S:

$$S = \frac{E(R_a - R_b)}{\sqrt{Var(R_a - R_b)}} \quad (11)$$

Where

S is Sharpe ratio

Ra is the period return of the asset or strategy

Rb is the period return of a suitable *benchmark*.

The Sharpe ratio was introduced by William Sharpe (Sharpe, 1966). Sharpe ratio can be used to measure the performance for holding stock by considering risk factor. The Sharpe ratio quantifies how much excess return you get for each unit of risk you are willing to take. Excess return is calculated from the difference between expected returns to risk-free rate. Dividing excess returns with a standard deviation of the daily returns will get Sharpe ratio. There is some limitation form using Sharpe ratio in that, during unstable economic condition, it is difficult to find expected return. Moreover, sometimes, the distribution of returns (risk) may not be a bell-curve so using standard deviation may be misleading. Some fund managers may hold the risky stock in hope for higher returns, but without considering risk factors. So Sharpe ratio can measure portfolio performance without overlook the risk factors. The higher Sharpe ratio means the portfolio will experience lesser volatility. In this paper, we will compare the difference between Sharpe ratio of AI and others (Buy-and-Hold, CTA) to test if it is significantly different or not.

3.6.2 Value at Risk (VAR)

VAR provides an ***estimate***, under a given degree of confidence, of the size of a loss from a trading system over a given period. For example, the hedge fund may determine the VAR of the portfolio as 5% 3-months VAR of 3%; this means that there is 5% probability that the portfolio will decrease in value for more than 3% of portfolio value in three months period.

The advantage of using VAR is to estimate the risk in the portfolio which are composed of several highly correlated assets. The financial institution can determine capital reserve to ensure that there is sufficient capital reserve.

There are two types of VAR calculation methods which are non-parametric and parametric. In this paper, we will use non-parametric method for VAR because we will use historical simulation instead of predicting the future parameters.

For loss *L* exceeding a value *VaR* with a confidence level *c* we have:

$$P(L \leq -VaR) = 1 - c \quad \textbf{(12)}$$

Where

L is loss

VaR is Value at Risk

c is confidence level

3.7 Hypothesis Testing

There are 2 hypothesis testing we need to conduct to answer the research questions we mentioned in chapter 1

Hypothesis 1

H0: There is no significant difference between the performance of AI agent and performance of the buy-and-hold strategy

H1: There is a significant difference that performance of AI agent is superior to the performance of the buy-and-hold strategy

To test the hypothesis 1, we will use Paired t-Test of Sharpe ratio and annual return to test the performance between buy-and-hold and AI agent.

Hypothesis 2

H0: There is no significant difference between the performance of AI agent and performance of experienced trader (CTA)

H1: There is a significant difference that performance of AI agent is superior to the performance of experienced trader (CTA)

To test the hypothesis 2, we will compare our AI agent with currency trader performance (CTA). We will use Paired t-Test of Sharpe ratio and annual returns to test the hypothesis.

CHAPTER 4

RESULT AND ANALYSIS

4.1 Basic Test with Sine Wave Test

We build simulated sine wave data to test the performance of the AI Agent whether the agent can learn and make a profitable trading decision by given predictable and stable pattern, we believe that if AI Agent can learn how to trade from sine wave pattern, it could learn from historical data as well.

We simulated the sine wave with total 800 time steps/days and 20 time steps (peak-to-trough) so that we will get below picture

The reason why we should see the sine wave because it is one of the components in Fourier transform mentioned in literature as Giampaoli (2009) had used advanced Fourier transform to analyze the financial data. He introduced the way to decompose the unevenly-spaced data into the frequency domain with "Lomb–Scargle Fourier transform (LSFT)" method, this can overcome the problem of transform unevenly-spaced data into evenly-spaced data. Lomb introduced this method with the sinusoidal curves (sine wave) to fit the data. Then, scale extends Lomb work later by introducing periodogram (Giampaoli, Ng, & Constantinou, 2009).

4.1.1 Sine Wave Equation

$$Y(t) = A\sin(2\pi ft + \varphi) = A\sin(\omega t + \varphi)$$

Where A = Amplitude

f = frequency

ω = angular frequency

φ = the phase

4.1.2 Steps of Sine Wave test

1. Create the Sine wave function using numpy python library from 800 days with 20 days cycle

2. Initialize state and data series (from close and diff close)
3. Train for 20 epoch
4. Update gradient until performance converge

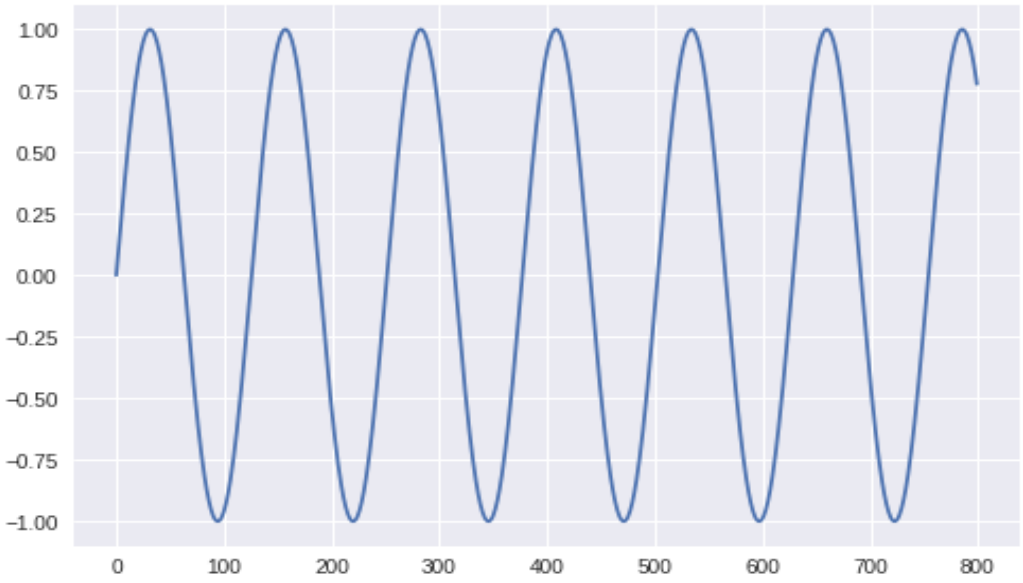


Figure 4.1 Simulated Sine Wave with 800 days

4.2 Experiment with Sine Wave

1. Mapping Trading with Reinforcement learning: we need to specify the state, reward, action to create the experience tuple (<S,A,R,S´>) for the agent
    - State: there are two states which are Close and Diff Close (different in Closing price).These will be used as inputs to feed into the deep neural network.
    - Action: there are 4 actions that AI Agent can take: Buy/sell/close/do nothing
    - Reward: if terminal state = true , reward = cumulative reward
      If terminal state = false, reward = profit for that time step
    - Next state observation: we will observe what happen to the next state (Close and Diff Close) after we have already taken action.
2. Model configuration: we will set parameters and topology for the deep network as following

- Deep network topology

  1 input layer with 2 input node (for 2 inputs)

  2 hidden layers with 4 nodes (fully connected)

  1 output layers with 4 inputs node (for 4 actions)

  The activation function is ‘liner’ to predict Q value

  All network are fully connected with 10% drop-out (to prevent curve fitting)

3. The architecture of our brain

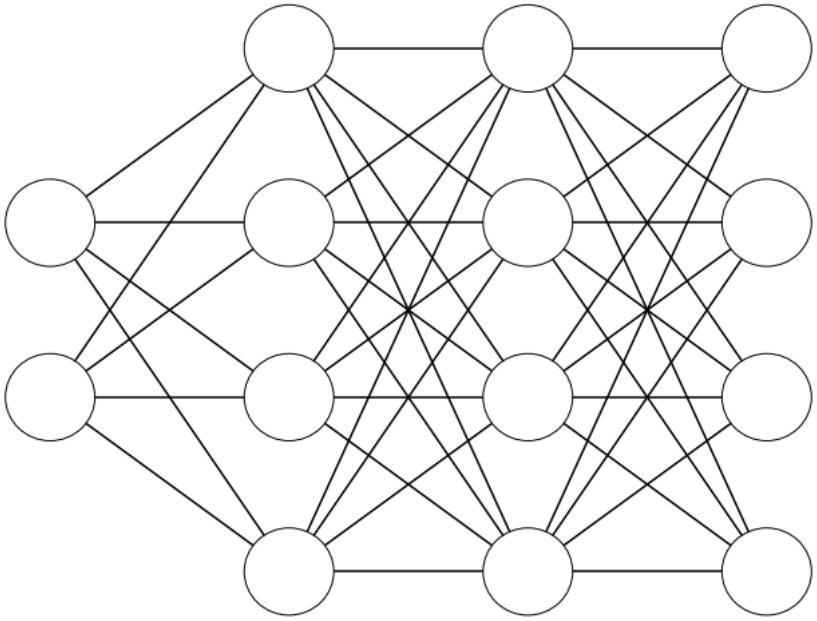


Figure 4.2 Deep Neural Network with 4 Layers (1 input layer, 2 hidden layers, 4 output layers)

4.3 Result of Sine Wave

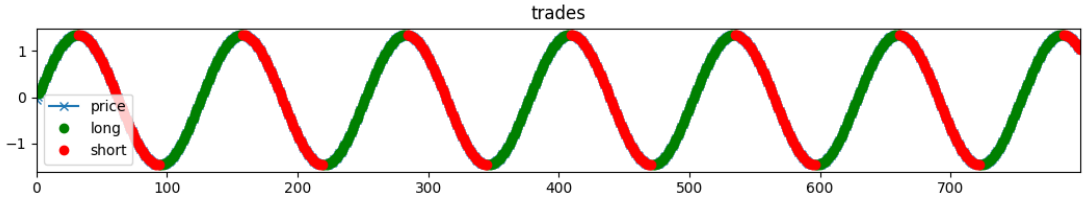


Figure 4.3 The Result of Sine Wave

The above picture shows the result of the sine wave for 800 days (time steps), the green color show possible buy signal and the red color show possible sell signal. We set that there will be only one trade at a time if we will open the new position we have to close the opened position before taking another trade.

Our AI agent will open a long trade when the color change from red to green and will open a short trade when the color change from green to red. There are 14 trades on the top and bottom for the whole series with Sharpe ratio is 33.54.

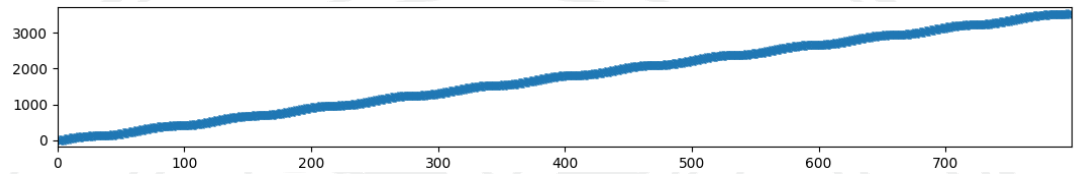


Figure 4.4 The Equity Curve of Sine Wave Test

Above picture show equity curve of sine wave trade with a steady performance, this can signal that it is possible to train the AI Agent to learn the pattern of the historical price. If we know price pattern and there is some degree of repeatable pattern, we would be able to train AI agent to learn the pattern and provide the trading signal very accurately.

4.4 Experiment with Real Historical Data

We will perform three experiments with real historical data (two experiments with two currencies and one experiment with Gold), Total historical data for all experiments will be from 1 January 2001 to 31 December 2015 (total 15 years), we will split the data into 2 sets (train/test), train set (01/01/2001-12/31/2003) and test set (01/01/2005- 12/31/2015). we use the following symbol to represent each currency.

- EURUSD = Euro/Dollar
- USDJPY = Dollar/Yen
- XAUUSD = GOLD/Dollar

4.4.1 Mapping the trading problem to reinforcement learning

The assumption of our backtest

- Initial capital for 100,000
- No transaction cost
- The position sizing is 1% for each trade
- Only one position can be opened at a time
- We enter into the close price of that day

Firstly, we need to perform Mapping Trading problem with Reinforcement learning problem. Therefore, we need to specify the state, reward, action to create the experience tuple for the agent to learn from (<S, A, R, S′>).

The performance of the AI agent also depends on what agent perceive its own environment which are states that the agent can see. Typically, deep learning is good at feature extraction; it can usually detect the relevant features for classification and regression problems. However, when we set up the states which represent the features that agent will learn, we still need human knowledge and experience to choose what to feed into the deep neural network.

States are composed of 7 inputs

1. Close: this feature is from the closing price of each day

2. Diff Close: this feature is calculated from the difference between consecutive closing price.

Diff Close = Close (t) – Close (t-1)

3. Diff Close and SMA(10) = Close- SMA(10) – moving average period 10

Where :

$$SMA(10) = Close_{10}+\ldots\ldots+Close_{1}/10$$

4. Diff Close and SMA(50) = Close- SMA(50) – moving average period 50

Where:

$$SMA(50) = Close_{50}+\ldots\ldots+Close_{1}/50$$

5. Diff Close and SMA(100) = Close- SMA(100)- moving average period 100

Where:

$$SMA(100) = Close_{100}+\ldots\ldots+Close_1/100$$

6. Diff SMA(10) and SMA(50) = SMA(10)-SMA(50)

Where:

$$Sma(50) = Close_{50}+\ldots\ldots+Close_1/50$$

$$SMA(100) = Close_{100}+\ldots\ldots+Close_1/100$$

7. Diff Close and Sine wave = Close – Sine wave

Where:

$$Y(t) = Asin(2\pi ft + \varphi) = Asin(\omega t + \varphi)$$

We add sine wave indicators due to the test on simulate sine wave shows the positive result, so if we transform the price into the cyclic indicator like a sine wave, we believe that we would increase the performance of the AI agent dramatically.

Actions: There are 4 actions which are

1. Buy
2. Sell
3. Close
4. Do Nothing

Reward: we have intermediate reward and long-term reward

If terminal state = false, our reward is based on the price difference if an agent takes action and price increase (long position), the agent gets profits as our reward. If an agent takes short position and price decrease (short position)

If terminal state = true, our reward is calculated for all cumulative profits

4.4.2 Model configuration

We will use deep learning model called ' Convolutional Neural Network (CNN)' which is suitable for developing Deep Q learning model.

4.4.3 Python libraries

a. Keras and Tensorflow (to build our neural network)

b. Pyfolio from Quantopian (to create performance tear sheet) and Backtrader library

c. Jupyter notebook environment (to run the python code)

4.4.4 Our Brain Structure (Network topologies)

1 input layer with 7 nodes

2 hidden layer with 48 nodes

1 output layer with 4 nodes

Our activation function is 'Linear' to output Q value

4.4.5 The architecture of our brain

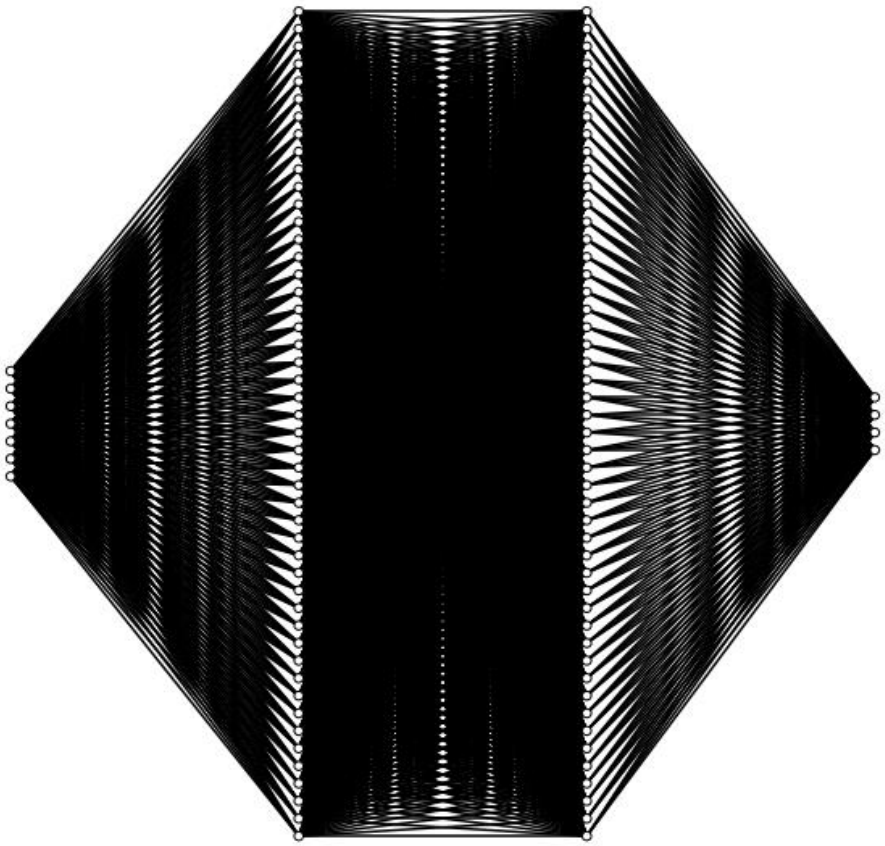


Figure 4.5 Architecture of Deep Neural Network (fully connected) with 7 input, 2 hidden layers (each 48 nodes), 4 output nodes

4.4.6 Our training method (Online learning)

Most trading systems will fail because the market is very volatile. There are several reasons such as market evolution, short-term news, and some noise in the market. The trading system that is not sufficiently adaptive will definitely fail in some market condition, so the better system should be adaptive to the market change. The way that we train the agent is pretty much similar to the real trade and making the trading system more adaptive. We first train our deep learning network with a small amount of data, and then we will rerun from the beginning. Then, In each day, we will add new trading example <S,A, R,S′> to the buffer. After that, we will use minibatch from the buffer to update the Q network by backpropagation.

4.5 EURUSD Result

Table 4.1 Performance Table for AI Agent Learn to Trade EURUSD

```
Entire data start date: 2001-01-03
Entire data end date: 2015-12-30


Out-of-Sample Months: 143
Backtest Months: 35
```

Performance statistics	All history	Backtest	Out of sample
annual_return	0.25	0.56	0.18
cum_returns_final	26.85	2.79	6.34
annual_volatility	0.52	0.82	0.41
sharpe_ratio	0.68	0.94	0.61

From above table, it shows the important metric (The result of annual returns from 2001-2015), annual return for all history(2001-2015) is 25%, annual return for backtest period (2001-2003) is 56%, and annual return for out of sample (OOS)

period (2001-2015) is 18 %, lower than the backtest period. It indicates that AI agent, learning online, can learn to trade from data during the out-of-sample period quite well even though the AI agent never sees out-of-sample data before, and we can expect the performance of backtesting or in-sample period will be higher than the performance during out-of-sample (OOS) period.

For the cumulative return, cumulative return for all history (2001-2015) is 2,685%, cumulative return for backtesting period (2001-2003) is 279%, and cumulative return for out of sample (OOS) period (2001-2015) is 634%, lower than the backtest period. It indicates that AI agent, learning online, can learn to trade from data during the out-of-sample period quite well even though the AI agent never sees out-of-sample data before, and we can expect the performance of backtesting or in-sample period will be higher than the performance during out-of-sample (OOS) period.

For annual volatility, annual volatility for all history (2001-2015) is 52%, annual volatility for backtesting period (2001-2003) is 82%, and annual return for out of sample (OOS) period (2001-2015) is 41%. This implication for the effect of volatility on the returns performance is about the stability of returns. If there are high returns, but high volatility, it can indicate the better metric to use for measuring performance is Sharpe ratio that takes the volatility into consideration.

For Sharpe ratio, Sharpe ratio for all history (2001-2015) is 0.68, annual return for backtesting period (2001-2003) is 0.94, and annual return for out of sample (OOS) period (2001-2015) is 0.61. It indicates the interesting performance perspective regarding the risk-adjusted return. Even though The AI agent return performance is quite satisfactory, the AI performance is still not very satisfactory if we take both return and risk perspective together. It is a common high-risk-high-return consequence when one developing a trading system.

4.5.1 Eurusd tear sheet

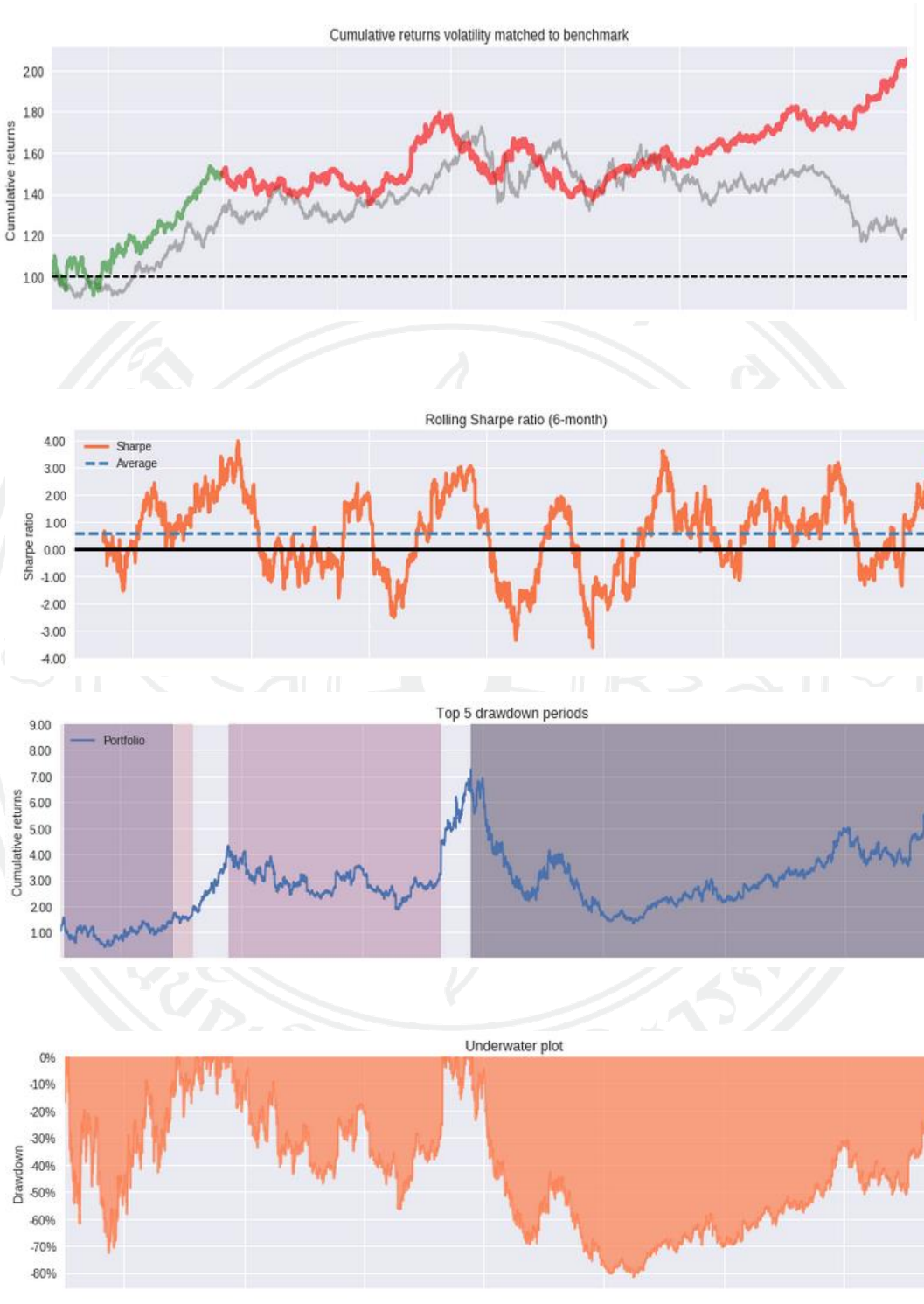


Figure 4.6 Performance Tear Sheet of AI Agent Learn to Trade EURUSD

From above figure, cumulative returns volatility matched to benchmark shows the AI performance compare to buy-and-hold (benchmark), we can visually see that the AI can learn better when getting more data.

Next picture shows the plot of 6-month rolling Sharpe ratio. We can see that rolling shape for six months is positive and close to 1.

Next picture shows top 5 drawdown periods and underwater plot; we can visually see that sometimes AI still experiences a hard time, getting the consecutive losing trades. It indicates that we should be aware of stability of returns in not only Sharpe ratio but also the drawdown as well.

We can visualize that the majority of monthly returns are positive. Most of the positive monthly returns are in the period of backtesting period. The performance during this period could be too good to be true because of curve-fitting effect.

Annual returns during 2001, 2004, 2006, 2008, 2009, and 2014 are negative. We could not see the learning pattern from this plot.

Distribution of monthly returns shows that the mean of monthly return is positive. It shows the statistical edge of the AI trading system when compared to mean of daily returns which are nearly zero.

4.5.2 Hypothesis testing for EURUSD_Agent

Paired t-Test will be used to test the hypothesis

1. AI Agent vs. buy-and-hold strategy using ***Sharpe ratio*** to test the hypothesis

H0: There is no significant difference between the performance of AI agent and the performance of the buy-and-hold strategy

H1: There is a significant difference that the performance of AI agent is superior to the performance of the buy-and-hold strategy

Table 4.2 Paired t-test result for EURUSD AI agent vs. BH (buy-and-hold) using Sharpe Ratio

t-Test: Paired Two Sample for Means		
	Sharpe _agent	**Sharpe _BH**
Mean	0.879333333	0.175133333
Variance	1.305220952	1.504498552
Observations	15	15
Pearson Correlation	0.370981418	
Hypothesized Mean Difference	0	
df	14	
t Stat	2.050010818	
P(T<=t) one-tail	0.029789104	
t Critical one-tail	1.761310136	
P(T<=t) two-tail	0.059578207	
t Critical two-tail	2.144786688	

From above table, Sharpe ratio mean of AI agent is 0.879(variance = 1.305), and Sharpe ratio mean of buy-and-hold strategy is 0.175 (variance =1.504). Both of Sharpe ratio have a positive correlation (0.37). It is significant that Sharpe ratio of the agent is superior to Sharpe ratio of buy-and-hold strategy ($P(T<=t)$ *one-tail=0.029*, $p < 0.05$)

Result: *There is a significant difference that the performance of AI agent is superior to the performance of the buy-and-hold strategy.*

2. AI Agent vs. buy-and-hold strategy using ***Annual return*** to test the hypothesis

H0: There is no significant difference between the performance of AI agent and the performance of the buy-and-hold strategy

H1: There is a significant difference that the performance of AI agent is superior to the performance of the buy-and-hold strategy

Table 4.3 Paired t-Test result for EURUSD AI agent vs. BH (buy-and-hold) using Annual Return

t-Test: Paired Two Sample for Means		
	Annual Returns_agent	**Annual Returns_BH**
Mean	43.88866667	1.466
Variance	5056.348212	108.3599257
Observations	15	15
Pearson Correlation	0.477950253	
Hypothesized Mean Difference	0	
df	14	
t Stat	2.461020542	
P(T<=t) one-tail	0.013727607	
t Critical one-tail	1.761310136	
P(T<=t) two-tail	0.027455215	
t Critical two-tail	2.144786688	

From above table, annual return mean of AI agent is 43.88 (variance = 5056.34) and annual return mean of buy-and-hold strategy is 1.46 (variance =108.35). Both of annual return have a positive correlation (0.47). It is significant that an annual return of the agent is superior to an annual return of buy-and-hold strategy (*P(T<=t) one-tail=0.013*, $p < 0.05$).

Result: *There is a significant difference that the performance of AI agent is superior to the performance of the buy-and-hold strategy.*

3. AI agent vs. CTA (experienced trader) using ***Annual return*** to test the hypothesis

H0: There is no significant difference between the performance of AI agent and the performance of CTA

H1: There is a significant difference that the performance of AI agent is superior to the performance of CTA

Table 4.4 Paired t-test result for EURUSD AI agent using Annual Return

t-Test: Paired Two Sample for Means		
	Annual Returns	**Annual Returns_CTA**
Mean	43.88866667	3.934666667
Variance	5056.348212	28.88141238
Observations	15	15
Pearson Correlation	-0.035775111	
Hypothesized Mean Difference	0	
df	14	
t Stat	2.164144073	
P(T<=t) one-tail	0.024114189	
t Critical one-tail	1.761310136	
P(T<=t) two-tail	0.048228379	
t Critical two-tail	2.144786688	

From above table, Annual return mean of AI agent is 43.88(variance = 5056.34) and Annual return mean of CTA is 3.93 (variance =28.88). Both of annual return has a negative correlation (-0.035). It is significant that an annual return of agent is superior to an annual return of buy-and-hold strategy ($P(T<=t)$ *one-tail=0.024*, $p < 0.05$)

Result: *There is a significant difference that the performance of AI agent is superior to the performance of CTA*

4.5.3 Summary of AI agent learn to trade EURUSD

1. AI outperforms buy-and-hold strategy (use Sharpe)
2. AI outperforms buy-and-hold strategy (use annual returns)
3. AI outperforms CTA (use annual returns)

4.6 USDJPY Result

Table 4.5 Performance table for AI Agent learn to trade USDJPY

```
Entire data start date: 2001-01-03
Entire data end date: 2015-12-30


Out-of-Sample Months: 143
Backtest Months: 35
```

Performance statistics	All history	Backtest	Out of sample
annual_return	0.19	0.17	0.20
cum_returns_final	12.87	0.58	7.80
annual_volatility	0.32	0.39	0.31
sharpe_ratio	0.70	0.58	0.74

From above table, it shows the vital metric (The result of annual returns from 2001-2015), annual return for all history(2001-2015) is 19%, annual return for backtest period (2001-2003) is 17%, and annual return for out-of-sample (OOS) period (2001-2015) is 20 %, higher than the backtest period. It indicates that AI agent, learning online, can learn to trade from data during the out-of-sample period quite well even though the AI agent never sees out-of-sample data before.

For the cumulative return, cumulative return for all history (2001-2015) is 1,287%, cumulative return for backtesting period (2001-2003) is 58%, and cumulative return for out of sample (OOS) period (2001-2015) is 780%, higher than the backtest

period. It indicates that AI agent, learning online, can learn to trade from data during the out-of-sample period quite well even though the AI agent never sees out-of-sample data before.

For annual volatility, annual volatility for all history (2001-2015) is 32%, annual volatility for backtesting period (2001-2003) is 39%, and annual return for out of sample (OOS) period (2001-2015) is 31%. This implication for the effect of volatility on the return performance is about the stability of returns. If there are high returns, but high volatility, it can indicate the better metric to use for measuring performance is Sharpe ratio that takes the volatility into consideration.

For Sharpe ratio, Sharpe ratio for all history (2001-2015) is 0.70, annual return for backtesting period (2001-2003) is 0.58, and annual return for out of sample (OOS) period (2001-2015) is 0.74. It indicates the interesting performance perspective regarding the risk-adjusted return. Even though The AI agent return performance is quite satisfactory, the AI performance is still not very satisfactory if we take both return and risk perspective together. It is a common high-risk-high-return consequence when one developing a trading system.

4.6.1 Usdjpy tear sheet

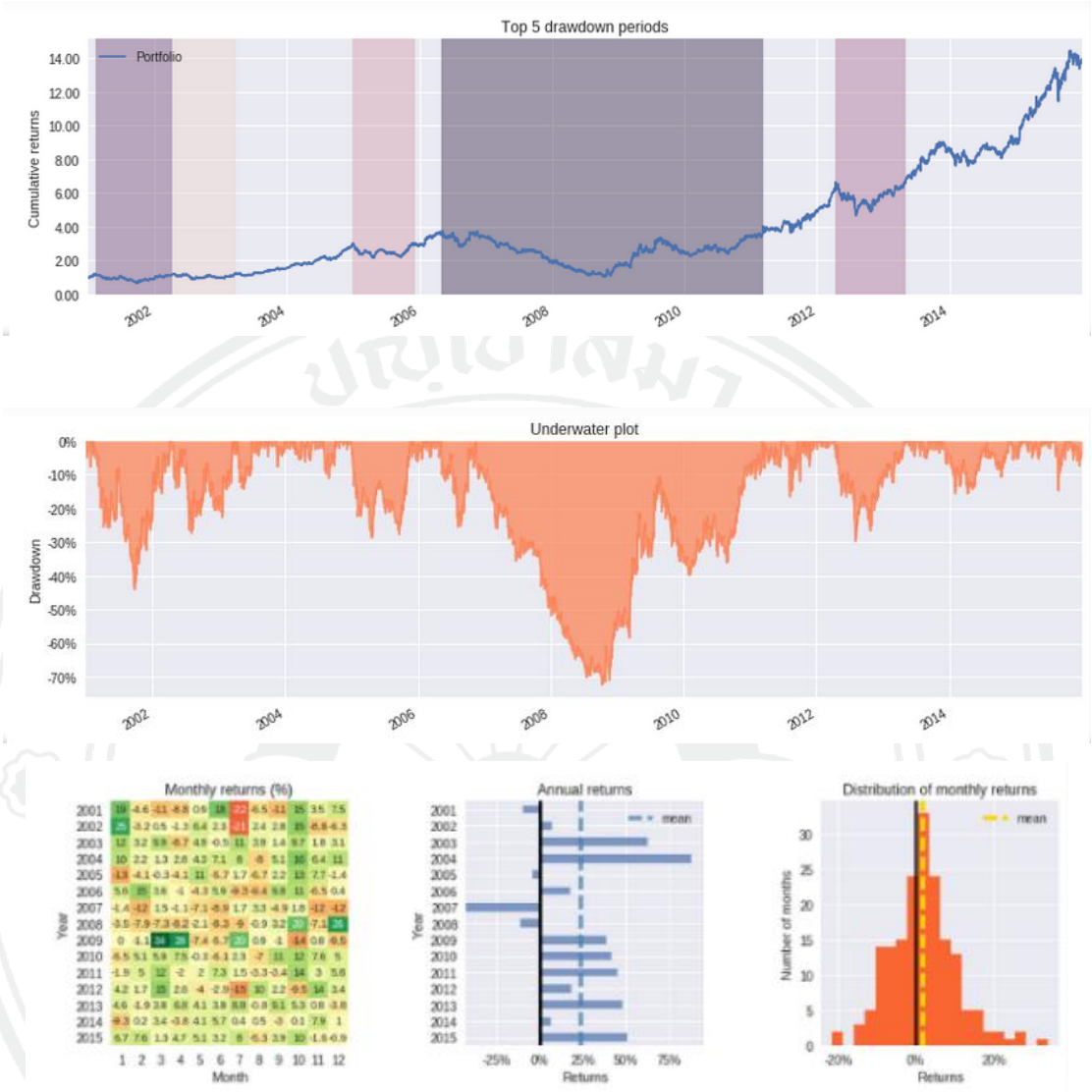


Figure 4.7 Performance Tear Sheet of AI Agent learn to trade USDJPY

From above figure, cumulative returns volatility matched to benchmark shows the AI performance compare to buy-and-hold (benchmark), we can visually see that the AI can learn better when getting more data and outperform the benchmark.

Next picture shows the plot of 6-month rolling Sharpe ratio. We can see that rolling shape for six months is positive and close to 1.

Next picture shows top 5 drawdown periods and underwater plot; we can visually see that sometimes AI still experiences the hard time (2007-2011), getting the consecutive losing trades. It indicates that we should be aware of stability of returns by looking at not only Sharpe ratio but also the drawdown as well.

We can see that the majority of monthly returns are positive. Most of the positive monthly returns are in the later test period. It implies that the when AI agent learns more from data, the AI agent will become smarter.

Annual returns during 2001, 2005, 2007 and 2008 are negative. We could not see the learning pattern from this plot.

Distribution of monthly returns shows that the mean of monthly return is positive. It shows the statistical edge of the AI trading system when compared to mean of daily returns which are nearly zero.

4.6.2 Hypothesis testing for USDJPY_Agent

Paired t-Test will be used to test the hypothesis

1. AI Agent vs. buy-and-hold strategy using ***Sharpe ratio*** to test the hypothesis

H0: There is no significant difference between the performance of AI agent and the performance of the buy-and-hold strategy

H1: There is a significant difference that the performance of AI is superior to the performance of the buy-and-hold strategy

Table 4.6 Paired t-test result for USDJPY AI Agent vs. BH (buy-and-hold) Using Sharpe Ratio

t-Test: Paired Two Sample for Means		
	Sharpe _agent	**Sharpe_BH**
Mean	1.136666667	0.158666667
Variance	1.88862381	1.231969524
Observations	15	15
Pearson Correlation	-0.172514948	
Hypothesized Mean Difference	0	
df	14	
t Stat	1.983459045	
P(T<=t) one-tail	0.033641974	
t Critical one-tail	1.761310136	
P(T<=t) two-tail	0.067283947	
t Critical two-tail	2.144786688	

From above table, Sharpe ratio mean of AI Agent is 1.13(variance = 1.88) and Sharpe ratio mean of buy-and-hold strategy is 0.15 (variance =1.23). Both of Sharpe ratios have a negative correlation (-0.17). It is significant that Sharpe ratio of the agent is superior to Sharpe ratio of buy-and-hold strategy ($P(T<=t)$ *one-tail=0.033*, $p < 0.05$)

Result: *There is a significant difference that the performance of AI agent is superior to the performance of the buy-and-hold strategy*

2. AI Agent vs. buy-and-hold strategy using ***Annual return*** to test the hypothesis

H0: There is no significant difference between the performance of AI agent and the performance of the buy-and-hold strategy

H1: There is a significant difference that the performance of AI agent is superior to the performance of the buy-and-hold strategy

Table 4.7 Paired t-test result for USDJPY AI Agent vs. BH (buy-and-hold) Using Annual Return

t-Test: Paired Two Sample for Means		
	Annual Return_agent	**Annual Returns_BH**
Mean	26.732	0.925333333
Variance	2255.99946	142.8156552
Observations	15	15
Pearson Correlation	0.076078354	
Hypothesized Mean Difference	0	
df	14	
t Stat	2.078459449	
P(T<=t) one-tail	0.028269352	
t Critical one-tail	1.761310136	
P(T<=t) two-tail	0.056538704	
t Critical two-tail	2.144786688	

From above table, annual return mean of AI Agent is 26.73(variance = 2255.99) and Annual return mean of buy-and-hold strategy is 0.92 (variance =142.81). Both of

annual return have a positive correlation (0.07). It is significant that the annual of return of the AI agent is superior to the annual return of buy-and-hold strategy (*P (T<=t) one-tail=0.028, p* < 0.05).

Result: *There is a significant difference that the performance of AI agent is superior to the performance of the buy-and-hold strategy*

3. AI Agent vs. CTA (experienced trader) using ***Annual return*** to test the hypothesis

H0: There is no significant difference between the performance of AI agent and the performance of CTA

H1: There is a significant difference that the performance of AI agent is superior to the performance of CTA

Table 4.8 Paired t-test result for USDJPY AI Agent Using Annual Return

t-Test: Paired Two Sample for Means		
	Annual Returns_agent	**Annual Returns_CTA**
Mean	26.732	3.934666667
Variance	2255.99946	28.88141238
Observations	15	15
Pearson Correlation	-0.474885183	
Hypothesized Mean Difference	0	
df	14	
t Stat	1.756304525	
P(T<=t) one-tail	0.0504389	
t Critical one-tail	1.761310136	
P(T<=t) two-tail	0.1008778	
t Critical two-tail	2.144786688	

From above table, annual return mean of AI agent is 26.73(variance = 2255.99) and annual return mean of CTA is 3.93 (variance =28.88). Both of annual returns have a negative correlation (-0.47). There was not significant that the annual return of agent is

superior to the annual return of buy-and-hold strategy (*P(T<=t) one-tail=0.0504, p >* 0.05)

Result: *There is no significant difference that the performance of AI agent is superior to the performance of CTA*

4.6.3 Summary of AI agent learn to trade USDJPY

1. AI outperform buy-and-hold strategy (use Sharpe)
2. AI outperforms buy-and-hold strategy (use Annual Returns)
3. Not significant that the performance of AI agent outperforms CTA (use returns)

4.7 XAUUSD (Gold) Result

Table 4.9 Performance table for AI Agent learn to trade USDJPY

```
Entire data start date: 2001-01-03
Entire data end date: 2015-12-30


Out-of-Sample Months: 71
Backtest Months: 107
```

Performance statistics	All history	Backtest	Out of sample
annual_return	0.07	0.04	0.12
cum_returns_final	1.90	0.46	0.99
annual_volatility	0.19	0.22	0.14
sharpe_ratio	0.47	0.30	0.88

From above table, it shows the critical metric (The result of annual returns from 2001-2015), annual return for all history(2001-2015) is 7%, annual return for backtest period (2001-2003) is 4%, and annual return for out-of-sample (OOS) period (2001-2015) is 12 %, higher than the backtest period. It indicates that AI agent, learning online, can learn to trade from data during the out-of-sample period quite well even though the AI agent never sees out-of-sample data before.

For the cumulative return, cumulative return for all history (2001-2015) is 190%, cumulative return for backtesting period (2001-2003) is 46%, and cumulative return for out of sample (OOS) period (2001-2015) is 99%, higher than the backtest period. It indicates that AI agent, learning online, can learn to trade from data during the out-of-sample period quite well even though the AI agent never sees out-of-sample data before.

For annual volatility, annual volatility for all history (2001-2015) is 19%, annual volatility for backtesting period (2001-2003) is 22%, and annual return for out of sample (OOS) period (2001-2015) is 14%. This implication for the effect of volatility on the return performance is about the stability of returns. If there is a high return, but high volatility, it can indicate the better metric to use for measuring performance is Sharpe ratio that takes the volatility into consideration.

For Sharpe ratio, Sharpe ratio for all history (2001-2015) is 0.47, annual return for backtesting period (2001-2003) is 0.30, and annual return for out of sample (OOS) period (2001-2015) is 0.88. It indicates the interesting performance perspective regarding the risk-adjusted return. Even though The AI agent return performance is entirely satisfactory, the AI performance is still not very satisfactory if we take both return and risk perspective together. It is a common high-risk-high-return consequence when one developing a trading system.

4.7.1 Xauusd tear sheet

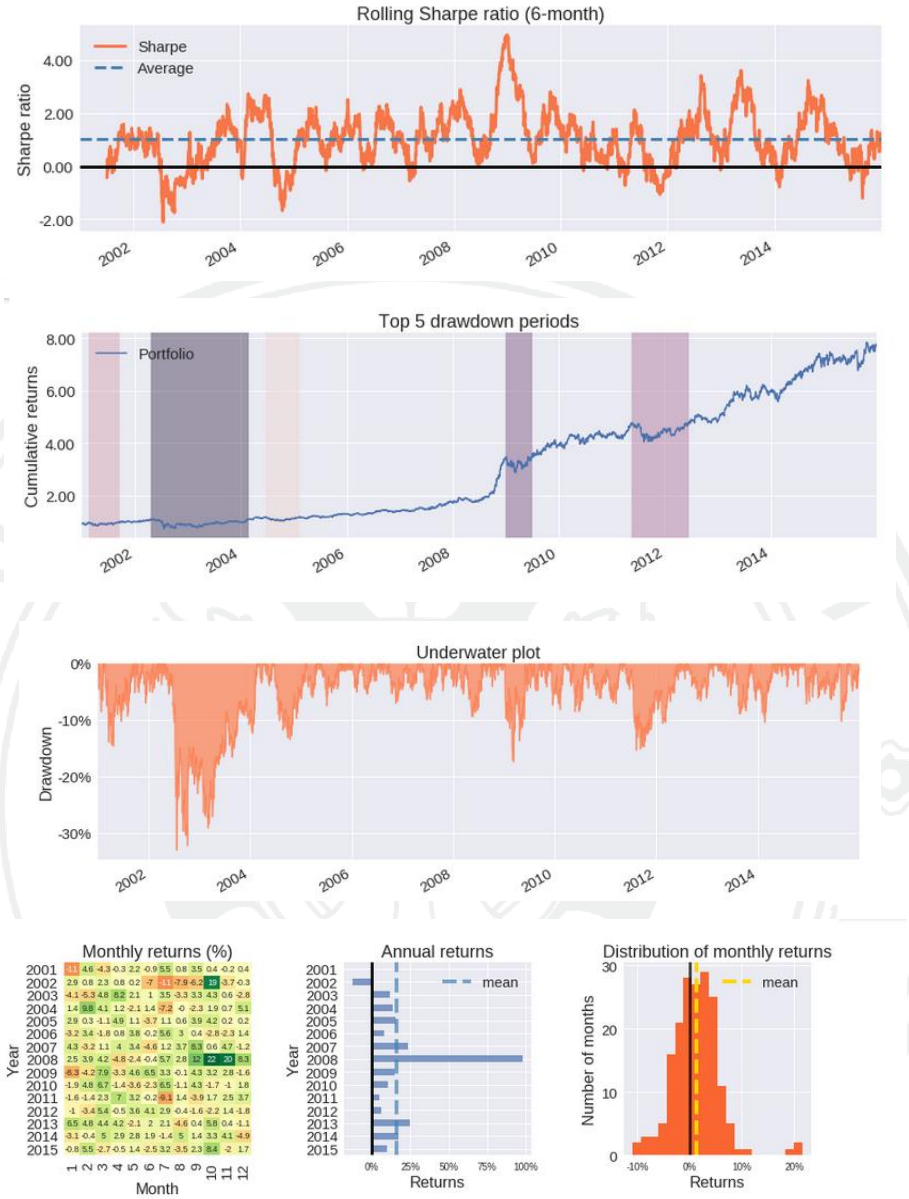


Figure 4.8 Performance Tear Sheet of AI Agent learn to trade XAUUSD

From above figure, cumulative returns volatility matched to benchmark shows the AI performance, compared to buy-and-hold (benchmark), we can visually see that the AI can learn better when getting more data and outperform the benchmark, especially during the later test period.

Next picture shows the plot of 6-month rolling Sharpe ratio. We can see that rolling shape for six months is positive and above 1.

Next picture shows top 5 drawdown periods and underwater plot; we can visually see that sometimes AI still experiences the hard time (2003-2004), getting the consecutive losing trades. Comparing to EURUSD and USDJPY, this XAUUSD agent shows lower and shorter period of drawdown because the volatility of EURUSD (52%) and USDJPY (32%) are higher than the volatility of XAUUSD. It implies that AI agent will learn better if there is lower volatility.

We can see that the majority of monthly returns are positive. Most of the positive monthly returns are in the later test period. It implies that the when AI agent learns more from data, the AI agent will become smarter.

Annual returns during 2002 are negative (backtesting period). We can also see the pattern that AI learn more from data.

Distribution of monthly returns shows that the mean of monthly return is positive. It shows the statistical edge of the AI trading system when compared to mean of daily returns which are nearly zero.

4.7.2 Hypothesis testing for XAUUSD_Agent

A paired t-test will be used to test the hypothesis

1. AI Agent vs. buy-and-hold strategy using ***Sharpe ratio*** to test the hypothesis

H0: There is no significant difference between the performance of AI agent and the performance of the buy-and-hold strategy

H1: There is a significant difference that the performance of AI agent is superior to the performance of the buy-and-hold strategy

Table 4.10 Paired t-test result for XAUUSD AI Agent vs. BH (buy-and-hold) Using Sharpe Ratio

t-Test: Paired Two Sample for Means		
	Sharpe_agent	**Sharpe_BH**
Mean	0.956666667	0.636666667
Variance	0.63272381	0.822838095
Observations	15	15

t-Test: Paired Two Sample for Means		
	Sharpe_agent	**Sharpe_BH**
Pearson Correlation	-0.251361575	
Hypothesized Mean Difference	0	
df	14	
t Stat	0.919100153	
P(T<=t) one-tail	0.186803901	
t Critical one-tail	1.761310136	
P(T<=t) two-tail	0.373607802	
t Critical two-tail	2.144786688	

From above table, Sharpe ratio mean of AI gent is 0.95(variance = 0.63) and Sharpe ratio mean of buy-and-hold strategy is 0.63 (variance =0.82). Both of Sharpe ratios have a negative correlation (-0.25). It is not significant that Sharpe ratio of AI agent is superior to Sharpe ratio of buy-and-hold strategy ($P(T<=t)$ *one-tail=0.18,* $p > 0.05$)

Result: *There is no significant difference that the performance of AI agent is superior to the performance of the buy-and-hold strategy*

2. AI Agent vs. Buy&Hold using ***Annual return*** to test the hypothesis

H0: There is no significant difference between the performance of AI agent and the performance of the buy-and-hold strategy

H1: There is a significant difference that the performance of AI agent is superior to the performance of the buy-and-hold strategy

Table 4.11 Paired t-test result for XAUUSD AI Agent vs. BH (buy-and-hold) Using Annual Return

t-Test: Paired Two Sample for Means		
	Annual Return_agent	**Annual Returns_BH**
Mean	7.707333333	10.67666667

t-Test: Paired Two Sample for Means		
	Annual Return_agent	**Annual Returns_BH**
Variance	167.2801638	269.7047238
Observations	15	15
Pearson Correlation	-0.099990996	
Hypothesized Mean Difference	0	
df	14	
t Stat	-0.52520322	
P(T<=t) one-tail	0.303829983	
t Critical one-tail	1.761310136	
P(T<=t) two-tail	0.607659966	
t Critical two-tail	2.144786688	

From above table, Annual return mean of AI agent is 7.70(variance = 167.28) and Annual return mean of buy-and-hold strategy is 10.67 (variance =269.70). Both of annual return has a negative correlation (-0.09). It is not significant that the annual return of the agent is superior to the annual return of buy-and-hold strategy (*P(T<=t) one-tail=0.30, p* > 0.05).

Result: *There is no significant difference that the performance of AI agent is superior to the performance of the buy-and-hold strategy*

3. AI Agent vs. CTA (experienced trader) using ***Annual return*** to test the hypothesis

H0: There is no significant difference between the performance of AI agent and the performance of CTA

H1: There is a significant difference that the performance of AI agent is superior to the performance of CTA

Table 4.12 Paired t-test result for XAUUSD AI Agent Using Annual Return

t-Test: Paired Two Sample for Means		
	Annual Returns_agent	**Annual Returns_CTA**
Mean	7.707333333	3.934666667
Variance	167.2801638	28.88141238
Observations	15	15
Pearson Correlation	-0.48593098	
Hypothesized Mean Difference	0	
df	14	
t Stat	0.89976294	
P(T<=t) one-tail	0.191731111	
t Critical one-tail	1.761310136	
P(T<=t) two-tail	0.383462222	
t Critical two-tail	2.144786688	

From above table, annual return mean of AI agent is 7.70(variance = 167.28) and annual return mean of CTA is 3.93 (variance =28.88). Both of annual return has a negative correlation (-0.48). There was not significant that the annual return of agent is superior to the annual return of CTA (*P(T<=t) one-tail=0.19*, $p > 0.05$).

Result: *There is no significant difference that performance of AI agent is superior to the performance of CTA*

4.7.3 Summary of AI agent learn to trade XAUUSD (Gold)

1. Not significant that the performance of AI agent outperforms buy-and-hold strategy (use Sharpe)
2. Not significant that the performance of AI agent outperforms buy-and-hold strategy (use returns)
3. Not significant that the performance of AI agent outperforms CTA (use returns)

CHAPTER 5

CONCLUSION

This chapter mainly focuses on the underlying assumption of this study, limitation of the study, findings, contributions and the suggestion for the future research.

5.1 Factors that Potentially Affect the Result

The primary assumption of this study is that, if there is a pattern in the data, the machine or AI should be able to detect the underlying pattern and make a trading decision better than the human expert whom we believe that they are vulnerable to bias developed from their own experience and knowledge. The key to understanding the modeling that we used to test the market depends on following factors

5.1.1 The Deep Learning Algorithm

In this paper, we would like to explore how to apply DQN (Deep Q Learning), which is one of the approaches for reinforcement learning. There are several parameters related to DQN that determine the performance of our algorithm. For example, the ratio between test/train set can show the different performance. More training data mean AI can learn several patterns and can adapt to several trading environments. If the data that we use to train AI and the data we use to test AI are in the same pattern, there is more likely that the performance will be better than training with different patterns.

The algorithm itself is complicated to be replicated due to randomness which is nature of the deep network. If we test the neural network model, in each time, it will show the result differently. Sometimes, the model will not converge, but the performance is getting better. When start running the model, the model will randomly

initialize the data and update the weight of the deep network by backpropagation. The model will usually update the weight until finding the global maxima/minima or the model become converged.

The amount we choose for the batch size to update the weight for the DQN algorithm also affect the training speed and also the performance of the deep neural network. When we choose the small batch size to update weight the training speed will be faster. However, the accuracy and the performance will become lower.

The learning rate can be too high/low. The typical learning rate is between 0.1-0.00001. If we set learning rate too low, when we train several epochs, the loss for DQN will not decrease that means optimal process stuck in a local minimum. If we set the learning rate too high, the loss will become NAN. Setting the appropriate learning rate is trial and error process.

5.1.2 Mapping trading problem to reinforcement learning problem

When we map the trading problem to reinforcement problem, we need to select the states which determine what AI will see the environment or perceive the world. We subjectively choose the indicators which we believe could detect some profitable patterns. Firstly, we tried to use input with close and moving average. The differences among moving average can signal the cross of the two moving average which mostly used as a trading signal. The difference between close and moving average was also chosen as a signal as well. Based on the result we test for sine wave data, we believe that if we could transform data to a pure sine wave, AI will learn how to trade as well. We then select one of the cyclic indicators which was sine wave indicators, and the performance got better once we added such indicators. So we believe that if we could transform the historical price into more predictable such as sine wave, we could train AI with advantage.

We have to identify reward function for the AI to learn and set up gamma which can determine how much we consider more for long-term or short-term profits. Gamma can be set between 0-1. Gamma is equal to 1 means we weight 100% on long-

term profits. Gamma is equal to 0 means we weight 100% on short-term or immediate profits.

Different reward function could result in different performance. If we choose reward function as winning rate or reward to risk, we could train AI with finer tune objective. For example, if we take more risk with the expectation of higher returns we could set to reward to risk as reward function to win significant profit but small winning rate, but our Sharpe ratio will be lower. If we want to be more conservative with the risk, we could use winning rate as a reward function to win more times with small profits.

5.1.3 Deep neural network architecture

The architecture of the deep neural network also contributes to the performance of the AI because the deep network is used as the function approximation to update the weight of each node after calculation of loss function. The small brain will typically result in lower performance compared to the bigger brain with the more hidden layer. However, a non-complex problem such as predictable pattern will be indifferent between small and big brain. For the large-complex problem, the bigger brain tends to be better.

5.1.4 Trading objective

There are several factors contribute to trading objectives such as risk tolerance of the investors, investment policy, account size, etc. Most institutional investors prefer low risk, but steady returns due to the account size which is very large. Hedge fund and investment company could take more risk compared to traditional institutional investors who are more conservative. Typical hedge fund strategies are to maximize Sharpe ratio to increase more leverage for higher returns. Individual investors, however, can take more risk compensated by higher returns.

Due to different objectives, Ai can be trained by different reward function such as maximized Sharpe ratio, profits, winning percentage, reward to risk ratio, annual returns, etc. When we change the reward function, the AI performance will change as well according to reward function.

5.2 Findings of this study

The findings for EURUSD AI agent, we found that the AI agent significantly outperforms buy-and-hold both using Sharpe ratio and annual returns. It is the indicator that AI can learn to trade from the data. If we look at the benchmark which is buy-and-hold, it is clear that if we hold the EURUSD for longer than ten years, we will get almost nothing. It is due to the nature of fiat currency that is not suitable to be the investment class. We would suggest that Trading by AI would be better than holding the currency.

The findings for EURUSD AI agent, we found that the AI agent significantly outperforms CTA (experienced trader) using annual returns. It does not mean that AI agent is undoubtedly better than a human expert. The differences between machine and human are emotion. AI can execute the trade without the emotion of fear or greed. When AI see the profitable pattern, it will not hesitate to take action. Therefore, we would suggest that trading by AI would be better if we care more about annual returns (we did not use Sharpe ratio because data is not available).

The findings for USDJPY AI agent, we found that the AI agent significantly outperforms buy-and-hold strategy both using Sharpe ratio and annual returns. It is the indicator that AI can learn to trade from the data. If we look at the benchmark which is buy-and-hold, it is clear that if we hold the USDJPY for longer than ten years, we will get loss slightly not to mention the inflation rate. It is due to the nature of fiat currency that is not suitable to be the investment class. We would suggest that trading by AI would be better than holding the currency.

The findings for USDJPY AI agent, we found that the AI agent does not significantly outperform CTA (experienced trader) using annual returns. In this case, we cannot be sure that AI is better than a human expert when we compare the returns even though the mean returns of the AI is better than CTA. However, the standard deviation also very much higher as well. Therefore, we would suggest that we need

more data to test this hypothesis again. We could not suggest that which one is better over another (we did not use Sharpe ratio because data is not available).

The findings for XAUUD AI agent, we found that the AI agent does not significantly outperform buy-and-hold both using Sharpe ratio and annual returns. However, if we look at the performance of AI itself, AI show some capability to learn from data. If we look at the benchmark which is buy-and-hold , it is clear that if we hold XAUUSD (Gold) for long-term investment, in some period, the performance will be better than AI. It is due to the nature of gold as a safe haven that is considered to be investment asset. We would suggest that trading by AI compare to holding gold is not significantly different.

The findings for XAUUSD AI agent, we found that the AI agent does not significantly outperform CTA (experienced trader) using annual returns. In this case, we cannot be sure that AI is better than a human expert when we compare the returns even though the mean returns of the AI is better than CTA. However, the standard deviation also very much higher as well. Therefore, we would suggest that we need more data to test this hypothesis again. We could trade prefer AI over Holding Gold if we can tolerate the high volatility of AI performance. We could not suggest that which on is better over another (We did not use Sharpe ratio because data is not available).

5.3 Limitation

1. Available data

To train AI, we need a massive amount of data to learn how to trade. However, we found limitation in that we can access to only historical price data (Open, High, Low, and Close). We could not access the valuable data such as actual volume and order flow between interbank orders. Those data are expensive and available only for the giant hedge fund or quant firm. The researcher believes that, with more complete data, the AI will learn better and make a more informed trading decision. With more data, we will try to test and may end up with a trading portfolio of several currency

pairs combined with several assets such as gold, silver, oil, etc. The robust portfolio trading will be possible to create if we can access to more data for AI. For investment, diversification is the closest thing to free lunch so if more data available we could diversify more of our portfolio.

2. Computing power

Training the deep neural network is quite expensive in that it consumes time and computing power for complicated calculation. We all know that deep neural network can surpass the performance of almost all machine learning algorithm. However, it comes with the cost we have to pay. In this study, we have to spend 2-3 days running computers to perform parameter tuning until the acceptable parameters. The researchers have tried and tweak countless possibilities of the almost all parameters. Typically, a bigger brain with more hidden layers would be able to detect more complicated pattern and perform complex computation.

We could, however, take more data (>15 years) to validate the performance of the model if we could have more computing power in the future. Cloud computing and GPU acceleration for training the model will be available economically in very near future. We hope that we could test more again when the resource is economically available.

3. Trading assumption

Even though AI performance is quite satisfactory, it does not mean we should jump into the trading with real money because the study just ignited the possibility that we can train AI to trade live in the future. However, when trade lives, we should be aware of how to set up risk management system to protect from unexpected events such as gap opening/ central bank intervention, nonfarm payroll, news, etc. This study will not include transaction cost and spread which could contribute to the big difference between backtesting and real trade. Moreover, our study simplifies reality for trading lives such as transaction cost, the spread between bid-ask, entry price, type of orders. It would indicate that there is plenty of room for improvement in the future.

5.4 Academic Contribution

From this study, there are several contributions to the academic study such as the application of artificial intelligence in algorithmic trading system development is a desirable method to replace the human-decision-making system in that the computer can read hidden profitable price pattern better than human and computer can execute the trade swiftly and accurately. AI can be the best candidate to replace the human. Academically, more studies can be conducted to compare the performance of human and AI.

This research is to touch only one specific area of investment and financial problem. Reinforcement learning technique can be applied in several areas of finance under the condition that there are enough data to learn from so that AI can make the better decision because of the adaptive capability to the environment. One of the applications which is very attractive to study is to use reinforcement learning for portfolio management problems. The AI agent will find the appropriate weight for each asset in the portfolio. AI will make decisions for not only about the asset to trade but also when to rebalance the portfolio (Moody et al., 1998).

Moreover, this study supports the opponents of EMH. It is possible to develop the trading system to outperform buy-and-hold in the long run. For EURUSD and USDJPY, the performance of AI agent surpassed the buy-and-hold significantly. However, the in-depth test and live trade should be conducted before jumping to the conclusion due to several of limitations.

5.5 Practical Contribution

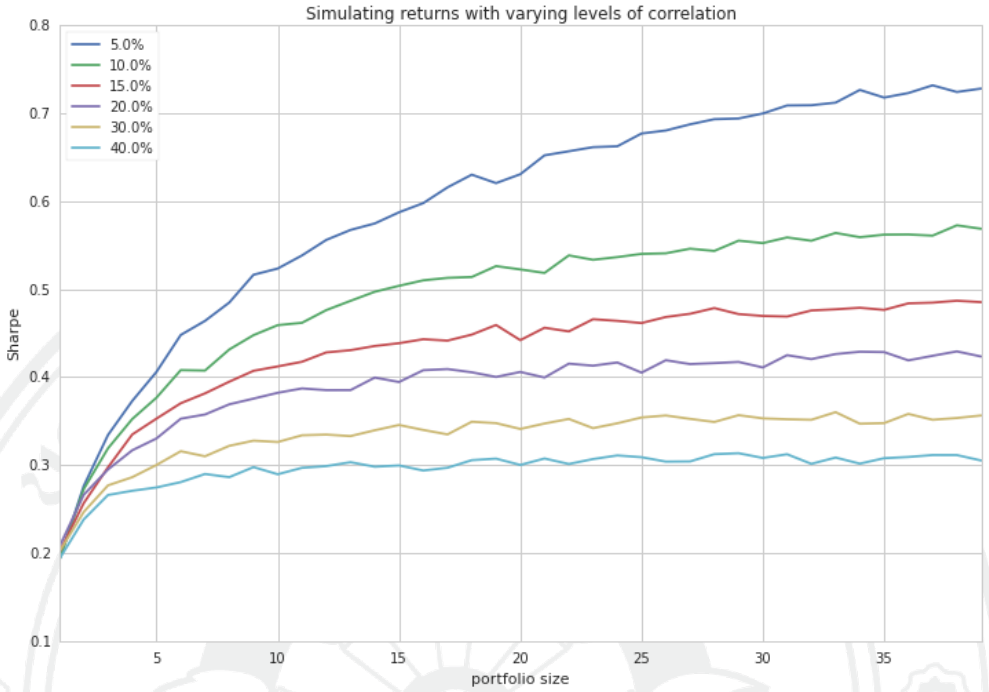


Figure 5.1 Increase in Sharpe Ratio compared to portfolio size with different levels of correlation
Source: Wiecki (2015a)

From this study, we can find the new, alternative to create return streams that have a low correlation to each other using the AI-generated trading system. As we can see from the results such as the annual return of AI agent and CTA have a very low correlation (-0.03) so that we could create the min-correlation, risk-diversified portfolio for stable returns. From the figure 5.1, it shows that if we can add more return streams with low correlation, we can increase Sharpe ratio. We can use several AIs to create several returns streams that are not correlated to each other.

5.6 Suggestion for Future Research

1. In the future research, we hope that the computing power will be available for training deep neural network with lower cost. If it is available, the possibility to try something new is endless. At the current time, there is an open-source project who

try to create peer-to-peer, sharing computing power. The supercomputing power will be much cheaper and faster.

2. We could try all possible states. We could input several thousand indicators and more of fundamental data. Moreover, more complex cyclic and time series analysis will be added on to test the model such as singular spectrum analysis to decompose the price series into the cycle.
3. We could try to add some filters such as Hidden Markov model. We could separately train another model to extract market mode only. Hidden Markov model will help us to identify the satisfactory market situation for each specific trading strategies.
4. We could use the larger brain for AI. We can add more layers for AI to increase the capability to learn more form data. We can tweak all the topology of several deep learning types such as convolutional neural network, GRU, GAN to beat the performance of the human expert.
5. We could combine several AIs to become super AI for the portfolio. We could train AI separately to identify what market AI is best for, to identify what market mode AI is best for, and to identify the correlation between all AI.
6. In the future research, we could extend by making the real live trade with some predetermine risk parameters such as risk per trade, add stop loss, add more advanced pending order, add more scale in/scale out an algorithm to teach AI to learn the more complex trading process.
7. During the completion time of this study, there is another huge leap from the company named “Deepmind” with alpha go zero. The AI that can learn by itself, not limited by the domain they train from, AI that can learn without a human expert to train for them. In the future research, it possible to follow what deep mind has achieved and trading for the better result.

BIBLIOGRAPHY

Alonso-González, A., Peris-Ortiz, M., & Almenar-Llongo, V. (2015). Providing Empirical Evidence from Forex Autotrading to Contradict the Efficient Market Hypothesis *New Challenges in Entrepreneurship and Finance* (pp. 71-85): Springer.

BarclayHedge, L. (2017). Barclay Currency Traders Index. from https://www.barclayhedge.com/research/indices/cta/sub/curr.html

Batten, J. A., Lucey, B. M., McGroarty, F., Peat, M., & Urquhart, A. (2018). Does intraday technical trading have predictive power in precious metal markets? *Journal of International Financial Markets, Institutions and Money, 52*, 102-113. doi: https://doi.org/10.1016/j.intfin.2017.06.005

Baur, D. G., & Glover, K. J. (2015). Speculative trading in the gold market. *International Review of Financial Analysis, 39*, 63-71. doi: https://doi.org/10.1016/j.irfa.2015.02.004

Chan, K. C., & Teong, F. K. (1995). *Enhancing technical analysis in the Forex market using neural networks.* Paper presented at the Neural Networks, 1995. Proceedings., IEEE International Conference on.

Coakley, J., Marzano, M., & Nankervis, J. (2016). How profitable are FX technical trading rules? *International Review of Financial Analysis, 45*, 273-282. doi: https://doi.org/10.1016/j.irfa.2016.03.010

Dempster, M. A. H., & Leemans, V. (2006). An automated FX trading system using adaptive reinforcement learning. *Expert Systems with Applications, 30*(3), 543-552. doi: 10.1016/j.eswa.2005.10.012

Deng, Y., Bao, F., Kong, Y., Ren, Z., & Dai, Q. (2017). Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems, 28*(3), 653-664.

Dourra, H., & Siy, P. (2002). Investment using technical analysis and fuzzy logic. *Fuzzy sets and systems, 127*(2), 221-240.

Du, X., Zhai, J., & Lv, K. (2016). Algorithm Trading using Q-Learning and Recurrent Reinforcement Learning. *positions, 1*, 1.

Dukascopy. (2017). Historical Data Feed. from https://www.dukascopy.com/swiss/english/marketwatch/historical

Ertel, W. (2018). *Introduction to artificial intelligence*: Springer.

Fama, E. F. (1995). Random walks in stock market prices. *Financial analysts journal, 51*(1), 75-80.

Farias Nazário, R. T., e Silva, J. L., Sobreiro, V. A., & Kimura, H. (2017). A literature review of technical analysis on stock markets. *The Quarterly Review of Economics and Finance, 66*, 115-126. doi: https://doi.org/10.1016/j.qref.2017.01.014

Farjam, M., & Kirchkamp, O. (2018). Bubbles in hybrid markets: How expectations about algorithmic trading affect human trading. *Journal of Economic Behavior & Organization, 146*, 248-269. doi: https://doi.org/10.1016/j.jebo.2017.11.011

Giampaoli, I., Ng, W. L., & Constantinou, N. (2009). Analysis of ultra-high-frequency financial data using advanced Fourier transforms. *Finance Research Letters, 6*(1), 47-53. doi: https://doi.org/10.1016/j.frl.2008.11.002

Gold, C. (2003). *FX trading via recurrent reinforcement learning.* Paper presented at

the Computational Intelligence for Financial Engineering, 2003. Proceedings. 2003 IEEE International Conference on.
Investopedia. (2017). What is a trading system. Retrieved 09/11/2017, from http://www.investopedia.com/university/tradingsystems/tradingsytems1.asp
Kalmus, L. P., Trojan, D. R., Mott, B., & Strampfer, J. (1987). Automated securities trading system: Google Patents.
LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436-444.
Levinson, M. (2014). *The Economist Guide To Financial Markets*: Economist book.
Malkiel, B. G. (1989). Efficient market hypothesis *Finance* (pp. 127-134): Springer.
Malkiel, B. G., & Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The journal of Finance, 25*(2), 383-417.
Moody, J., & Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE Trans Neural Netw, 12*(4), 875-889.
Moody, J., Wu, L., Liao, Y., & Saffell, M. (1998). Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting, 17*(56), 441-470.
Nasdaq. (2018). Retrieved 04/21/18, 2018, from https://www.nasdaq.com/investing/glossary/c/commodity-trading-advisor
Seth, S. (2015). The World Of High Frequency Algorithmic Trading.
Sezer, O. B., & Ozbayoglu, A. M. (2018). Algorithmic Financial Trading with Deep Convolutional Neural Networks: Time Series to Image Conversion Approach. *Applied Soft Computing*. doi: https://doi.org/10.1016/j.asoc.2018.04.024
Sharpe, W. F. (1966). Mutual fund performance. *The Journal of business, 39*(1), 119-138.
Silver, D. (2015). Deep Reinforcement Learning. from http://videolectures.net/rldm2015_silver_reinforcement_learning/
Taylor, M. P., & Allen, H. (1992). The use of technical analysis in the foreign exchange market. *Journal of international Money and Finance, 11*(3), 304-314.
Vajda, V. (2014). Could a Trader Using Only "Old" Technical Indicator be Successful at the Forex Market? *Procedia Economics and Finance, 15*, 318-325. doi: https://doi.org/10.1016/S2212-5671(14)00515-2
Veloso, M. (2017). *What is machine learning ?* Retrieved 09/12/2017, from https://www.ml.cmu.edu/
Vladimir, G., & Kabysh, A. Supporting Coherence in Multi-Agent System: Related Temporal Difference with Influence Trace.
Wang, Y., Wang, D., Zhang, S., Feng, Y., Li, S., & Zhou, Q. (2016). Deep Q-trading.
Wiecki, T. (2015a). Monte-Carlo simulations of correlated portfolios -- the quest for uncorrelated return streams.
Wiecki, T. (2015b). Prediction future returns of trading algorithm.

APPENDIX

Python Code

Sine wave test

1. **Build simulated sine wave**

```
from __future__ import print_function

import numpy as np
np.random.seed(100)
from sklearn import metrics, preprocessing

import pandas as pd
from matplotlib import pyplot as plt

# define sine wave
def load_data():
    sinewave = np.sin(np.arange(800)/20.0)
    return sinewave

sinewave  = load_data()
```

2. **Building neural network**

```
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop

model = Sequential()
model.add(Dense(4, init='lecun_uniform', input_shape=(2,)))
model.add(Activation('relu'))

model.add(Dense(4, init='lecun_uniform'))
model.add(Activation('relu'))

model.add(Dense(4, init='lecun_uniform'))
model.add(Activation('linear'))
rms = RMSprop()
model.compile(loss='mse', optimizer=rms)
```

3. Sample backtesting with btgym

```
import backtrader as bt
import backtrader.indicators as btind
import numpy as np
import scipy.signal as signal
from scipy import stats

from gym import spaces

from btgym import BTgymEnv, BTgymStrategy, BTgymDataset

from btgym.a3c import Launcher, LSTMPolicy

class MyStrategy(BTgymStrategy):

    def __init__(self, **kwargs):
        super(MyStrategy,self).__init__(**kwargs)
        self.order_penalty = 1
        self.trade_just_closed = False
        self.trade_result = None

    def notify_trade(self, trade):
        if trade.isclosed:
            # Set trade flag and result:
            self.trade_just_closed = True
            self.trade_result = trade.pnlcomm

    def get_state(self):
        T = 1e3 # amplifier
        X = np.gradient(self.raw_state, axis=0)
        X *= T
        self.state['model_input'] = X
        return self.state

    def get_reward(self):
        r = (self.broker.get_value() / self.env.broker.startingcash - 1) * 10

        if self.trade_just_closed:
            r += self.trade_result
            self.trade_just_closed = False

        if self.order_failed:
```

```
            r -= self.order_penalty
            self.order_failed = False

        return r / 20

time_embed_dim = 30

state_shape = {
    'raw_state': spaces.Box(low=-1, high=1, shape=(time_embed_dim, 4)),
    'model_input': spaces.Box(low=-100, high=100, shape=(time_embed_dim, 4))
}

MyCerebro = bt.Cerebro()

MyCerebro.addstrategy(
    MyStrategy,
    state_shape=state_shape,
    portfolio_actions=('hold', 'buy', 'sell', 'close'),
    drawdown_call=5, # max to loose, in percent of initial cash
    target_call=20,  # max to win, same
    skip_frame=10,
)

# Set leveraged account:
MyCerebro.broker.setcash(100000)
MyCerebro.broker.setcommission(commission=0.0001, leverage=1) #
commisssion to imitate spread
MyCerebro.broker.set_shortcash(False)
MyCerebro.addsizer(bt.sizers.SizerFix, stake=10000,)

MyCerebro.addanalyzer(bt.analyzers.DrawDown)

MyDataset = BTgymDataset(
    filename='../data/test_sine_wave.csv',
    start_weekdays=[0, 1, 2, 3, ],
    episode_len_days=1,
    episode_len_hours=23,
    episode_len_minutes=0,
    start_00=False,
    time_gap_hours=2,
)
env_config = dict(
```

```
    dataset=MyDataset,
    engine=MyCerebro,
    render_modes=['episode', 'human', 'model_input'],
    render_state_as_image=True,
    render_ylabel='OHLC Price Gradients',
    render_size_episode=(12,8),
    render_size_human=(8, 3.5),
    render_size_state=(10, 5),
    render_dpi=75,
    port=5100,
    data_port=5099,
    connect_timeout=60,
    verbose=0,
)

# Set tensorflow distributed cluster and a3c configuration:
cluster_config = dict(
    host='127.0.0.1',
    port=42222,
    num_workers=8,
    num_ps=1,
)
launcher = Launcher(
    cluster_config=cluster_config,
    env_class=BTgymEnv,
    env_config=env_config,
    policy_class=LSTMPolicy,
    rollout_length=20,
    test_mode=False,
    train_steps=1000000000,
    model_summary_freq=20,
    episode_summary_freq=1,
    env_render_freq=10,
    verbose=1

)
```

Historical Data Test

1. **Import data use library called DUKA**

pip install duka

usage

```
usage: duka [options]

 positional arguments:
   SYMBOLS              symbol list using format EURUSD EURGBP

 optional arguments:
   -h          show help message and exit
   -v          show program's version number and exit
   -d DAY      specific day format YYYY-MM-DD (default today)
   -s STARTDATE start date format YYYY-MM-DD (default today)
   -e ENDDATE   end date format YYYY-MM-DD (default today)
   -c CANDLE    use candles instead of ticks. Accepted values M1 M2 M5 M10 M15
M30 H1 H4 D1
   -f FOLDER    the dowloaded data will be saved in FOLDER (default '.')
   -t THREAD    number of threads (default 10)
   --header     include CSV header (default false)

duka EURUSD -s 2001-01-01 -e 2015-12-31
```

2. **Build Neural Network for AI agent**

```
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop

model = Sequential()
model.add(Dense(48, init='lecun_uniform', input_shape=(7,)))
model.add(Activation('relu'))

model.add(Dense(48, init='lecun_uniform'))
model.add(Activation('relu'))

model.add(Dense(4, init='lecun_uniform'))
model.add(Activation('linear'))
rms = RMSprop()
model.compile(loss='mse', optimizer=rms)
```

3. **Example of training network and backtest**

```
import numpy as np

from market_env import MarketEnv
```

```
from market_model_builder import MarketModelBuilder

class ExperienceReplay(object):
    def __init__(self, max_memory=100, discount=.9):
        self.max_memory = max_memory
        self.memory = list()
        self.discount = discount

    def remember(self, states, game_over):
        self.memory.append([states, game_over])
        if len(self.memory) > self.max_memory:
            del self.memory[0]

    def get_batch(self, model, batch_size=10):
        len_memory = len(self.memory)
        num_actions = model.output_shape[-1]
        inputs = []

        dim = len(self.memory[0][0][0])
        for i in xrange(Vladimir & Kabysh):
            inputs.append([])

        targets = np.zeros((min(len_memory, batch_size), num_actions))
        for i, idx in enumerate(np.random.randint(0, len_memory,
size=min(len_memory, batch_size))):
            state_t, action_t, reward_t, state_tp1 =
self.memory[idx][0]
            game_over = self.memory[idx][1]

            for j in xrange(Vladimir & Kabysh):
                inputs[j].append(state_t[j][0])

            targets[i] = model.predict(state_t)[0]
            Q_sa = np.max(model.predict(state_tp1)[0])
            if game_over: # if game_over is True
                targets[i, action_t] = reward_t
            else:
                targets[i, action_t] = reward_t + self.discount *
Q_sa

        inputs = [np.array(inputs[i]) for i in xrange(Vladimir & Kabysh)]
```

```
                return inputs, targets

if __name__ == "__main__":
        import sys
        import codecs

        codeListFilename = sys.argv[1]
        modelFilename = sys.argv[2] if len(sys.argv) > 2 else None

        codeMap = {}
        f = codecs.open(codeListFilename, "r", "utf-8")

        for line in f:
                if line.strip() != "":
                        tokens = line.strip().split(",") if not "\t" in line else
line.strip().split("\t")
                        codeMap[tokens[0]] = tokens[1]

        f.close()

        env = MarketEnv(dir_path = "./data/", target_codes = codeMap.keys(),
input_codes = [], start_date = "2001-01-01", end_date = "2015-12-31", sudden_death = -1.0)

        # parameters
        epsilon = .5  # exploration
        min_epsilon = 0.1
        epoch = 100000
        max_memory = 5000
        batch_size = 128
        discount = 0.8

        from keras.optimizers import SGD
        model = MarketModelBuilder(modelFilename).getModel()
        sgd = SGD(lr = 0.001, decay = 1e-6, momentum = 0.9, nesterov = True)
        model.compile(loss='mse', optimizer='rmsprop')

        # Initialize experience replay object
        exp_replay = ExperienceReplay(max_memory = max_memory, discount =
discount)

        # Train
        win_cnt = 0
        for e in range(epoch):
```

```
loss = 0.
env.reset()
game_over = False
# get initial input
input_t = env.reset()
cumReward = 0

while not game_over:
    input_tm1 = input_t
    isRandom = False

    # get next action
    if np.random.rand() <= epsilon:
        action = np.random.randint(0, env.action_space.n,
size=1)[0]

        isRandom = True
    else:
        q = model.predict(input_tm1)
        action = np.argmax(q[0])

        #print "  ".join(["%s:%.2f" % (l, i) for l, i in zip(env.actions,
q[0].tolist())])
        if np.nan in q:
            print "OCCUR NaN!!!"
            exit()

    # apply action, get rewards and new state
    input_t, reward, game_over, info = env.step(action)
    cumReward += reward

    if env.actions[action] == "LONG" or env.actions[action] ==
"SHORT":
        color = bcolors.FAIL if env.actions[action] == "LONG"
else bcolors.OKBLUE
        if isRandom:
            color = bcolors.WARNING if env.actions[action]
== "LONG" else bcolors.OKGREEN
        print "%s:\t%s\t%.2f\t%.2f\t" % (info["dt"], color +
env.actions[action] + bcolors.ENDC, cumReward, info["cum"]) + ("\t".join(["%s:%.2f" % (l, i)
for l, i in zip(env.actions, q[0].tolist())]) if isRandom == False else "")
```

```
                exp_replay.remember([input_tm1, action, reward, input_t],
game_over)

                inputs, targets = exp_replay.get_batch(model,
batch_size=batch_size)

                loss += model.train_on_batch(inputs, targets)

            if cumReward > 0 and game_over:
                win_cnt += 1

            print("Epoch {:03d}/{} | Loss {:.4f} | Win count {} | Epsilon
{:.4f}".format(e, epoch, loss, win_cnt, epsilon))

            model.save_weights("model.h5" if modelFilename == None else
modelFilename, overwrite=True)
            epsilon = max(min_epsilon, epsilon * 0.89)
```

4. **Example of analyzing stock returns (**from https://github.com/backtrader/backtrader/blob/master/samples/analyzer-annualreturn/analyzer-annualreturn.py)

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)

import argparse
import datetime

# The above could be sent to an independent module
import backtrader as bt
import backtrader.feeds as btfeeds
import backtrader.indicators as btind
from backtrader.analyzers import (SQN, AnnualReturn, TimeReturn,
SharpeRatio,
                                  TradeAnalyzer)


class LongShortStrategy(bt.Strategy):
    '''This strategy buys/sells upong the close price crossing
    upwards/downwards a Simple Moving Average.

    It can be a long-only strategy by setting the param "onlylong" to True
    '''
    params = dict(
        period=15,
```

```
    stake=1,
    printout=False,
    onlylong=False,
    csvcross=False,
)

def start(self):
    pass

def stop(self):
    pass

def log(self, txt, dt=None):
    if self.p.printout:
        dt = dt or self.data.datetime[0]
        dt = bt.num2date(dt)
        print('%s, %s' % (dt.isoformat(), txt))

def __init__(self):
    # To control operation entries
    self.orderid = None

    # Create SMA on 2nd data
    sma = btind.MovAv.SMA(self.data, period=self.p.period)
    # Create a CrossOver Signal from close an moving average
    self.signal = btind.CrossOver(self.data.close, sma)
    self.signal.csv = self.p.csvcross

def next(self):
    if self.orderid:
        return  # if an order is active, no new orders are allowed

    if self.signal > 0.0:  # cross upwards
        if self.position:
            self.log('CLOSE SHORT , %.2f' % self.data.close[0])
            self.close()

        self.log('BUY CREATE , %.2f' % self.data.close[0])
        self.buy(size=self.p.stake)

    elif self.signal < 0.0:
        if self.position:
            self.log('CLOSE LONG , %.2f' % self.data.close[0])
            self.close()

        if not self.p.onlylong:
            self.log('SELL CREATE , %.2f' % self.data.close[0])
```

```
            self.sell(size=self.p.stake)

    def notify_order(self, order):
        if order.status in [bt.Order.Submitted, bt.Order.Accepted]:
            return  # Await further notifications

        if order.status == order.Completed:
            if order.isbuy():
                buytxt = 'BUY COMPLETE, %.2f' % order.executed.price
                self.log(buytxt, order.executed.dt)
            else:
                selltxt = 'SELL COMPLETE, %.2f' % order.executed.price
                self.log(selltxt, order.executed.dt)

        elif order.status in [order.Expired, order.Canceled, order.Margin]:
            self.log('%s ,' % order.Status[order.status])
            pass  # Simply log

        # Allow new orders
        self.orderid = None

    def notify_trade(self, trade):
        if trade.isclosed:
            self.log('TRADE PROFIT, GROSS %.2f, NET %.2f' %
                     (trade.pnl, trade.pnlcomm))

        elif trade.justopened:
            self.log('TRADE OPENED, SIZE %2d' % trade.size)


def runstrategy():
    args = parse_args()

    # Create a cerebro
    cerebro = bt.Cerebro()

    # Get the dates from the args
    fromdate = datetime.datetime.strptime(args.fromdate, '%Y-%m-%d')
    todate = datetime.datetime.strptime(args.todate, '%Y-%m-%d')

    # Create the 1st data
    data = btfeeds.BacktraderCSVData(
        dataname=args.data,
        fromdate=fromdate,
        todate=todate)

    # Add the 1st data to cerebro
```

```
    cerebro.adddata(data)

    # Add the strategy
    cerebro.addstrategy(LongShortStrategy,
                        period=args.period,
                        onlylong=args.onlylong,
                        csvcross=args.csvcross,
                        stake=args.stake)

    # Add the commission - only stocks like a for each operation
    cerebro.broker.setcash(args.cash)

    # Add the commission - only stocks like a for each operation
    cerebro.broker.setcommission(commission=args.comm,
                                 mult=args.mult,
                                 margin=args.margin)

    tframes = dict(
        days=bt.TimeFrame.Days,
        weeks=bt.TimeFrame.Weeks,
        months=bt.TimeFrame.Months,
        years=bt.TimeFrame.Years)

    # Add the Analyzers
    cerebro.addanalyzer(SQN)
    if args.legacyannual:
        cerebro.addanalyzer(AnnualReturn)
        cerebro.addanalyzer(SharpeRatio, legacyannual=True)
    else:
        cerebro.addanalyzer(TimeReturn, timeframe=tframes[args.tframe])
        cerebro.addanalyzer(SharpeRatio, timeframe=tframes[args.tframe])

    cerebro.addanalyzer(TradeAnalyzer)

    cerebro.addwriter(bt.WriterFile, csv=args.writercsv, rounding=4)

    # And run it
    cerebro.run()

    # Plot if requested
    if args.plot:
        cerebro.plot(numfigs=args.numfigs, volume=False, zdown=False)


def parse_args():
    parser = argparse.ArgumentParser(description='TimeReturn')
```

```
parser.add_argument('--data', '-d',
                    default='../../datas/2005-2006-day-001.txt',
                    help='data to add to the system')

parser.add_argument('--fromdate', '-f',
                    default='2005-01-01',
                    help='Starting date in YYYY-MM-DD format')

parser.add_argument('--todate', '-t',
                    default='2006-12-31',
                    help='Starting date in YYYY-MM-DD format')

parser.add_argument('--period', default=15, type=int,
                    help='Period to apply to the Simple Moving Average')

parser.add_argument('--onlylong', '-ol', action='store_true',
                    help='Do only long operations')

parser.add_argument('--writercsv', '-wcsv', action='store_true',
                    help='Tell the writer to produce a csv stream')

parser.add_argument('--csvcross', action='store_true',
                    help='Output the CrossOver signals to CSV')

group = parser.add_mutually_exclusive_group()
group.add_argument('--tframe', default='years', required=False,
                   choices=['days', 'weeks', 'months', 'years'],
                   help='TimeFrame for the returns/Sharpe calculations')

group.add_argument('--legacyannual', action='store_true',
                   help='Use legacy annual return analyzer')

parser.add_argument('--cash', default=100000, type=int,
                    help='Starting Cash')

parser.add_argument('--comm', default=2, type=float,
                    help='Commission for operation')

parser.add_argument('--mult', default=10, type=int,
                    help='Multiplier for futures')

parser.add_argument('--margin', default=2000.0, type=float,
                    help='Margin for each future')

parser.add_argument('--stake', default=1, type=int,
                    help='Stake to apply in each operation')
```

```
        parser.add_argument('--plot', '-p', action='store_true',
                            help='Plot the read data')

        parser.add_argument('--numfigs', '-n', default=1,
                            help='Plot using numfigs figures')

        return parser.parse_args()


    if __name__ == '__main__':
        runstrategy()
```

5. **Sample of Building RL Brain (**from https://github.com/noootown/Forex-DQN/blob/master/train/RLBrain.py)

```
import numpy as np
import tensorflow as tf
import datetime as dt

from account import Account
from constants import STOP, BUY, SELL, CLOSE, SHOW_HAND
from helper import mkdir

np.random.seed(dt.datetime.now().microsecond)
tf.set_random_seed(dt.datetime.now().microsecond)

class DeepNetwork:
 def __init__(
   self,
   forex,
   dates,
   featureNum,
   config,
 ):
   self.forex = forex
   self.dates = dates
   self.config = config

   self.episodes = config['episodes']
   self.interval = config['interval']

   self.n_actions = 4
   self.n_features = featureNum * config['count']
   self.lr = config['learning_rate']
   self.gamma = config['reward_decay']
   self.epsilon_max = config['e_greedy'] if config['isTrain'] else 1
```

```
    self.replace_target_iter = config['replace_target_iter']
    self.memory_size = config['memory_size']
    self.batch_size = config['batch_size']
    self.epsilon_increment = config['e_greedy_increment']
    self.epsilon = 0 if config['e_greedy_increment'] is not None else
self.epsilon_max

    # account
    self.initBalance = 100000

    self.isTrain = config['isTrain']
    self.dir = config['dir']

    self.ckptFile = config['ckptFile']
    self.ckptSavePeriod = config['ckptSavePeriod']

    # total learning step
    self.step = -config['startStep']
    self.totalLoss = 0
    self.totalMaxQ = 0
    self.r_actions = []

    # initialize zero memory [s, a, r, s_]
    self.memory = np.zeros((self.memory_size, self.n_features * 2 + 2))
    self.memory_counter = 0

    self.sess = tf.Session()

    # consist of [target_net, evaluate_net]
    self.buildNet()

    self.saver = tf.train.Saver(max_to_keep = int(self.episodes /
self.ckptSavePeriod))

    if config['isLoad'] or not self.isTrain:
      self.saver.restore(self.sess, 'data/%s/%s' % (self.dir, self.ckptFile))
      print('Load data/%s/%s sucessfully!\n' % (self.dir, self.ckptFile))
    else:
      self.sess.run(tf.global_variables_initializer())
      print('Apply global initializer!\n')

  def subTrain(self, isTrain, dates):
    account = Account(
      balance = self.initBalance,
      cliOutput = self.config['cliOutput'],
    )
    for date in dates:
```

```
    self.forex.setDate(date)

    startTime, endTime = self.forex.getTime()
    price, state = self.forex.getPrice(startTime)

    for time in range(startTime, endTime - self.interval, self.interval):
      # Q learning start
      action = self.chooseAction(state, isTrain)

      reward = 0
      if action == STOP:
        reward = account.stop()
      elif action == BUY or action == SELL:
        reward = account.order(
          price, # price is at column 0
          {
            'type': action,
            'unit': SHOW_HAND,
          },
          time = time
        )
      elif action == CLOSE:
        reward = account.closePosition(price, time = time)

      price, state_ = self.forex.getPrice(time + self.interval)

      self.storeTransition(
        transition = np.hstack((state, [action, reward], state_)),
        mode = 0 if isTrain else 1,
      )

      if isTrain:
        if self.step > 0 and self.step % self.config['learn_period'] == 0:
          self.learn()
        self.step += 1

      state = state_

  return account.balance

def train(self):
  if self.isTrain:
    print('Start training\n')
  else:
    print('Start testing\n')

  for episode in range(self.episodes):
```

```
      if episode % 10 == 0:
        print('episode', episode)

      if self.isTrain:
        epsilonBalance = self.subTrain(isTrain = True, dates = self.dates)

        # to get the actual balance
        realBalance = self.subTrain(isTrain = False, dates = self.dates)

        self.finishEpisode(episode, epsilonBalance, realBalance)
      else:
        if self.config['cliOutput']:
          print(Account.getCloseHeader())

        print(self.subTrain(isTrain = False, dates = self.dates))

    if self.isTrain:
      print('Finish training\n')
    else:
      print('Finish testing\n')

  def optimize(self):
    if self.config['optimizer'] == 'RMSProp':
      return tf.train.RMSPropOptimizer(
        self.lr,
        decay = 0.9 if self.config['op_decay'] == None else self.config['op_decay'],
        momentum = 0.0 if self.config['op_momentum'] == None else
self.config['op_momentum'],
        epsilon = pow(10, -10) if self.config['op_epsilon'] == None else
self.config['op_epsilon'],
      ).minimize(self.loss)

  def buildNet(self):
    def addLayer(
          name,
          input,
          output_dim,
          w_init,
          b_init,
          c_names,
          active_fn = None,
    ):
      with tf.variable_scope(name):
        w = tf.get_variable('w', [input.get_shape().as_list()[1], output_dim],
                    initializer = w_init, collections = c_names)
        b = tf.get_variable('b', [1, output_dim], initializer = b_init, collections =
c_names)
```

```
        if active_fn != None:
          out = active_fn(tf.matmul(input, w) + b)
        else:
          out = tf.matmul(input, w) + b

      return out, w, b

    # ------------------ build evaluate_net ------------------
    self.s = tf.placeholder(tf.float32, [None, self.n_features], name='s')  # input
    self.q_target = tf.placeholder(tf.float32, [None, self.n_actions],
name='Q_target')  # for calculating loss

    # config of layers
    self.w = {}

    self.w_init = \
      tf.random_normal_initializer(self.config['init_w_mean'],
self.config['init_w_std'])
    self.b_init = \
      tf.constant_initializer(self.config['init_b'])

    active_fn = tf.nn.relu

    with tf.variable_scope('eval_net'):
      # c_names(collections_names) are the collections to store variables
      c_names = ['eval_net_params', tf.GraphKeys.GLOBAL_VARIABLES]

      self.w['l1_o'], self.w['l1_w'], self.w['l1_b'] =\
        addLayer('l1', self.s, self.config['l1_dim'], self.w_init, self.b_init, c_names,
active_fn)

      # output layer
      self.q_eval, self.w['lout_w'], self.w['lout_b'] = \
        addLayer('lout', self.w['l1_o'], self.n_actions, self.w_init, self.b_init,
c_names)

    with tf.name_scope('loss'):
      self.loss = tf.reduce_sum(tf.squared_difference(self.q_target, self.q_eval))

    with tf.name_scope('train'):
      self._train_op = self.optimize()

    # ------------------ build target_net ------------------
    self.s_ = tf.placeholder(tf.float32, [None, self.n_features], name='s_')    # input
    self.t_w = {}
```

```
    with tf.variable_scope('target_net'):
      c_names = ['target_net_params', tf.GraphKeys.GLOBAL_VARIABLES]

      self.t_w['l1_o'], self.t_w['l1_w'], self.t_w['l1_b'] = \
        addLayer('l1', self.s_, self.config['l1_dim'], self.w_init, self.b_init, c_names,
active_fn)

      # output layer
      self.q_next, self.t_w['lout_w'], self.t_w['lout_b'] = \
        addLayer('lout', self.t_w['l1_o'], self.n_actions, self.w_init, self.b_init,
c_names)

    with tf.variable_scope('summary'):
      # e_XX means with epsilon
      # r_XX means without epsilon, which is real simulation
      scalar_summary_tags = ['loss_avg', 'e_balance', 'r_balance',
                             'q_max', 'q_total', 'epsilon']

      self.summary_placeholders = {}
      self.summary_ops = {}

      for tag in scalar_summary_tags:
        self.summary_placeholders[tag] = tf.placeholder('float32', None,
name=tag.replace(' ', '_') + '_0')
        self.summary_ops[tag] = tf.summary.scalar(tag,
self.summary_placeholders[tag])

      histogram_summary_tags = ['r_actions']

      for tag in histogram_summary_tags:
        self.summary_placeholders[tag] = tf.placeholder('float32', None,
name=tag.replace(' ', '_') + '_0')
        self.summary_ops[tag] = tf.summary.histogram(tag,
self.summary_placeholders[tag])

    with tf.variable_scope('param'):
      w_c_names = 'eval_net_params_summaries'
      histogram_w_tags = ['l1_w', 'l1_b', 'lout_w', 'lout_b']
      for tag in histogram_w_tags:
        tf.summary.histogram(tag, self.w[tag], collections = [w_c_names])

    if self.isTrain:
      self.merged = tf.summary.merge_all(key = w_c_names)
      self.writer = tf.summary.FileWriter('data/' + self.dir, self.sess.graph)

  # state action reward next state
  # mode: 0: store train 1: store test
```

```
def storeTransition(self, transition, mode = 0):
  # replace the old memory with new memory
  self.memory[self.memory_counter % self.memory_size] = transition
  self.memory_counter += 1

def chooseAction(self, observation, isTrain = True):
  # to have batch dimension when feed into tf placeholder
  observation = observation[np.newaxis, :]

  if not isTrain:
    actions = self.sess.run(self.q_eval, feed_dict={self.s: observation})
    action = np.argmax(actions)
    self.r_actions.append(action)

  elif self.step < 0 or np.random.uniform() >= self.epsilon:
    action = np.random.randint(0, self.n_actions)

  else:
    # forward feed the observation and get q value for every actions
    actions = self.sess.run(self.q_eval, feed_dict={self.s: observation})
    action = np.argmax(actions)

  return action

def replaceTargetParams(self):
  t_params = tf.get_collection('target_net_params')
  e_params = tf.get_collection('eval_net_params')
  self.sess.run([tf.assign(t, e) for t, e in zip(t_params, e_params)])

def learn(self):
  # check to replace target parameters
  if self.step % self.replace_target_iter == 0:
    self.replaceTargetParams()

  # sample batch memory from all memory
  batch_memory =\
  self.memory[np.random.choice(
    self.memory_size\
    if self.memory_counter > self.memory_size\
    else self.memory_counter,
    self.batch_size), :]

  q_next, q_eval = self.sess.run(
    [self.q_next, self.q_eval],
    feed_dict={
      self.s_: batch_memory[:, -self.n_features:],
      self.s: batch_memory[:, :self.n_features]
```

```
      })

    # change q_target w.r.t q_eval's action
    q_target = q_eval.copy()

    q_target[np.arange(self.batch_size), batch_memory[:,
self.n_features].astype(LeCun, Bengio, & Hinton)] = \
      batch_memory[:, self.n_features + 1] + self.gamma * np.max(q_next, axis=1)

    # train eval network
    _, self.param_summary, cost = \
      self.sess.run([self._train_op, self.merged, self.loss],
                feed_dict={self.s: batch_memory[:, :self.n_features],
                      self.q_target: q_target,
                      })

    # increasing epsilon
    self.epsilon =\
      self.epsilon + self.epsilon_increment\
      if self.epsilon < self.epsilon_max\
      else self.epsilon_max
    self.totalLoss += cost
    self.totalQ += q_eval.mean(axis = 1).mean(axis = 0)
    self.totalMaxQ += np.max(q_eval, axis=1).mean()

  # mode 0: normal save, 1: period save
  def saveParam(self, dir = 'tmp', mode = 0):
    subdir = ''

    if mode == 1:
      subdir = 'history/%s/' % (dir)

    fulldir = 'data/%s/%s' % (self.dir, subdir)

    mkdir(fulldir)
    self.saver.save(self.sess, '%s%s' % (fulldir, self.ckptFile))

  def injectSummary(self, tag_dict, episode):
    summary_str_lists = self.sess.run([self.summary_ops[tag] for tag in
tag_dict.keys()], {
      self.summary_placeholders[tag]: value for tag, value in tag_dict.items()
    })
    for summary_str in summary_str_lists:
      self.writer.add_summary(summary_str, episode)

    self.writer.add_summary(self.param_summary, episode)
```

```
def finishEpisode(self, episode, epsilonBalance, realBalance):
  if self.step > 0:
    injectDict = {
      # scalar
      'loss_avg': self.totalLoss,
      'e_balance': epsilonBalance,
      'r_balance': realBalance,
      'epsilon': self.epsilon,
      'q_max': self.totalMaxQ,
      'q_total': self.totalQ,
      # histogram
      'r_actions': self.r_actions,
    }
    self.injectSummary(injectDict, episode)

    self.saveParam(mode = 0)
    if episode % self.ckptSavePeriod == 0:
      self.saveParam(dir = '%d' % (episode), mode = 1)

  self.r_actions = []
  self.totalLoss = 0
  self.totalQ = 0
  self.totalMaxQ = 0
```

6. **Sample of DQN trade**
   (From https://github.com/jjakimoto/DQN/blob/master/model/dqn.py)

```
import tensorflow as tf
from keras.layers.convolutional import Convolution2D
from keras.layers.pooling import MaxPooling2D
from keras.layers.core import Flatten, Lambda
from keras.models import Sequential
from keras.layers import BatchNormalization
from keras.layers import Activation
from keras.layers.core import Dense
from keras.engine.topology import Merge
from keras.layers.advanced_activations import PReLU
from keras.layers import SpatialDropout2D
from keras.layers import Dropout, Reshape
from keras import backend as K
import numpy as np
import pandas as pd
import time
# local library
from memory import SequentialMemory

class DQN(object):
```

```
    """Deep Q-Learning Networ

    Basend on DQN and Multiscale CNN, find the optimal time to
    exit from a stock market.

    Available function
    - build_model: build network based on tensorflow and keras
    - train: given DateFrame stock data, train network
    - predict_action: givne DataFrame stock data, return optimal protfolio
    """

    def __init__(self, config):
        """initialized approximate value function

        config should have the following attributes

        Args:
            device: the device to use computation, e.g. '/gpu:0'
            gamma(float): the decay rate for value at RL
            history_length(LeCun et al.): input_length for each scale at CNN
            n_feature(LeCun et al.): the number of type of input
                (e.g. the number of company to use at stock trading)
            n_history(LeCun et al.): the nubmer of history that will be used as
input
            n_smooth, n_down(LeCun et al.): the number of smoothed and down
sampling input at CNN
            k_w(LeCun et al.): the size of filter at CNN
            n_hidden(LeCun et al.): the size of fully connected layer
            n_batch(LeCun et al.): the size of mini batch
            n_epochs(LeCun et al.): the training epoch for each time
            update_rate (0, 1): parameter for soft update
            learning_rate(float): learning rate for SGD
            memory_length(LeCun et al.): the length of Replay Memory
            n_memory(LeCun et al.): the number of different Replay Memories
            alpha, beta: [0, 1] parameters for Prioritized Replay Memories
        """
        self.device = config.device
        self.save_path = config.save_path
        self.is_load = config.is_load
        self.gamma = config.gamma
        self.history_length = config.history_length
        self.n_stock = config.n_stock
        self.n_feature = config.n_feature
        self.n_smooth = config.n_smooth
        self.n_down = config.n_down
        self.k_w = config.k_w
        self.n_hidden = config.n_hidden
```

```
        self.n_batch = config.n_batch
        self.n_epochs = config.n_epochs
        self.update_rate = config.update_rate
        self.alpha = config.alpha
        self.beta = config.beta
        self.lr = config.learning_rate
        self.memory_length = config.memory_length
        self.n_memory = config.n_memory
        # the length of the data as input
        self.n_history = max(self.n_smooth + self.history_length, (self.n_down +
1) * self.history_length)
        print ("building model....")
        # have compatibility with new tensorflow
        tf.python.control_flow_ops = tf
        # avoid creating _LEARNING_PHASE outside the network
        K.clear_session()
        self.sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True,
log_device_placement=False))
        K.set_session(self.sess)
        with self.sess.as_default():
            with tf.device(self.device):
                self.build_model()
        print('finished building model!')

    def train(self, input_data, noise_scale=0.1):
        """training DQN, which has two actions: 0-exit, 1-stay

        Args:
            data (DataFrame): stock price for self.n_feature companies
        """
        stock_data = input_data.values
        date = input_data.index
        T = len(stock_data)
        self.noise_scale = noise_scale

        # frequency for output
        print_freq = int(T / 100)
        if print_freq == 0:
            print_freq = 1
        print ("training....")
        st = time.time()
        #  udpate rate for prioritizing parameter
        db = (1 - self.beta) / 1000

        # result for return value
        values = [[] for _ in range(self.n_stock)]
        date_label = [[] for _ in range(self.n_stock)]
```

```
        date_use = []
        stock_use = []
        # will not train until getting enough data
        t0 = self.n_history + self.n_batch
        self.initialize_memory(stock_data[:t0], scale=noise_scale)
        save_data_freq = 10
        save_weight_freq = 10
        count = 0
        input_data.to_csv("stock_price.csv")
        for t in range(t0, T):
            stock_use.append(stock_data[t])
            date_use.append(date[t])
            action = self.predict_action(stock_data[t])
            for i in range(self.n_stock):
                if action[i] == 0:
                    date_label[i].append(date[t])
                    values[i].append(stock_data[t][i])
            self.update_memory(stock_data[t])
            count += 1
            for epoch in range(self.n_epochs):
                # select transition from pool
                self.update_weight()
                # update prioritizing paramter untill it goes over 1
            self.beta  += db
            if self.beta >= 1.0:
                self.beta = 1.0
            idx = np.random.randint(0, self.n_memory)

            experiences, weights = self.memory[idx].sample(self.n_batch,
self.n_history, self.alpha, self.beta)
            max_idx = self.get_max_idx(experiences.state1)
            target_value = self.sess.run(self.target_value,
                                feed_dict={self.state_target: experiences.state1,
                                self.reward: experiences.reward,
                                        self.max_idx_target: max_idx})

            if t % print_freq == 0:
                print ("time:",  date[t])
                error = self.sess.run(self.error,
                            feed_dict={self.state: experiences.state0,
                                    self.target: target_value,
                                    self.reward: experiences.reward,
                                    K.learning_phase(): 0})
                print("error:", np.mean(error))
                action = self.predict_action(stock_data[t])
                print("portfolio:", action)
                print ("elapsed time", time.time() - st)
```

```
print("**********************************************************
**********")

            if count % save_data_freq == 0:
                for i in range(self.n_stock):
                    result = pd.DataFrame(values[i],
index=pd.DatetimeIndex(date_label[i]))
                    result.to_csv("exit_result_{}.csv".format(i))
                data_use = pd.DataFrame(stock_use,
index=pd.DatetimeIndex(date_use))
                data_use.to_csv("stock_price.csv")

            if count % save_weight_freq == 0:
                save_path = self.saver.save(self.sess, self.save_path)
                print("Model saved in file: %s" % self.save_path)

        save_path = self.saver.save(self.sess, self.save_path)
        print("Model saved in file: %s" % self.save_path)
        print ("finished training")

        return [pd.DataFrame(values[i], index=pd.DatetimeIndex(date_label[i]))
for i in range(self.n_stock)]

    def predict_action(self, state):
        """Preduct Optimal strategy

        Args:
            state(float): stock data with size: [self.n_stock, ]
        Retrun:
            integer: 0-exit, 1-stay
        """
        pred_state = self.memory[0].sample_state_uniform(self.n_batch,
self.n_history)
        new_state = pred_state[-1]
        new_state = np.concatenate((new_state[1:], [state]), axis=0)
        pred_state = np.concatenate((pred_state[:-1], [new_state]), axis=0)
        action = self.max_action.eval(
            session=self.sess,
            feed_dict={self.state: pred_state, K.learning_phase(): 0})[-1]
        return action

    def update_weight(self):
        """Update networks' parameters and memories"""
        idx = np.random.randint(0, self.n_memory)
        experiences, weights = self.memory[idx].sample(self.n_batch,
self.n_history, self.alpha, self.beta)
```

```
        max_idx = self.get_max_idx(experiences.state1)
        # get target value for optimization
        target_value = self.sess.run(self.target_value,
                                     feed_dict={self.state_target: experiences.state1,
                                self.reward: experiences.reward,
                                             self.max_idx_target: max_idx})
        # optimize network
        self.sess.run(self.critic_optim,
                      feed_dict={self.state: experiences.state0,
                                 self.target: target_value,
                                 self.weights: weights,
                                 self.learning_rate: self.lr,
                                 K.learning_phase(): 1})
        # compute errors to determine prioritizing ratio
        error = self.sess.run(self.error,
                              feed_dict={self.state: experiences.state0,
                                         self.target: target_value,
                                         self.reward: experiences.reward,
                                         K.learning_phase(): 0})
        self.memory[idx].update_priority(error)
        # softupdate for critic network
        old_weights = self.critic_target.get_weights()
        new_weights = self.critic.get_weights()
        weights = [self.update_rate * new_w + (1 - self.update_rate) * old_w
                   for new_w, old_w in zip(new_weights, old_weights)]
        self.critic_target.set_weights(weights)

    def initialize_memory(self, stocks, scale=10):
        self.memory = []
        for i in range(self.n_memory):
            self.memory.append(SequentialMemory(self.memory_length))
        for t in range(len(stocks)):
            for idx_memory in range(self.n_memory):
                action = None
                reward = np.concatenate((np.reshape(stocks[t], (self.n_stock, 1)),
np.zeros((self.n_stock, 1))), axis=-1)
                self.memory[idx_memory].append(stocks[t], action, reward)

    def update_memory(self, state):
        """Update memory without updating weight"""
        for i in range(self.n_memory):
            self.memory[i].observations.append(state)
            self.memory[i].priority.append(1.0)
        # to stabilize batch normalization, use other samples for prediction
        pred_state = self.memory[0].sample_state_uniform(self.n_batch,
self.n_history)
        for i in range(self.n_memory):
```

```
            action_off = None
            reward_off = np.concatenate((np.reshape(state, (self.n_stock, 1)),
np.zeros((self.n_stock, 1))), axis=-1)
            self.memory[i].rewards.append(reward_off)
            self.memory[i].actions.append(action_off)

    def get_max_idx(self, state):
        max_action = self.sess.run(self.max_action_target,
feed_dict={self.state_target: state})
        shape = max_action.shape
        max_idx = []
        for i in range(shape[0]):
            for j in range(shape[1]):
                max_idx.append([i, j, max_action[i][j]])
        return np.array(max_idx, dtype=int)


    def build_model(self):
        """Build all of the network and optimizations

        just for conveninece of trainig, seprate placehoder for train and target
network
        critic network input: [raw_data, smoothed, downsampled]
        """
        self.critic = self.build_critic()
        self.critic_target = self.build_critic()
        # transform input into the several scales and smoothing
        self.state =  tf.placeholder(tf.float32, [None, self.n_history, self.n_stock],
name='state')
        self.state_target = tf.placeholder(tf.float32, [None, self.n_history,
self.n_stock], name='state_target')
        # reshape to convolutional input
        state_ = tf.reshape(self.state, [-1, self.n_history, self.n_stock, 1])
        state_target_ = tf.reshape(self.state_target, [-1, self.n_history,
self.n_stock, 1])
        raw, smoothed, down = self.transform_input(state_)
        raw_target, smoothed_target, down_target =
self.transform_input(state_target_)

        # build graph for citic training
        input_q = [raw,] +  smoothed + down
        self.Q = self.critic(input_q)
        self.max_action = tf.argmax(self.Q, dimension=2)
        # target network
        input_q_target = [raw_target,] +  smoothed_target + down_target
        Q_target = self.critic_target(input_q_target)
```

```
        self.reward = tf.placeholder(tf.float32, [None, self.n_stock, 2],
name='reward')
        double_Q = self.critic(input_q_target)
        self.max_action_target = tf.argmax(double_Q, 2)
        self.max_idx_target = tf.placeholder(tf.int32, [None, 3], "double_idx")
        Q_max = tf.gather_nd(Q_target, self.max_idx_target)
        Q_max = tf.reshape(Q_max, [-1, self.n_stock, 1])
        Q_value = tf.concat(2, (tf.zeros_like(Q_max), Q_max))
        self.target_value = self.reward  + self.gamma * Q_value
        self.target_value = tf.cast(self.target_value, tf.float32)
        self.target = tf.placeholder(tf.float32, [None, self.n_stock, 2],
name="target_value")
        # optimization
        self.learning_rate = tf.placeholder(tf.float32, shape=[],
name="learning_rate")
        # get rid of bias of prioritized
        self.weights = tf.placeholder(tf.float32, shape=[None], name="weights")
        self.loss = tf.reduce_mean(self.weights *
tf.reduce_sum(tf.square(self.target - self.Q), [1, 2]), name='loss')
        # TD-error for priority
        self.error = tf.reduce_sum(tf.abs(self.target - self.Q), [1, 2])
        self.critic_optim = tf.train.AdamOptimizer(self.learning_rate) \
          .minimize(self.loss, var_list=self.critic.trainable_weights)

        self.saver = tf.train.Saver()
        is_initialize = True
        if self.is_load:
          if self.load(self.save_path):
            print('succeded to load')
            is_initialize = False
          else:
            print('failed to load')

        # initialize network
        tf.initialize_all_variables().run(session=self.sess)
        weights = self.critic.get_weights()
        self.critic_target.set_weights(weights)

    def build_critic(self):
        """Build critic network

        recieve transformed tensor: raw_data, smooted_data, and
downsampled_data
        """
        nf = self.n_feature
        # layer1
        # smoothed input
```

```
        sm_model = [Sequential() for _ in range(self.n_smooth)]
        for m in sm_model:
            m.add(Lambda(lambda x: x,  input_shape=(self.history_length,
self.n_stock, 1)))
            m.add(Convolution2D(nb_filter=nf, nb_row=self.k_w, nb_col=1,
border_mode='same'))
            m.add(BatchNormalization(mode=2, axis=-1))
            m.add(PReLU())
        # down sampled input
        dw_model = [Sequential() for _ in range(self.n_down)]
        for m in dw_model:
            m.add(Lambda(lambda x: x,  input_shape=(self.history_length,
self.n_stock, 1)))
            m.add(Convolution2D(nb_filter=nf, nb_row=self.k_w, nb_col=1,
border_mode='same'))
            m.add(BatchNormalization(mode=2, axis=-1))
            m.add(PReLU())
        # raw input
        state = Sequential()
        nf = self.n_feature
        state.add(Lambda(lambda x: x,  input_shape=(self.history_length,
self.n_stock, 1)))
        state.add(Convolution2D(nb_filter=nf, nb_row=self.k_w, nb_col=1,
border_mode='same'))
        state.add(BatchNormalization(mode=2, axis=-1))
        state.add(PReLU())
        merged = Merge([state,] + sm_model + dw_model, mode='concat',
concat_axis=-1)
        # layer2
        nf = nf * 2
        model = Sequential()
        model.add(merged)
        model.add(Convolution2D(nb_filter=nf, nb_row=self.k_w, nb_col=1,
border_mode='same'))
        model.add(BatchNormalization(mode=2, axis=-1))
        model.add(PReLU())
        model.add(Flatten())
        # layer3
        model.add(Dense(self.n_hidden))
        model.add(BatchNormalization(mode=1, axis=-1))
        model.add(PReLU())
        # layer4
        model.add(Dense(int(np.sqrt(self.n_hidden))))
        model.add(PReLU())
        # output
        model.add(Dense(2 * self.n_stock))
        model.add(Reshape((self.n_stock, 2)))
```

```
        return model

    def transform_input(self, input):
        """Transform data into the Multi Scaled one

        Args:
            input: tensor with shape: [None, self.n_history, self.n_stock]
        Return:
            list of the same shape tensors, [None, self.length_history, self.n_stock]
        """
        # the last data is the newest information
        raw = input[:, self.n_history - self.history_length:, :, :]
        # smooth data
        smoothed = []
        for n_sm in range(2, self.n_smooth + 2):
            smoothed.append(
                tf.reduce_mean(tf.pack([input[:, self.n_history - st -
self.history_length:self.n_history - st, :, :]
                                        for st in range(n_sm)]),0))
        # downsample data
        down = []
        for n_dw in range(2, self.n_down + 2):
            sampled_ = tf.pack([input[:, idx, :, :]
                                for idx in range(self.n_history-n_dw*self.history_length,
self.n_history, n_dw)])
            down.append(tf.transpose(sampled_, [1, 0, 2, 3]))
        return raw, smoothed, down

    def load(self, checkpoint_dir):
        print(" [*] Reading checkpoints...")
        try:
            self.saver.restore(self.sess, self.save_path)
            return True
        except:
            return False
```

BIOGRAPHY

NAME	MR. SUTTA SORNMAYURA
ACADEMIC BACKGROUND	M.B.A. (2002), marketing and international business, Thammasat University, Thailand M.Eng.Sc. (1999), Biomedical Engineering, The University of New South Wales, Australia B.Eng. (1997), Electrical Engineering, King Mongkut Institute of Technology Thonburi, Thailand
EXPERIENCES	Lecturer, Industrial Management and Logistics Martin de tour school of management Assumption University, Thailand