Imperial College London

Imperial College London

Department of Mathematics

Reinforcement Learning With Continuous Controls For Foreign Exchange Trading

Author: Tzyy Shyang Tong (CID: 00402411)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2020-2021

Abstract

The Foreign Exchange (FX) marketplace is hugely complex driven by vast quantities of transactional and pricing data. Information that can be gleaned from the data offers opportunities for modelling and problem solving toward profitable, data-driven solutions. However, attempting to utilise forecasts from traditional supervised learning approaches can be difficult to translate into robust, profitable trading strategies [1]. In this thesis, a Deep Deterministic Policy Gradient (DDPG) model which can handle continuous controls is applied to derive the optimal strategies for an agent who trades in FX market. Within the DDPG framework, Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN) are used for the DDPG's actor and critic networks for performance comparison. Hindsight Experience Replay (HER) is also investigated as a means to augment trading performance.

In particular, this thesis will explain some procedures such as market data preparation, environment setup, model training, the derivation of reward function and the trading simulation for the training of the DDPG framework. A performance comparison between DDPG+MLP, DDPG+CNN, each with and without HER will be presented. The results demonstrate that DDPG+MLP models can perform better than DDPG+CNN models during some market conditions. Moreover, the results show that the HER algorithm can affect the inventories of DDPG+MLP and DDPG+CNN models and has some impact on the models' daily PnLs.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my internal supervisor Dr. Paul Bilokon for his continued support of my MSc research. His guidance helped me in all the time of research and writing of this thesis.

I would not have completed this project without the support of my external supervisors, Dr. David Lindsay and Dr. Sian Lindsay. Thank you for their encouragement, insightful comments and ideas on research and writing.

Last but not least, I would like to thank Algolabs for providing the market data for my research and experiments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

FX trading markets comprise many thousands of brokers serving millions of traders worldwide, trading currency pairs with each other in a large, decentralised global market that is open 24 hours a day, 5 days a week. Brokers offer bid and ask quotes which are the prices at which they will buy and sell at respectively, with the intention to profit from their transactions. The FX trading market is massively complex with automated or manual trades taking place over many different time horizons (e.g. minute-by-minute or daily) between many different types of brokerages and traders (i.e. large investment banks through to smaller hedge funds and to individuals trading via broker platforms on their mobile phones). The sheer volume and complexity of transactional and pricing data in FX trading is vast and, when appropriately labelled, formulated and analysed, offers quantitative traders (a.k.a. "Quants") a wealth of information on which to develop profitable data-driven trading models, algorithms or strategies.

Since the early 2000's there has been an ever-increasing use of machine learning techniques by Quants in financial firms such as hedge funds or investment banks that have access to vast amounts of historic market data. Quants have experimented with supervised- or unsupervised-learning and even reinforcement-learning (RL) techniques. Traditional supervised learning works best when the data is an 'independent identical data' (IID) type. However, FX market data comprises a set of consecutive time series which is time-dependent on other datasets; this temporal nature of market data is not suitable for traditional supervised learning tools such as linear regression, support vector machines (SVM), decision trees and so on [5]. Moreover Moddy and Saffell [1] demonstrated that the trading systems trained using reinforcement-learning could outperform the systems trained using supervised learning methods.

In reinforcement learning, an agent learns to act within a complex environment in order to maximise the total reward which is dependant upon the actions chosen from the policies. Therefore reinforcement learning provides a suitable framework to model complex financial systems such as Foreign Exchange (FX) trading.

This paper aims to apply the reinforcement learning (RL) with continuous controls to solve some complex financial problems presented in FX trading. With some trading constraints, we will try to use Deep Deterministic Policy Gradient (DDPG) framework with two types of networks: multi-layer perceptron (MLP) and convolutional neural network (CNN) to find out the optimal actions for FX trading. In Chapter 2, some background knowledge that is needed to understand some complex models used in this thesis will be introduced. It will cover the topics like FX trading, neural networks, reinforcement learning and its algorithms like DDPG and HER. The methodologies on how to structure the RL problem statements using Markov Decision Process (MDP) are mentioned in Chapter 3. Then in Chapter 4, the methodologies to set up the experiments and train the RL models will be explained. The results produced by trained models as well as the trading performance on out-of-sample data will be analysed and discussed in Chapter 5. Last but not least, the future work and conclusion will be specified in Chapter 6 and 7 respectively.

Chapter 2

Literature Reviews

Deep Reinforcement Learning (DRL) is the emerging subfield of Artificial Intelligence that is used to mimic how humans tackle the problem from the experience that was learnt from trial and error in daily life. Reinforcement learning becomes more popular and has been applied to many areas such as autonomous driving [6], healthcare [7], robotics [8], [9] and gaming sectors [10].

In robotic fields, Jan and Sethu [11] introduced Natural Actor-Critic algorithm to learn the non-linear dynamic motor primitives for humanoid robot control. The ability for the algorithm to tackle the learning of high-dimensionally continuous state-action systems offers a promising route for the development of RL system. Silver and his colleagues from DeepMind [12] presented the bespoke deep Q-learning combined with CNN to let the agent to play the Atari games. Several new concepts like replay buffers for the agents to learning from experience, new deep policy network designs and bespoke reward engineering methods were introduced. A few years later, Silver and Hannabis et al [13] introduced a revolutionary approach that lets the agent to use value and policy networks to evaluate the Go board game and to make the moves. Their algorithms also applied the Monte-Carlo Tree-Search (MCTS) algorithm to calculate the long term rewards in large search space. This was the first ever RL algorithm to tackle the full-sized game of Go and to defeat the human professional players. Since then RL becomes a popular framework to be used to tackle the problems within complex and dynamic environments. Financial sector is one of the sectors that starts to adopt the RL in automated trading to trade in complex financial market.

There are some literature on applying RL in trading. Moddy and Saffell [14] applied the differential Sharpe ratio to optimise the risk-adjusted return using RL framework. Souradeep [15] used a bespoke concept known as Financial Markov Decission Process (FMDP) to model a deep reinforcement learning framework which can make trading decisions on oil future and FX markets. The framework used deep Long-short term memory (LSTM) network to generate buy, hold and sell actions. More recently, Hongyang and Xiao Yang [16] adopted an ensemble strategy which combines multiple actor-critic algorithms: Advantage Actor Critic (A2C), Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO) to trade on US stock markets. They used the most recent historical data to validate the performance of three models first, then they picked the model with the best performance to trade for next few months. Alvaro & Sebastian [17] used double deep Q network learning (DDQN) and a new variant of reinforced deep Markov models (RDMM) to derive the optimal trading strategies for an agent who trades in a FX triplet. The RDMM can figure out the inventory constraints of the triplet currencies and generate the optimal actions for these triangular currencies.

Chapter 3

Background

3.1 Forex Trading

The foreign exchange (also known as FX or ForEx) market is the largest and most liquid asset market in the world. According to the Bank of International Settlements, there are more than \$5.1 trillion of forex being traded in volume daily. Unlike stock market where the stocks are traded in centralised order books managed by the stock exchanges, this international FX market has no central marketplace for trading. Rather, currency trading is conducted electronically over the counter (OTC) and quote-driven, which means that the dealers are working with other parties such as investment banks, commercial banks and broker-dealers to provide quoted prices for the customers to trade. The market is opened from 10pm on Sunday until 10pm on Friday, with 24 hours a day every week.

3.1.1 Foreign Exchange

Currencies are traded and priced in pairs. For example, a currency pair EUR/USD is quoted as 1.1256, EUR is the base currency and USD is the quote currency. Another perspective on currency trading comes from considering the position an investor is taking on each currency pair. The base currency can be thought as a short position, i.e. to sell the base currency in exchange of quote currency. In turn, the quote currency is a long position on the currency pair. In FX trading, the smallest unit to trade a currency pair is known as **lot** which has multiplier effect on the currency prices. For instance, if an investor buys one lot of EUR/USD at 1.1256 and sell at 1.1266, he will gain $\$112660 - \$112560 = \$100$, since one lot of EUR/USD has multiplier of 100000.

3.1.2 Technical Indicators

In the FX trading world, by analysing the historical data, the traders tend to use technical indicators to predict the future price movements. Technical indicators are heuristic or pattern-based signals that may be produced by the historical prices, volumes or open interest of stocks, derivatives or contracts in the market. Below are some of the most commonly used technical indicators:

- **Relative Strength Index (RSI)** quantifies the trend of the recent price changes as shown in equations below.

$$\begin{aligned} \boldsymbol{U}_t &= \begin{cases} close_t - close_{t-1}, & \text{if } close_t > close_{t-1} \\ 0, & \text{if } close_t < close_{t-1} \end{cases} \\ \boldsymbol{D}_t &= \begin{cases} close_{t-1} - close_t, & \text{if } close_t < close_{t-1} \\ 0, & \text{if } close_t > close_{t-1} \end{cases} \end{aligned} \tag{3.1.1}$$

$$\begin{aligned} \boldsymbol{RS_t} &= EMA(U_t, n)/EMA(D_t, n) \\ \boldsymbol{RSI_t} &= 100 - 100/(1 + RS_t) \end{aligned} \tag{3.1.2}$$

where RS is defined as the ratio of the n-day exponential moving average (EMA) of the U_t and D_t time series. Very often, $n = 14$ is used for EMA [18]. The range of the RSI is $[0, 100]$. If RSI moves below support line (<20), it indicates the stock is oversold, meaning the trader can perform buy action. If it moves above the resistance line (>70), it indicates the stock is overbought, then sell action can be performed [18].

- **Bollinger Band (BB)** interprets the strength of a trend and helps to identify the tops and bottoms of the market prices. Given a time series y_t at time $t = t^*$, n-day SMA is defined below.

$$\text{SMA}_{t^*} = \sum_{t=t^*-n+1}^{t=t^*} y_t/n \quad t^* = n, \dots, T \tag{3.1.3}$$

Next, the n-day rolling variance at time $t = t^*$, σ^2_{t*} is defined as:

$$\hat{\sigma}^2_{t^*} = \sum_{t=t^*-n+1}^{t=t^*} \left(y_t - \text{SMA}_{t^*}\right)^2/(n-1) \quad t^* = n, \dots, T \tag{3.1.4}$$

Since BB is formed by three oscillating bands: the upper band (UB), middle band (MB) and lower band (LB), then the relations of BB with 3.1.3 and 3.1.4 are constructed as below:

$$\begin{aligned} MB_{t^*} &= SMA_{t^*} \\ UB_{t^*} &= SMA_{t^*} + k * \hat{\sigma}_{t^*} \\ LB_{t^*} &= SMA_{t^*} - k * \hat{\sigma}_{t^*} \end{aligned} \tag{3.1.5}$$

where k is defined as the width of the band from the MB in standard deviation unit. Normally k=2 is used [19].

- **Commodity Channel Index (CCI)** is a momentum indicator that identifies the cyclical trends and trend reversals of the market price. Unlike RSI, CCI is an unbounded oscillator, meaning $CCI \in (-\infty, \infty)$. The calculation of CCI with n lookback days is presented below.

$$CCI_{t^*} = \frac{TP_{t^*} - SMA_{t^*}}{k * MD_{t^*}} \tag{3.1.6}$$

where

$$\begin{aligned} TP_{t^*} &= \sum_{t=t^*-n+1}^{t=t^*} \left(y^H_{t^*} + y^L_{t^*} + y^C_{t^*}\right)/3) \quad t^* = n, \dots, T \\ MD_{t^*} &= \sum_{t=t^*-n+1}^{t=t^*} |\left(TP_{t^*} - SMA_{t^*}\right)|/n) \quad t^* = n, \dots, T \end{aligned} \tag{3.1.7}$$

$y^H_{t^*}$, $y^L_{t^*}$, and $y^C_{t^*}$ refers to high, low and close prices. k is the multiplier of mean deviation (MD) between TP and SMA. According to Mansoor [20], k=0.015 is used due to approximately 70-80% of time stability of CCI value falls in between -100 and +100.

3.2 Deep Neural Network

Deep neural network is a type of neural network with more than two layers apart from input and output layers. In mathematical terms, deep neural network could be seen as non-linear function, $\boldsymbol{f} = (f_1, \ldots, f_O) : \mathbb{R}^I \to \mathbb{R}^O$ that turns $I \in \mathbb{N}(:= \{1, 2, \ldots, \}$, inputs $x_1, \ldots, x_I$ into $O \in \mathbb{N}$ outputs $f_1(x_1, \ldots, x_I), \ldots, f_O(x_1, \ldots, x_I)$.

3.2.1 Neural Network Properties

In a process of designing a network, we need to consider the type of the network architecture depending on the data structure of the inputs. The Multi-layer perceptron (MLP) and convolutional neural network (CNN) are the most commonly used networks in reinforcement learning systems. Also, within the network, the type of the activation functions for different layers need to be decided as well. Once the network design is decided, we need to choose a suitable **loss function** and **optimiser**. Then the network will be trained by using a technique called **backpropagation** [21].

A. Activation Function

The activation functions are normally placed after the nodes in a network layer. The most common activation functions are summarised in Table 3.1

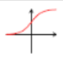
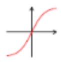
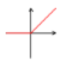
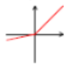
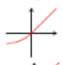
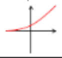
Activation	Chart	Definition	Derivative	Range
Sigmoid		$g(x) = \frac{1}{1+e^{-x}}$	$g'(x) = g(x)(1 - g(x))$	(0,1)
Hyperbolic Tangent (tanh)		$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$g'(x) = 1 - g(x)^2$	(-1,1)
Rectified Linear Unit (ReLU)		$g(x) = \max\{x, 0\}$	$g'(x) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases}$	$[0, \infty)$
Parametric ReLU (PReLU)		$g(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases} \alpha > 0$	$g'(x) = \begin{cases} \alpha, & x < 0 \\ 1, & x > 0 \end{cases}$	$\mathbb{R}$
Exponential Linear Unit (ELU)		$g(x) = \begin{cases} \alpha\left(e^x - 1\right), & x < 0 \\ x, & x \geq 0 \end{cases} \alpha > 0$	$g'(x) = \begin{cases} g(x) + \alpha, & x < 0 \\ 1, & x > 0 \end{cases}$	$(-\alpha, \infty)$
Softplus		$g(x) = \log\left(1 + e^x\right)$	$g'(x) = \frac{1}{1+e^{-x}}$	$[0, \infty)$

Table 3.1: The list of activation functions

B. Loss Function

The optimality of the neural network is determined by using a *loss function*. Mathematically, for a neural network $\boldsymbol{f} : \mathbb{R}^I \to \mathbb{R}^O, \boldsymbol{f} \in N_r(I, d_1, \ldots, d_{r-1}, O)$, the loss function ℓ is defined as

$$\ell = L(\hat{y}, \boldsymbol{y}), \text{ where } \hat{y} = \boldsymbol{f}(\boldsymbol{x}) \tag{3.2.1}$$

where $x \in \mathbb{R}^I$ is input and $y \in \mathbb{R}^O$ is reference value. In supervised learning, the value $y \in \mathbb{R}^O$ would be the *label*. Figure 3.1 illustrates some of the common loss functions used for different types of target outputs.

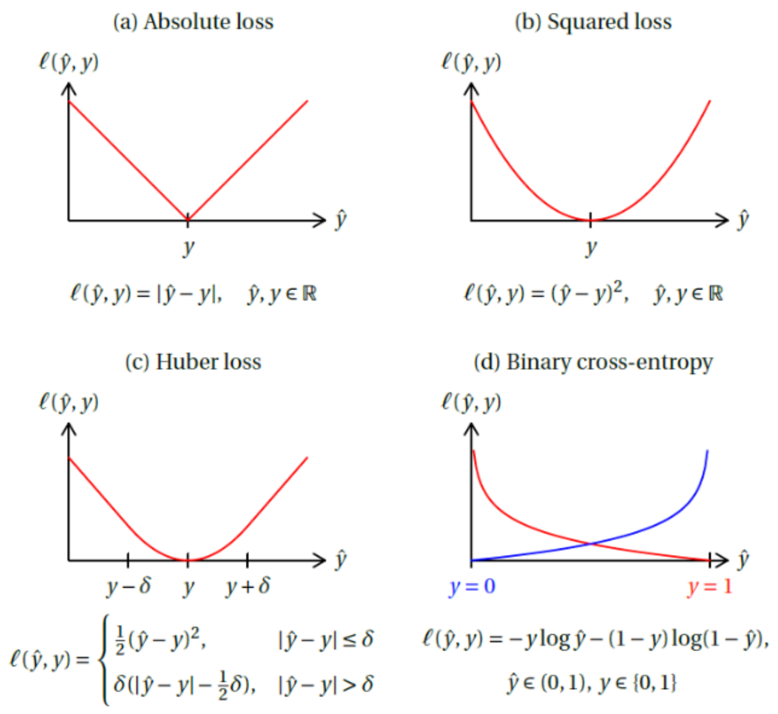


Figure 3.1: One-dimensional loss functions. Source: Deep Learning [2]

C. Optimiser

Stochastic gradient descent (SGD) technique is commonly used to optimise the loss function for network training. In SGD, a sample is selected randomly for each iteration. In general, SGD is computationally less expensive than typical gradient descent methods which take the whole data set for each iteration. Diederik and Jimmy [22] have proposed a momentum-based SGD algorithm called **ADAM** which shows that it performs better than other SGD algorithms such as Root Mean Square (RMS) Prop and AdaGrad. The algorithm for ADAM is defined in Algorithm 1.

3.2.2 Multi-Layer Perceptron (MLP)

The multi-layer perceptron is also known as feed-forward neural network (FNN). From M. Pakkanen [2], given input I and output O, a function $\boldsymbol{f} : \mathbb{R}^I \to \mathbb{R}^O$ is considered as feedforward neural network (FNN) with $r - 1 \in \{0, 1, \ldots\}$ hidden layers, where there are $d_i \in \mathbb{N}$ units in the i-th hidden layer for any $i = 1, \ldots, r - 1$, and activation functions $\boldsymbol{\sigma}_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}, i = 1, \ldots, r$, where $d_r := O$, if

$$\boldsymbol{f} = \boldsymbol{\sigma}_r \circ \boldsymbol{L}_r \circ \cdots \circ \boldsymbol{\sigma}_1 \circ \boldsymbol{L}_1 \tag{3.2.2}$$

where $\boldsymbol{L}_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$, for any $i = 1, \ldots, r$, is an affine function $\boldsymbol{L}_i(\boldsymbol{x}) := W^i \boldsymbol{x} + \boldsymbol{b}^i, \quad \boldsymbol{x} \in \mathbb{R}^{d_{i-1}}$.

The affine function has parameters presented as *weight matrix*, W^i

$$W^i = \left[W^i_{i,k}\right]_{j=1,\ldots,d_i, k=1,\ldots,d_{i-1}} \in \mathbb{R}^{d_i \times d_{i-1}} \tag{3.2.3}$$

and *bias vector* $\boldsymbol{b}^i = \left(b^i_1, \ldots, b^i_{d_i}\right) \in \mathbb{R}^{d_i}$, with $d_0 := I$. Such function should be denoted as

$$N_r\left(I, d_1, \ldots, d_{r-1}, O; \boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_r\right) \tag{3.2.4}$$

Algorithm 1 ADAM Stochastic Gradient Descent Algorithm

```
1: Require $\alpha$,                                   ▷ $\alpha$: Stepsize
2: Require $\beta_1, \beta_2$,          ▷ $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for moment estimates
3: Require $f(\theta)$,                 ▷ $f(\theta)$: Stochastic objective function with parameters $\theta$
4: Require $\theta_0$                                  ▷ $\theta_0$: Initial parameter vector
5: $m_0 \leftarrow 0$                                  ▷ Initialize $1^{st}$ moment vector
6: $v_0 \leftarrow 0$                                  ▷ Initialize $2^{nd}$ moment vector
7: $t \leftarrow 0$                                    ▷ Initialize time step
8: while $\theta_t$ not converged do
9:     $t \leftarrow t + 1$
10:    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$                       ▷ Get gradients w.r.t. stochastic objective at t
11:    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$           ▷ Update biased first moment estimate
12:    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$         ▷ Update biased second raw moment estimate
13:    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$                        ▷ Compute bias-corrected first moment estimate
14:    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$                        ▷ Compute bias-corrected second raw moment estimate
15:    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / \left(\sqrt{\widehat{v}_t} + \epsilon\right)$   ▷ Update parameters
16: return $\theta_t$                                  ▷ Resulting parameters
```

Figure 3.2 shows the graphical representation of the MLP network.

3.2.3 Convolutional Neural Network (CNN)

According to Keiron [23], one of the main differences between MLP and CNN is that the neurons within the layers of the CNN are comprised of neurons organised into three dimensions, which form the spatial dimensionality of the input (height, width and depth) and the filters. CNN used the convolution technique which is a linear filtering operation, ubiquitous in *signal processing*, *image processing* and *time series analysis*. This allows the network to encode the image-specific, time-specific or any successive input values are likely to be serially dependent, more efficiently compared to MLP. Figure 3.3 shows an example of the CNN architecture.

The CNN can be broken down into four main layers: *input layer*, *convolutional layer*, *pooling layer* and *fully-connected layer*.

- The **input layer** can take image, temporal data or signals which have spatial dimensions with height, width and depth.
- For one-dimensional **convolutional layer** with d inputs and kernel $\boldsymbol{k} = (k_{-l'}, \dots, k_l)$, in which $l, l' = 0, 1, \dots$ such that $l + l' < d$, is a function $C_k : \mathbb{R}^d \to \mathbb{R}^{d-l-l'}$ denoted as

$$C_k(x)_i := \sum_{j=-l'}^{l} k_j x_{i+j}, \quad i = l' + 1, \dots, d - l \tag{3.2.5}$$

  A convolutional layer with **stride** $s = 1, 2, \dots$ is a function $C_{k,s} : \mathbb{R}^d \to \mathbb{R}^{d'}$ is denoted as

$$C_{k,s}(x)_i := C_k(x)_i, \quad i \in \left\{ sj + l' + 1 : j \in \mathbb{Z}, 0 \le j \le \frac{d - l - l' - 1}{s} \right\} \tag{3.2.6}$$

  where $d' := \# \left\{ sj + l' + 1 : j \in \mathbb{Z}, 0 \le j \le \frac{d-l-l'-1}{s} \right\}$. Sometimes, we need to deal with the boundaries of the input vector $\mathbf{x}$ by decreasing dimensionality from d to $d - l - l'$. This could be solved by adding a layer, known as padding. It substitutes any out-of-bounds values of $\mathbf{x}$ with dummy data, normally initialised as zeros, as shown in Figure 3.4
- **Pooling layer** allows the network to downsample the output from the previous layer, by aggregating data in a sliding window with either maximum or averaging operation.

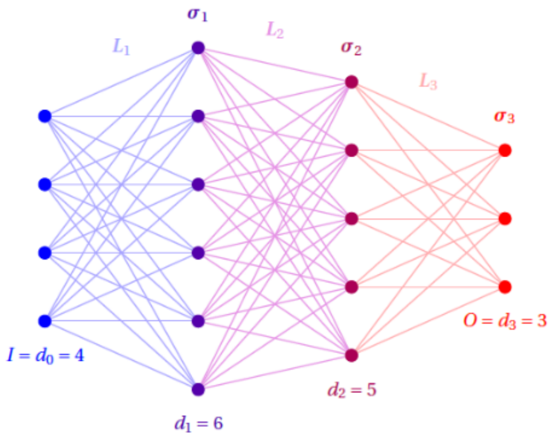


Figure 3.2: Graphical representation of a neural network with $r = 3, I = d_0 = 4, d_1 = 6, d_2 = 5$ and $O = d_3 = 3$. Source: Deep Learning [2]

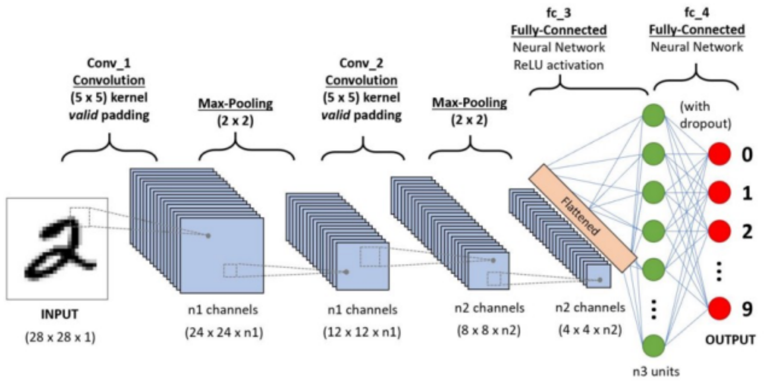


Figure 3.3: Graphical representation of a convolutional neural network. Source: TowardsDataScience.com [3]

A **max pooling layer** with d inputs, pool size $l = 2, 3, \ldots$ and stride $s = 1, 2, \ldots$ is a function $\boldsymbol{P}^{\max}_{l,s} : \mathbb{R}^d \to \mathbb{R}^{d'}$ denoted by

$$\boldsymbol{P}^{\max}_{l,s}(\boldsymbol{x_i}) := \max\left\{x_{s_{i+1}}, \ldots, x_{s_{i+l}}\right\}, \quad i \in \left\{j \in \mathbb{Z} : 0 \leq j \leq \frac{d-l}{s}\right\} \tag{3.2.7}$$

where $d' := \#\left\{j \in \mathbb{Z} : 0 \leq j \leq \frac{d-l}{s}\right\}$.

An **average pooling layer** with d inputs, pool size $l = 2, 3, \ldots$ and stride $s = 1, 2, \ldots$ is a function $\boldsymbol{P}^{\text{avg}}_{l,s} : \mathbb{R}^d \to \mathbb{R}^{d'}$ denoted by

$$\boldsymbol{P}^{\text{avg}}_{l,s}(\boldsymbol{x_i}) := \frac{x_{s_{i+1}} + \cdots + x_{s_{i+l}}}{l}, \quad i \in \left\{j \in \mathbb{Z} : 0 \leq j \leq \frac{d-l}{s}\right\}. \tag{3.2.8}$$

The definition of a pooling layer is illustrated in Figure 3.5.

- **Full-connected layer** is a MLP network in which its inputs are taken from the flattened outputs of CNN layers as illustrated in Figure 3.3.

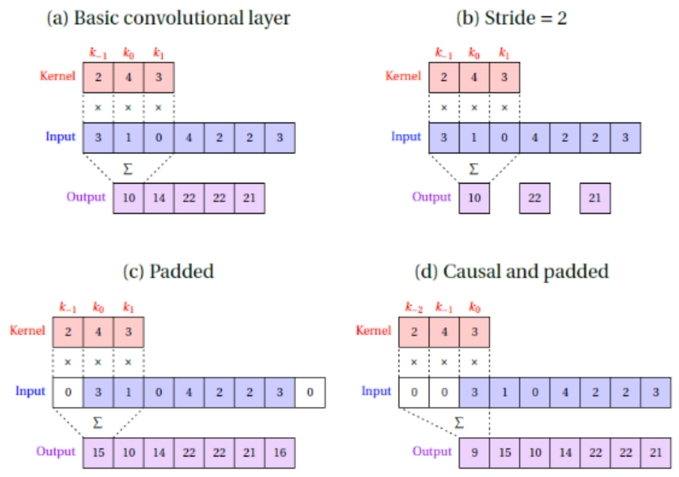


Figure 3.4: Convolutional layers with padding and other variations. Source: Deep Learning [2]

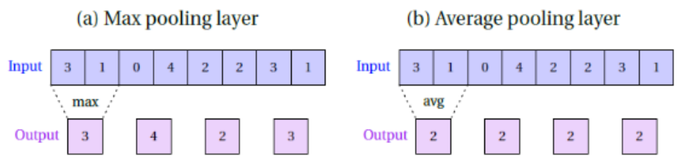


Figure 3.5: Illustration of pooling layers with pool size $l = 2$ and stride $s = 2$. Source: Deep Learning [2]

3.3 Reinforcement Learning

Reinforcement learning is a system that uses stochastic dynamic programming approach to make decisions that maximize rewards or minimize costs over a period of time. It involves the concept of trial-and-error learning from the decision-maker called agent through a dynamic environment. According to Sutton [4], other than the agent and the environment, the reinforcement learning system consists of four basic sub-elements: a policy, a value function, a reward function and a model of the environment. Recently, Some of the recently proposed RL algorithms are listed in Table 3.2. This thesis will focus only DDPG.

Name	Acronym	Policy	Action Space	State Space	Operator
State-**A**ction-**R**eward-**S**tate-**A**ction	SARSA	On-Policy	Discrete	Discrete	Q-value
Deep Q Network	DQN	Off-Policy	Discrete	Continuous	Q-value
Deep Deterministic Policy Gradient	DDPG	Off-Policy	Continuous	Continuous	Q-value
Asynchronous Advantage Actor-Critic	A3C	On-Policy	Continuous	Continuous	Advantage
Trust Region Policy Optimization	TPRO	On-Policy	Continuous	Continuous	Advantage
Proximal Policy Optimization	PPO	On-Policy	Continuous	Continuous	Advantage
Twin Delayed DDPG	TD3	Off-Policy	Continuous	Continuous	Q-value
Soft Actor-Critic	SAC	Off-Policy	Continuous	Continuous	Advantage

Table 3.2: The list of Reinforcement Learning algorithms.

3.3.1 Reinforcement Learning Properties

A. Policy

A policy $\pi(a|s)$ is defined as a mapping function that maps the current states of the environment to the corresponding actions that were taken when in those states. Depending on the state space size of the environment, the policy could be a simple function, a lookup table or even a deep neural network. In general, policy function is the core of a reinforcement learning agent as it mimics a human's behaviour that makes decisions in a new environment. It specifies how the agent's policy is changed as a result of its experience. In some cases in RL, policies may be stochastic by specifying probabilities for each action.

B. Reward

A reward signal quantifies the feedback value coming from the environment when it receives an action in a single step. The high reward value means it is more likely the agent will achieve the goal in the short term. In other words, the reward defines what are the good and bad events for the agent. The policy values are adjusted based on the reward signal. For instance, if an action selected by the policy is followed by a low reward, then the policy may be changed to select another action in that situation in the future.

C. Value Function

A value function specifies what is good in the long run. In other words, the value of a state s under policy π, denoted $v_\pi(s)$ is the total amount of reward that an agent can expect to accumulate over the future, starting from that state. The final element is a model of the environment which represents the behaviour of the environment. The model will predict the resultant next state and next reward when it's fed with a state and an action. The RL systems use models for planning, which means to decide on a course of action by considering many different possible future scenarios that have never been experienced before.

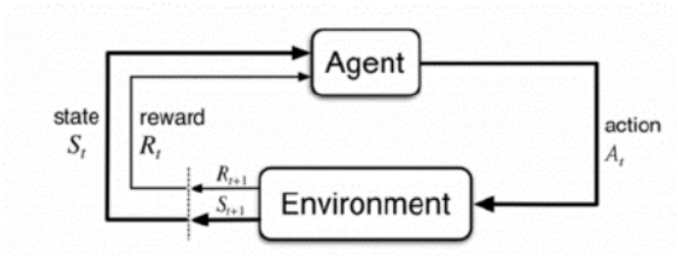


Figure 3.6: The interaction of agent and environment. Source: Sutton&Barto [4]

D. Environment

Figure 3.6 depicts how does the elements like agent, state, action and reward interact with each other within an environment in RL framework. At the t-th step of interaction, when the system has a state s_t, an agent chooses an action a_t among possible actions from a policy. The environment will take the action a_t and provide a numerical reward r_{t+1} in response. In the subsequent time period $t+1$, the learning agent chooses a next action a_{t+1} with the best reward value based on its past experiences (exploitation) or it may explore a new path by choosing a random action (exploration). This completes one step in the iteration process. As this process repeats, the system will find the optimal state-action pair that optimises the long-run reward. The environment is normally formulated by using Markov Decision Process (MDP) framework.

3.3.2 Markov Decision Process (MDP)

The reinforcement learning cycle described in Figure 3.6 can be formulated mathematically by using Markov Decision Process (MDP). The MDP and agent together thereby generate a sequence of *trajectory*: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$, and this process can be represented by $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P_a}, \boldsymbol{R_a})$ where

- $\boldsymbol{S}$ is a set of states called state space.
- $\boldsymbol{A}$ is a set of actions called action space. It can be discrete or continuous.
- $\boldsymbol{P_a}(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is probability that the action a in state s at time t leads to state s' at time $t+1$.
- $\boldsymbol{R_a}(s, s')$ is the immediate reward received after transit from state s to state s', due to action a.

The goal in a MDP is to find an optimal **policy** $\pi^*(s)$ for the agent. For MDP, we can define value function $v_\pi(s)$ and its optimal value $v_\pi^*(s)$ formally by

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right], \text{ for all } s \in \mathcal{S}, \tag{3.3.1}$$

$$v_*(s) = \max_\pi v_\pi(s) \text{ for all } s \in \mathcal{S} \tag{3.3.2}$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π at any time step t. Note that the value of the terminal state, if any, is always zero.

Similarly, we define $Q_\pi(s,a)$ as the value of taking action a in state s under a policy π and $Q^*_\pi(s)$ as its optimal value:

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right], \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{3.3.3}$$

$$Q^*(s,a) = \max_\pi Q_\pi(s,a), \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{3.3.4}$$

The equation 3.3.3 can be explained as the expected future returns starting from state s, taking the action a. The value functions v_π and Q_π can be estimated from experience. For the state-action pair (s,a), we can write Q^* in terms of v^* as follows:

$$Q^*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v^*\left(S_{t+1}\right) \mid S_t = s, A_t = a\right] \tag{3.3.5}$$

Both value functions v_π and q_π can be estimated by using Monte Carlo method, dynamic programming (DP), neural networks etc. The MDP framework is abstract and flexible and can be applied to many different problems in many areas such as trading, robotics, autonomous driving, games etc.

3.3.3 Deep Deterministic Policy Gradient (DDPG)

Deep deterministic policy gradient (DDPG), proposed by Lilicrap et al. [24], is an algorithm which uses actor-critic techniques to learn a Q-function and a policy concurrently. It is an off-policy algorithm. In contrast to Deep Q-Network (DQN) which can only handle discrete action space, DDPG can only be used for environments with *continuous* action spaces. The key approach for DDPG is to compute the optimal action $a^*(s)$ in any state s from the optimal action-value function $Q^*(s,a)$ by solving:

$$a^*(s) = \arg\max_a Q^*(s,a) \tag{3.3.6}$$

When there are a finite number of discrete actions, it is easy to compute the action which maximises the Q-values directly. However for continuous action space, we cannot exhaustively evaluate the space and also it is unacceptably expensive to compute $\arg\max_a Q^*(s,a)$ by using conventional optimisation algorithm. Therefore Lillicrap [24] proposed some novel techniques on learning a Q-function and a policy to solve $Q^*(s,a)$.

First, the DDPG algorithm uses **target policy network** to approximate $\arg\max_a Q^*(s,a)$ by minimising the following mean square error (MSE) function with stochastic gradient descent (SGD) method:

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)Q_{\phi_{\text{targ}}}\left(s', \pi_{\theta_{\text{targ}}}\left(s'\right)\right)\right)\right)^2\right], \tag{3.3.7}$$

where $\pi_{\theta_{\text{targ}}}$ is the target policy. $(s,a,r,s',d) \sim \mathcal{D}$ means taking the a batch of sample transitions (s,a,r,s',d) from experience replay buffer $\mathcal{D}$, which is a finite-sized memory cache, in each iteration.

Second, since the action space is continuous and assuming the Q-function is differentiable w.r.t action, to learn a deterministic policy $\mu_\theta(s)$ which gives the action that maximises $Q_\phi(s,a)$, SGD is used to solve the following:

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathrm{E}}\left[Q_\phi\left(s, \pi_\theta(s)\right)\right] \tag{3.3.8}$$

Then the parameters between the original networks ϕ and target networks ϕ_{targ} are updated once in each iteration by averaging:

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi \tag{3.3.9}$$

where ρ is a hyper-parameter between 0 and 1.

Last but not least, to make the DDPG policies explore better, the author suggests to add stochastic time-correlated Ostein-Urlenbeck (OU) noise $\mathcal{N}$ to actor policy $\pi'_\theta\left(s_t\right) = \pi_\theta(s_t) + \mathcal{N}$. The pseudocodes for DDPG is shown in Algorithm 2.

3.3.4 Hindsight Experience Replay (HER)

DDPG algorithm uses experience replay buffer of past experiences to sample a finite batch size of training examples $(s, a, r, s', d) \sim \mathcal{D}$ to update the neural networks. Given a behaviour policy $\pi(.)$, at each step t associated with a transition tuple (s, a, r, s', d), the agent can generate a trajectory $\tau = \{(s_0, a_0), \cdots, (s_{T-1}, a_{T-1})\}$ of any length T. In many multi-goal RL tasks, the reward only depends on whether the trajectory can reach a desired goal g or not. In most of the settings, only the successful trajectories get non-negative rewards. Since the policy $\pi(.)$ is not fully-trained and has low success rate, the collected successful trajectories are usually insufficient for training, which results in the sparse reward problem.

Mancini [25] addressed the sparse reward problem by suggesting Hindsight Experience Replay (HER) algorithm. This algorithm treats the failed experience as successes and learns from them again. For any off-policy RL algorithm like DQN, DDPG, NAF and etc, HER modifies the memorised states s and s' from transition tuple (s, a, r, s', d) with the desired goal state g_t and g_{t+1}. In this research, the concepts of desired goal g_t and achieved goal g'_t are used. The desired goal g_t is the actual goal that the agent aims to achieve, and the achieved goal g'_t is a state that the agent has already achieved. For example, in a golf game, a golf ball could be modelled if landing in a sand pit by receiving a reward $r_t = -1$ (achieved state could be the ball's position in the sand pit), then from the agent's perspective, the desired goal state for golf ball should be on the green (desired goal state should be ball's position on the green) with $r_t = 1$. Once the achieved goal g'_t is replaced by a desired goal g_t, the corresponding failed experience is assigned with a desired non-negative reward and is thus added to replay buffer $\mathcal{D}$. This will provide more successful experiences for the agent which can lead to better learning policies. The pseudocodes for HER are displayed in Algorithm 3.

Algorithm 2 DDPG Algorithm

```
 1: Randomly initialise critic-network $Q_\phi(s,a)$ and actor network $\pi_\theta(s)$ with weights $\phi$ and $\theta$
 2: Initialise target network $Q_{\phi_{\text{targ}}}$ and $\pi_{\theta_{\text{targ}}}$ with weights $\phi^{\text{targ}} \leftarrow \phi$, $\theta^{\text{targ}} \leftarrow \theta$
 3: Empty replay buffer $D$
 4: for episode=1, M do
 5:     Observe state s
 6:     Select action $a_t = \pi_\theta(s) + \mathcal{N}_t$ according to current policy and exploration noise
 7:     Execute $a_t$ in the environment
 8:     Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
 9:     Store $(s,a,r,s',d)$ in replay buffer $\mathcal{D}$
10:     If $s'$ is terminal, reset environment state.
11:     if it's time to update then
12:         for $\eta$=1,N do                ▷ $\eta$ is optimization step size used to train the network
13:             Randomly sample a batch of transitions, $B = (s,a,r,s',d)$ from $\mathcal{D}$
14:             Compute targets
                    $$y(r,s',d) = r + \gamma(1-d)Q_{\phi_{\text{targ}}}\left(s', \pi_{\theta_{\text{targ}}}(s')\right) \quad (3.3.10)$$
15:             Update the critic (Q-function) by one step of gradient by minimising the loss:
                    $$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_\phi(s,a) - y(r,s',d)\right)^2 \quad (3.3.11)$$
16:             Update the actor policy by one step of gradient ascent
                    $$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \pi_\theta(s)) \quad (3.3.12)$$
17:             Update the target networks:
                    $$\begin{aligned} \phi_{\text{targ}} &\leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi \\ \theta_{\text{targ}} &\leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta \end{aligned} \quad (3.3.13)$$
```

Algorithm 3 HER Algorithm

```
1: Given
     • an off-policy RL-algorithm $\mathbb{A}$,                      ▷ e.g. DQN, DDPG, NAF, SQDN
     • a strategy $\mathbb{S}$ for sampling goals for replay,        ▷ e.g. $\mathbb{S}(\mathtt{s_0}, \ldots, \mathtt{s_T}) = \mathtt{m}(\mathtt{s_T})$
     • a reward function $r : \mathcal{S}$ x $\mathcal{A}$ x $\mathcal{G} \rightarrow \mathbb{R}$      ▷ e.g. $\mathtt{r}(\mathtt{s}, \mathtt{a}, \mathtt{g}) = -[\mathtt{f_g}(\mathtt{s}) = 0]$
2: Initialise $\mathbb{A}$                                           ▷ initialize neural networks
3: Initialise replay buffer $D$
4: for episode=1, M do
5:     Sample a goal $g$ and an initial state $s_0$
6:     for t=0, T-1 do
7:         Sample an action $a_t$ using the behavioral policy from $\mathbb{A}$:
8:         $a_t \leftarrow \pi_b\,(s_t \| g)$                        ▷ || denotes concatenation
9:         Execute the action $a_t$ and observe a new state $s_{t+1}$
10:    for t=0, T-1 do
11:        $r_t := r\,(s_t, a_t, g)$
12:        Store the transition $(s_t \, \| g, a_t, r_t, s_{t+1} \| \, g)$ in $D$     ▷ Standard experience replay
13:        Sample a set of additional goals for replay $G := \mathbb{S}($ current episode $)$
14:        for $g' \in G$ do
15:            $r' := r\,(s_t, a_t, g')$
16:            Store the transition $(s_t \, \| g', a_t, r', s_{t+1} \| \, g')$ in $D$       ▷ HER
17:    for t=1, N do
18:        Sample a minibatch $B$ from the replay buffer $D$
19:        Perform one step of optimization using $\mathbb{A}$ and minibatch $B$
```

Chapter 4

Methodology

This chapter describes the ways to do FX trading using the reinforcement learning system. Before we use RL framework for automated trading, we use a novel way to build the FX Market Markov Decision Process (MDP-FX) for the system setup. Within this framework, we use the DDPG models with two different types of neural network architectures (MLP and CNN) as the actor-critic functions. By going through the training processes, the optimal policy of this deep reinforcement learning framework could be achieved.

For the trading simulation, we set some trading constraints and make some assumptions:

- **No slippage cost**: The market is very liquid, hence the orders can be executed immediately with the latest prices $\mathbf{p_t} \in \{\mathbf{p^{Bid}}, \mathbf{p^{Ask}}\}$ for all currency pairs $\mathbb{S}$ at time t, where $\mathbf{p_t^{Bid}}$ and $\mathbf{p_t^{Ask}}$ be the vector of the bid-ask prices for currency pairs $\mathbb{S}$.

- **No leverage**: The agent is not allowed to borrow money for trading. This means that the account balance b_t is always non-negative, $b_t \geq 0$. We set $\mathbf{x_i^B}$ as buy amount and $\mathbf{x_i^S}$ as sell amount for each currency pair $i \in \mathbb{S}$, then after each transaction, the balance will be updated as
$$b_{t+1} = b_t - \mathbf{p_t^{Ask}} \mathbf{x_t^B} + \mathbf{p_t^{Bid}} \mathbf{x_t^S} \geq 0 \tag{4.0.1}$$

- **Non-negative position**: During the trading, the agent's position or inventory values $\mathbf{I}_t$ are non-negative for each currency pair $i \in \mathbb{S}$.
$$I_{i,t+1} = I_{i,t} + p_{i,t}^{Ask} * x_{i,t}^B - p_{i,t}^{Bid} * x_{i,t}^S \geq 0, i \in \mathbb{S} \tag{4.0.2}$$

- **Transaction cost**: In real trading world, there are many types of transaction costs such as commission fees, overnight interest rate charges, execution fees and etc. For our case, we set the total costs of the every transaction as
$$c_t = \mathbf{p^T} \mathbf{x_t} * 0.1\%, \text{where } \mathbf{x_t} \in \{\mathbf{x^B}, \mathbf{x^S}\} \tag{4.0.3}$$

- **Intra-day trading**: The trading agent will perform intra-day trading only. This means that it trades only at certain time in a day and it has to close all of its currency pairs' positions at the end of trading hour T, i.e. $\mathbf{I_{t=T}} = 0$. When the agent closes all position at the last minutes, we assume no slippage cost and all transactions can be executed immediately.

At next few sections, we will explain how would the above trading rules and constraints can be adopted to MDP framework and used for the trading environment setup.

4.1 Markov Decision Process for FX Market (MDP-FX)

Inspired by the previous works done by Souradeep [15], which suggested a MDP framework specific to stock market, we create a generalized bespoke MDP framework for FX trading market known as MDP-FX. This MDP framework treats the trading agent as an trading algorithm which will make the trading decisions based on the observations from the FX market as well as from its account states. The aim of the agent is to generate consistent profits (PnL) with optimal inventory controls in the dynamic environment with the trading constraints mentioned above. Also the methods on how to setup the FX trading environment for simulation will be mentioned below.

4.1.1 State Space

The MDP-FX is defined in such a way that an agent trades on any currency pairs with some trading constraints in a dynamic market. During the trading hours, a trading agent needs to take care of its inventory and balance while observing the market data. The agent relies on the technical indicators to make the trading decisions to buy (long) and sell (short) the suitable amount of currency pairs at certain time t. Technical indicators are useful as they could capture some statistical properties or market behaviours of the market. With these in minds and inspired by Hongyang[16], the state space for the MDP-FX consists of account state and the technical indicators.

During trading simulation, at each time t, the agent needs to record the cash balance b_t, the mid-price $\mathbf{p_t^i}$ of currency pair $i \in \mathbb{S}$ and the latest inventory of currency j, I_t^i. **Seven** technical indicators chosen:

- Moving Average $\mathbf{M_t^W} \in \mathbb{R}^{\mathbb{N}}$
- Bollinger Band (consists of upper band $\mathbf{U_t^W} \in \mathbb{R}^{\mathbb{N}}$ and lower band $\mathbf{L_t^W} \in \mathbb{R}^{\mathbb{N}}$)
- Commodity Channel Index (CCI) $\mathbf{C_t^W} \in \mathbb{R}^{\mathbb{N}}$
- Average True Range $\mathbf{A_t^W} \in \mathbb{R}_+^{\mathbb{N}}$
- Percentage return of each time bar $\mathbf{R_t^W} \in \mathbb{R}^{\mathbb{N}}$
- Relative Strength Index (RSI) $\zeta_\mathbf{t}^\mathbf{W} \in [\mathbf{-100}, \mathbf{100}] \in \mathbb{R}_+^{\mathbb{N}}$

The superscript W is the lookback window used for calculation. For example, if $L = 14$, then

$$\mathbf{M_t^{14}} = \frac{1}{14} \sum_{t=t^*-14+1}^{t=t^*} p_t, \quad t^* = 14, \dots, T$$

is the 14-day moving average.

Assuming an agent trades α currency pairs in its portfolio and each technical indicator has β lookback windows to calculate, then the state space size $|\mathbf{S}| = 7 * \beta + 2 * \alpha + 1$. The state space for MLP input can be represented by a vector $\mathbf{S_t} \in \mathcal{R}^{|\mathbf{S}|}$

$$\mathbf{S_t} = [\mathrm{b_t}, \mathbf{p_t^1} \dots \mathbf{p_t^\alpha}, \mathbf{I^1}_t \dots \mathbf{I_t^\alpha}, \mathbf{M_t^{W_1}} \dots \mathbf{M_t^{W_\beta}}, \mathbf{L_t^{W_1}} \dots \mathbf{L_t^{W_\beta}},$$
$$\mathbf{U_t^{W_1}} \dots \mathbf{U_t^{W_\beta}}, \mathbf{C_t^{W_1}} \dots \mathbf{C_t^{W_\beta}}, \mathbf{A_t^{W_1}} \dots \mathbf{A_t^{W_\beta}}, \mathbf{R_t^{W_1}} \dots \mathbf{R_t^{W_\beta}}, \zeta_\mathbf{t}^\mathbf{W_1} \dots \zeta_\mathbf{t}^\mathbf{W_\beta}]$$

For example, if an agent trades with only 1 currency pair X and each technical indicator has 1 lookback window $W_1 = 20$. The state space for MLP input is represented as $|S| = 10$-dimensional vector:

$$\mathbf{S_t} = [\mathrm{b_t}, \mathbf{p_t^X}, \mathbf{I^X}_t, \mathbf{M_t^{20}}, \mathbf{L_t^{20}}, \mathbf{U_t^{20}}, \mathbf{C_t^{20}}, \mathbf{A_t^{20}}, \mathbf{R_t^{20}}, \zeta_\mathbf{t}^\mathbf{20}]$$

The state space for CNN input consists a consecutive series of features, so it is represented by a matrix $\mathbf{S_t} \in \mathcal{R}^{T*|S|}$ as shown below. To simplify the expression, we assume the input with

T consecutive steps has only one currency pair X and 1 lookback window W for all technical indicators.

$$\mathbf{S_t} = \begin{bmatrix} \mathrm{b}_{t0} & \mathbf{p^X_{t0}} & \mathbf{I^X}_{t0} & \mathbf{M^W_{t0}} & \mathbf{L^W_{t0}} & \mathbf{U^W_{t0}} & \mathbf{C^W_{t0}} & \mathbf{A^W_{t0}} & \mathbf{R^W_{t0}} & \zeta^{\mathbf{W}}_{\mathbf{t0}} \\ \mathrm{b}_{t1} & \mathbf{p^X_{t1}} & \mathbf{I^X}_{t1} & \mathbf{M^W_{t1}} & \mathbf{L^W_{t1}} & \mathbf{U^W_{t1}} & \mathbf{C^W_{t1}} & \mathbf{A^W_{t1}} & \mathbf{R^W_{t1}} & \zeta^{\mathbf{W}}_{\mathbf{t1}} \\ \vdots & & & & \vdots & & & & & \vdots \\ \mathrm{b}_{T} & \mathbf{p^X_{T}} & \mathbf{I^X}_{T} & \mathbf{M^W_{T}} & \mathbf{L^W_{T}} & \mathbf{U^W_{T}} & \mathbf{C^W_{T}} & \mathbf{A^W_{T}} & \mathbf{R^W_{T}} & \zeta^{\mathbf{W}}_{\mathbf{T}} \end{bmatrix}$$

4.1.2 Action Space

We define $\mathbf{a_t} = [a_{1,t}, \ldots, a_{\alpha,t}]$ as a vector of action values for α currency pairs traded by the agent. They are continuous values generated by the policy π in DDPG and ranged within $[-1, 1]^\alpha \in \mathbb{R}^\alpha$. In the simulation, during the trading hour $t = 1 \ldots T-1$ before the end of the day, the environment will multiply $a_{i,t}$ by a multiplier K_i of currency pair i and we denote the executed action $a^E_{i,t} = a_{i,t} * K_i$. Since the agent needs to close all of the positions at the end of the day, then the agent have to sell $a^E_{i,T} = -1 * \mathbf{I^i_T} * p^i_T$ for each currency pair i at time T as shown below:

$$\mathbf{a^E_t} = \begin{cases} \mathbf{a_t K}, & \text{if } t = 1 \ldots T-1 \\ -\mathbf{I_t p_t}, & \text{if } t = T \end{cases}$$

The range of the action space for $\mathbf{a^E}$ is therefore $(-\infty, 0] \bigcup [-K, K] = (-\infty, K]$. $\mathbf{K}$ is a predefined parameter.

4.1.3 State Transition

The state value $\mathbf{S_t}$ consists of portfolio state and technical indicators. For each time step t, the cash balance b_t and inventories $\mathbf{I^1}_t \ldots \mathbf{I^\alpha_t}$ for α number of currency pairs, which are part of $\mathbf{S_t}$, are updated in response to the action $\mathbf{a_t}$ received. By considering the trading constraints we defined earlier, the state changes for the buy (long) and sell (short) transactions for a currency pair i are denoted below. c is the fraction of transaction cost.

- **Buy (long) within trading hours**, i.e. $a_{i,t} > 0, \quad t = 0, \ldots T-1$

$$\begin{aligned} x^B_{i,t} &= \min[K_i * a_{i,t} * p_{i,t} * (1+c),\ b_t] \\ y^B_{i,t} &= \min[K_i * a_{i,t} * p_{i,t},\ x^B_{i,t}] \\ I_{i,t+1} &= I_{i,t} + y^B_{i,t} \\ b_{t+1} &= b_t - x^B_{i,t} \end{aligned}$$

  where c_t is the transaction cost %, $x^B_{i,t}$ is buy amount with transaction cost included, $y^B_{i,t}$ is buy amount without transaction cost. To long $x^B_{i,t}$ amount of currency pair i, the agent needs to check its balance if it has enough cash. If there is not enough cash in its account then it will only long $x^B_{i,t} = b_t$ amount of currency pair i.

- **Sell (short) within trading hours**, i.e. $a_{i,t} < 0, \quad t = 0, \ldots T-1$

$$\begin{aligned} x^S_{i,t} &= \min[K_i * a_{i,t} * p_{i,t} * (1-c),\ I_{i,t} * p_{i,t} * (1-c)] \\ y^S_{i,t} &= \min[K_i * a_{i,t} * p_{i,t},\ I_{i,t} * p_{i,t}] \\ I_{i,t+1} &= I_{i,t} - y^S_{i,t} \\ b_{t+1} &= b_t + x^S_{i,t} \end{aligned}$$

  where $x^S_{i,t}$ is sell amount with transaction cost included, $y^S_{i,t}$ is sell amount without transaction cost. To short $x^S_{i,t}$ amount of currency pair i, the agent needs to check its inventory if it has enough position for selling, since negative position is not allowed. If there is not enough cash position in its inventory then it can only short $x^S_{i,t} = I_{i,t} * p_{i,t}$ amount of currency pair i.

- **Close all positions at the end of the trading day**, i.e. $t = T$

$$\begin{aligned}
&x^S_{i,T} = I_{i,T-1} * p_{i,t} * (1-c) \\
&I_{i,T} = 0 \\
&b_T = b_{T-1} + x^S_{i,T}
\end{aligned}$$

  The above state changes are straight forward, the agent just needs to sell all of the currency pair i in its inventory at the end of the day.

4.1.4 Reward

The reward function $r(s_t, a_t, s_{t+1})$ at time t is defined as below.

$$\boldsymbol{r}(s_t, a_t, s_{t+1}) = \boldsymbol{r}_A + \boldsymbol{r}_W - \boldsymbol{\alpha}_t \tag{4.1.1}$$

where

$$\boldsymbol{r}_A = \begin{cases} \min(\log x_t, 1.0), & \text{if } x_t \geq \epsilon_A \\ -\min(\log(|x_t|), 1.0), & \text{if } x_t \leq -\epsilon_A \\ 0, & \text{otherwise} \end{cases} \tag{4.1.2}$$

$$\boldsymbol{r}_W = \begin{cases} 3.0, & \text{if } y_t > \epsilon_W \\ 1.0, & \text{if } 0.5 * \epsilon_W < y_t \leq \epsilon_W \\ 0.2, & \text{if } 0.0 < y_t \leq 0.5 * \epsilon_W \\ -0.2, & \text{if } -0.5 * \epsilon_W < y_t \leq 0.0 \\ -1.0, & \text{if } -\epsilon_W < y_t \leq -0.5 * \epsilon_W \\ -3.0, & \text{if } y_t \leq -\epsilon_W \end{cases} \tag{4.1.3}$$

$$\boldsymbol{\alpha}_t = |\tanh(a^E_t - a_t)| * \epsilon_\alpha \tag{4.1.4}$$

ϵ_A, ϵ_W and ϵ_α are the predefined thresholds which can be set before the simulation. From equation 4.1.1, the reward function is broken into three parts: cash asset reward r_A, PnL reward r_W and mismatch action penalty α_t.

$r_A \in (-1, 1)$ is designed to give reward (penalty) if the trade is profitable (loss) when there is change in portfolio balance from b_t to b_{t+1}. We define $x_t = b_{t+1} - b_t$ as the short term PnL with only one step t after the agent buys or sells from time t to $t+1$. ϵ_W is the predefined parameter for the threshold of x_t.

r_W (equation 4.1.3) is a step-function which considers longer term PnL with lookback period of N steps from current t. Similar to x_t, $y^N_t = b_t - b_{t-N}$ is the difference of portfolio balance from b_{t-N} to b_t, but it reflects longer term PnL which should have more advantages for trading. Therefore, $r_W \in [-3, 3]$ has higher reward or loss compared to r_A. ϵ_W is the predefined parameter for the threshold of y_t.

Sometimes, the $a_{i,t}$ assigned by the policy π may not be executed in the environment if the agent does not have enough cash to long or does not have any currencies from the inventory to short as described in section 4.1.3. For example, if $a_{i,t} = 0.5610$ is assigned, it means to long $0.5610 * K$ currency pair i at time t. However, the agent has zero balance (no cash) in its account, i.e. $b_t = 0$, then execution action will be zero, i.e. $a^E_{i,t} = 0$. This misalignment between the assigned action $a_{i,t}$ and execution action $a^E_{i,t}$ will cause the instability in network training. Therefore, we use $\alpha_t \in [-\epsilon_\alpha, \epsilon_\alpha]$ to give more penalty when there's bigger discrepancies between assigned and execution action.

4.2 Actor-Critic Network Architecture

From Algorithm 2, two types of networks are needed: critic networks ($Q_\phi(s,a)$ and $Q_{\phi_{targ}}(s,a)$) and actor networks ($\pi_\theta(s)$ and $\pi_{\theta_{targ}}(s)$). We will describe the network architectures used for DDPG-MLP and DDPG-CNN.

4.2.1 DDPG-MLP

A. Actor network architecture

Table 4.1 illustrates the network architecture for DDPG-MLP's actor function $\pi_\theta(s)$. Recall the description from section 3.2.2, the actor function $a_t = \pi_\theta(s) = \mathcal{N}_r(I, N_1, N_2, O)$. This means that the network has an input layer one-dimensional input size I, two hidden layers (first hidden layer has N_1 nodes and second hidden layer has N_2 nodes) and then an output layer with N_{Action} outputs. Batch normalisation (BN) layers are used after two hidden layers to normalise the inputs that could have large variances within the network. This can help to stabilise the parameters by not generating large output values. Activation functions ReLU are used after BN layers. The range for the output needs to be $(-1, 1)$, therefore activation function tanh is used to generate the action values $a_t \in (-1, 1)$. In summary, the network takes a batch size of environment states (observations) as input and then generate action values as output.

Layer No	Layer Type	Nodes
L0	Input Layer	I
L1	Linear Layer (Hidden 1)	N_1
B1	Batch Normalisation Layer	N_1
R1	ReLU	N_1
L2	Linear Layer (Hidden 2)	N_2
B2	Batch Normalisation Layer	N_2
R2	ReLU	N_2
L3	Output Layer	N_{action}
R3	Tanh	N_{action}

Table 4.1: Actor network architecture for model DDPG-MLP with or without HER

B. Critic network architecture

Table 4.2 shows the network architecture of DDPG-MLP's critic function $Q_\phi(s,a)$. The critic network has similar architecture as actor network. However, as the critic function has two inputs: environment state (observation) and action, one extra simple neural network (known as NN2) with only one linear layer, which takes the actions as the inputs, is created. The outputs of NN2 network are then combined with second hidden layer (L2) of the main network (NN1). The output layer has only one node to generate a single value. The value produced from this critic network that can be called as state-action value or Q-value which has the range $(-\infty, \infty)$. Therefore activation *tanh* is not needed after the output layer.

4.2.2 DDPG-CNN

A. Actor network architecture

The actor network architecture for DDPG-CNN is more complicated than MLP network. From Table 4.3, the network consists of two parts: CNN with two 1D-convolutional layers and MLP network with two hidden layers. It takes the environment state (observation) s with input size of

Layer No	Layer Type (NN1)	Nodes (NN1)	Layer Type (NN2)	Nodes (NN2)
L0	Input Layer	I		
L1	Linear Layer (Hidden 1)	N_1		
B1	Batch Normalisation Layer	N_1		
R1	ReLU	N_1		
L2	Linear Layer (Hidden 2)	N_2		
B2	Batch Normalisation Layer	(N_2)		
E0			Input Layer	N_{action}
E1			Linear Layer	N_2
E2			ReLU	N_2
L2(C)	Combined B2 with E2 ($L2 + E2$)	N_2		
R2	ReLU	N_2		
L3	Output Layer	1		

Table 4.2: Critic network architecture for model DDPG-MLP with or without HER

(T, N_1) as input. T is the time step size and N_1 is the feature size of the state s. Imagine the s as a 2-dimensional image, the T is the height and N_1 is the width of the image. The input is then passed to 1D-convolutional layer with N_{F1} of filters. Within CNN, leaky-ReLU is used as the negative values should not be ignored as they could be useful for preventing the dying ReLU problem. Leaky-ReLU is a parametric-ReLU (PReLU) with $\alpha = 0.01$. Dropout layer is used after CR1 to prevent overfitting during the network training.

The outputs after CR2 is flattened into full-connected layer with node size $N_{F2} * W$ and

$$\begin{aligned} W_1 &= N_1 \\ W_{n+1} &= \frac{W_n - \kappa + 2\rho}{\ell} + 1, n = 1 \dots C \\ W &= W_C \end{aligned}$$

where C is the number of convolutional layers in CNN, κ is the kernel size, ρ is padding (at here, $\rho = 0$) and ℓ is the stride. Full-connected layer is actually a linear layer. The other layers after L1 have similar architecture as MLP shown in previous section. The output of the network is $a_t \in (-1, 1)$.

Layer No	Layer Type	Nodes
C1	Convolutional Layer (1D)	(N_T, N_{F1})
CR1	LeakyReLU	(N_1, N_{F1})
CD1	Dropout Layer (2D)	(N_1, N_{F1})
C2	Convolutional Layer (1D)	(N_{F1}, N_{F2})
CR2	LeakyReLU	(N_{F1}, N_{F2})
L1	Full-Connected Layer	$N_{F2} * W$
L2	Linear Layer (Hidden 1)	N_3
LB1	Batch Normalisation Layer	N_3
LR1	ReLU	N_3
L3	Linear Layer (Hidden 2)	N_4
LB3	Batch Normalisation Layer	N_4
LR3	ReLU	N_4
L4	Linear Layer	N_{action}
O	Tanh (Output)	N_{action}

Table 4.3: Actor network architecture for model DDPG-CNN with or without HER

B. Critic network architecture

The critic network architecture for DDPG-CNN model is similar to actor network and since $Q_\phi(s, a)$ has two inputs: environment state s and action a, one extra simple neural network (known as NN2) with only one linear layer, which takes the a as the input, is created. The rest is the same design described in section B of 4.2.1 to generate Q-value for DDPG-CNN. Table 4.4 illustrates the CNN architecture of the critic network.

Layer No	Layer Type (NN1)	Nodes (NN1)	Layer Type (NN2)	Nodes (NN2)
C1	Convolutional Layer (1D)	(N_1, N_{F1})		
CR1	LeakyReLU	(N_1, N_{F1})		
CD1	Dropout Layer (2D)	(N_1, N_{F1})		
C2	Convolutional Layer (1D)	(N_{F1}, N_{F2})		
CR2	LeakyReLU	(N_{F1}, N_{F2})		
L1	Full-Connected Layer	$N_{F2} * W$		
L2	Linear Layer (Hidden 1)	N_3		
LB2	Batch Normalisation Layer	N_3		
LR2	ReLU	N_3		
L3	Linear Layer (Hidden 2)	N_4		
LB3	Batch Normalisation Layer	N_4		
E0			Input Layer	N_{action}
E1			Linear Layer	N_4
E2			ReLU	N_4
L3(C)	Combined LB3 with E2 $(LB2 + E2)$	N_4		
LR3	ReLU	N_4		
O	Linear Layer	1		

Table 4.4: Critic network architecture for model DDPG-CNN with or without HER

4.3 Hindsight Experience Replay (HER) Modelling

In trading world, a trader does not just to maximise the PnL or Sharpe ratio, but also needs to keep other constraints in mind. One of the most important constraints is the inventory control, this is particularly important for intra-day trader. Due to the intra-day trading constraint specified before, it's always good for the trader to have smaller inventories or positions before the end of the trading day. Hence, coupled with DDPG framework, the HER algorithm is used for intra-day trading's inventory control.

We propose some simple mathematical models as illustrated in Figure 4.1 for the desired amount of cash b_t^D and the desired currency position I_t^D during intra-day trading. After some mathematics derivations, the equations for b_t^D and I_t^D are shown in 4.3.1 and 4.3.2.

$$b_t^D = \frac{4 * b_t^{max} * t^2}{T^2} - \frac{4 * b_t^{max} * t}{T} + b_t^{max} \tag{4.3.1}$$

$$I_t^D = -\frac{4 * I_t^{max} * t^2}{T^2} + \frac{4 * I_t^{max} * t}{T} \tag{4.3.2}$$

where b_t^{max} and I_t^{max} are maximum cash and inventory allowed during the trading hours, T is the total time steps in a trading day. The thoughts for the above models are simple, we hope the trading agent has maximum position at mid-day $t = T/2$ and no position at the end of the day $t = T$ to minimise the risk when the agent close all position at the end of the trading day.

4.3.1 DDPG-MLP

To apply HER for inventory control, for each time step t, the agent calculates b_{t-1}^D, b_t^D, I_{t-1}^D and I_t^D using the equations 4.3.1 and 4.3.2. The concepts of desired goal and achieved goal are shown in Table 4.5. The tuple $\theta = (s_t, a_t, r_t, s_{t+1}, d, \eta, \eta', \gamma, \gamma')$ generated after each action step a_t will be stored into a memory buffer $\mathbb{T}$ known as transition memory. If $d = 1$, it means the trading simulation for all periods has completed or the end of an episode.

At the end of an episode, the agent will scan though all of the transition memory $\mathbb{T}$. For each tuple θ taken from HER experiences, the agent replaces the cash balance b_t and positions I_t within s_t and s_{t+1} with the desired goals γ and γ' if some criteria are met (see Algorithm 4). Also the reward value r_t will be updated. Last but not least, these modified experiences with

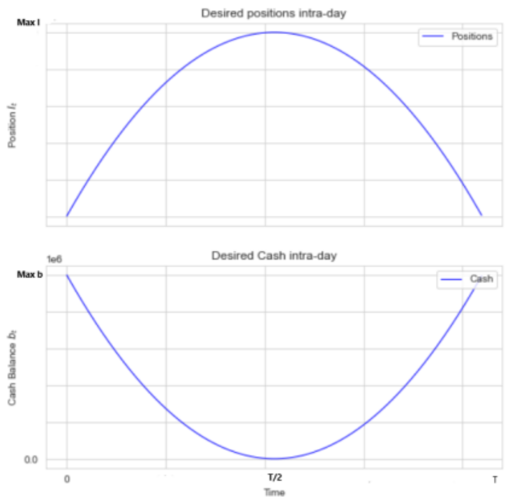


Figure 4.1: The models to control cash balance and position intra-day

$\theta = (s_t, a_t, r_t, s_{t+1}, d, \eta, \eta', \gamma, \gamma')$ improvements will be stored into the memory buffer known as HER experiences $\mathbb{H}$. This means that the agent will have more good experiences for training to achieve its optimal inventory control during intra-day trading guided by the above mathematics equations. This is summarised in Algorithm 4.

Algorithm 4 HER for DDPG-MLP

```
1:  procedure ApplyHER($\mathbb{T}$)   ▷ Apply HER function for each step t, $\mathbb{T}$ is transition memory
2:      Initialise $\mathbb{H}$   ▷ Initialise HER experiences
3:      for $\theta = (s_t, a_t, r_t, s_{t+1}, d, \eta, \eta', \gamma, \gamma')$ in $\mathbb{T}$ do
4:          $(b_t^D, I_t^D) \leftarrow \gamma$
5:          $(b_{t+1}^D, I_{t+1}^D) \leftarrow \gamma'$
6:          $(b_t, I_t) \leftarrow \eta$
7:          $(b_{t+1}, I_{t+1}) \leftarrow \eta'$
8:          if $|\eta - \gamma| > \epsilon$ then   ▷ If achieved goal and current goal have big difference
9:              Take $(b_t, I_t) \in s_t$   ▷ Get cash and position from current state s
10:             Take $(b_{t+1}, I_{t+1}) \in s_{t+1}$   ▷ Get cash and position from next state s
11:             $b_t, I_t \leftarrow b_t^D, I_t^D$
12:             $b_{t+1}, I_{t+1} \leftarrow b_{t+1}^D, I_{t+1}^D$
13:             $r_t \leftarrow r_t * 0.2$   ▷ Reduce the penalty value as $r_t$ is negative
14:         Store updated $(s_t, a_t, r_t, s_{t+1}, d)$ into $\mathbb{H}$   ▷ store the goal state into HER experiences
15:     Return $\mathbb{H}$
```

Name	Symbol	Value	Description
Current desired goal	γ	(b_t^D, I_t^D)	Desired cash balance and position calculated from the models above at time t
Next desired goal	γ'	(b_{t+1}^D, I_{t+1}^D)	Desired cash balance and position calculated from the models above at time t+1
Current achieved goal	η	(b_t, I_t)	Cash balance and position taken from trading account during trading hour above at time t
Next achieved goal	η'	(b_{t+1}, I_{t+1})	Latest cash balance and position taken from trading account during trading hour above at time $t+1$

Table 4.5: Explanation of desired goal and achieved goal used for DDPG-MLP

Name	Symbol	Value	Description
Current desired goal	γ	$(b^D_{t-W} \dots b^D_t, I^D_{t-W} \dots I^D_t)$	Desired cash balance and position calculated from the models above from time t-W to t (with lookback period W)
Next desired goal	γ'	$(b^D_{t-W+1} \dots b^D_{t+1}, I^D_{t-W+1} \dots I^D_{t+1})$	Desired cash balance and position calculated from the models above at from time t-W+1 to t+1
Current achieved goal	η	$(b_{t-W} \dots b_t, I_{t-W} \dots I_t)$	Cash balance and position taken from trading account during trading hour above from time t-W to t
Next achieved goal	η'	$(b_{t-W+1} \dots b_{t+1}, I_{t-W+1} \dots I_{t+1})$	Latest cash balance and position taken from trading account during trading hour above from time t-W+1 to t+1

Table 4.6: Explanation of desired goal and achieved goal used for DDPG-CNN

4.3.2 DDPG-CNN

For DDPG-CNN, HER algorithm is used similarly as DDPG-MLP. Since CNN contains series of states $\mathbf{s}$ with W steps for observation fed into the networks, then each item within the tuple $\theta = (\mathbf{s_t}, a_t, r_t, \mathbf{s_{t+1}}, \mathbf{d}, \eta, \eta', \gamma, \gamma')$ is the vector (except a_t and r_t) which contains series of W elements. Table 4.6 shows the differences between the DDPG-MLP and DDPG-CNN for desired and achieved goals. Algorithm 5 summarises the HER usage for inventory control.

Algorithm 5 HER for DDPG-CNN

```
1: procedure ApplyHER($\mathbb{T}$)    ▷ Apply HER function for each step t, $\mathbb{T}$ is transition memory
2:     Initialise $\mathbb{H}$    ▷ Initialise HER experiences
3:     for $\theta = (\mathbf{s_t}, a_t, r_t, \mathbf{s_{t+1}}, \mathbf{d}, \eta, \eta', \gamma, \gamma')$ in $\mathbb{T}$ do
4:         $(b^D_{t-W} \dots b^D_t, I^D_{t-W} \dots I^D_t) \leftarrow \gamma$
5:         $(b^D_{t-W+1} \dots b^D_{t+1}, I^D_{t-W+1} \dots I^D_{t+1}) \leftarrow \gamma'$
6:         $(b_{t-W} \dots b_t, I_{t-W} \dots I_t) \leftarrow \eta$
7:         $(b_{t-W+1} \dots b_{t+1}, I_{t-W+1} \dots I_{t+1}) \leftarrow \eta'$
8:         if $|\mathbb{E}[\eta] - \mathbb{E}[\gamma]| > \epsilon$ then    ▷ If achieved goal and current have big difference
9:             Take $(b_{t-W} \dots b_t,\ I_{t-W} \dots I_t) \in \mathbf{s_t}$ ▷ Get W consecutive steps of cash and positions
   from current state s
10:            Take $(b_{t-W+1} \dots b_{t+1},\ I_{t-W+1} \dots I_{t+1}) \in \mathbf{s_{t+1}}$
11:            $(b_{t-W} \dots b_t, I_{t-W} \dots I_t) \leftarrow (b^D_{t-W+1} \dots b^D_{t+1}, I^D_{t-W+1} \dots I^D_{t+1}$
12:            $(b_{t-W+1} \dots b_{t+1},\ I_{t-W+1} \dots I_{t+1}) \leftarrow (b^D_{t-W+1} \dots b^D_{t+1}, I^D_{t-W+1} \dots I^D_{t+1}$
13:            $r_t \leftarrow r_t * 0.2$    ▷ Reduce the penalty value as $r_t$ is negative
14:        Store updated $(\mathbf{s_t}, a_t, r_t, \mathbf{s_{t+1}}, \mathbf{d})$ into $\mathbb{H}$    ▷ store the goal state into HER experiences
15:    Return $\mathbb{H}$
```

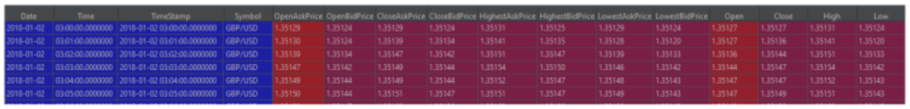
Chapter 5

Experiment

Using the methods described in Chapter 4, the experiments are setup for the model training and testing by using the Foreign Exchange (FX) market data. We use the experiments to investigate if it's possible to obtain the optimal continuous actions on the selected trading days using DDPG with different type of neural networks and with or without HER.

5.1 Data

We use the FX market data provided by AlgoLabs. The data is aggregated into minute bar data as shown in 5.1. We use one currency pair GBP/USD for the experiment to simplify the training process of the DDPG algorithm.

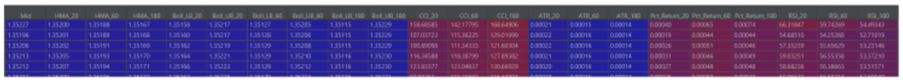


Date	Time	TimeStamp	Symbol	OpenAskPrice	OpenBidPrice	CloseAskPrice	CloseBidPrice	HighestAskPrice	HighestBidPrice	LowestAskPrice	LowestBidPrice	Open	Close	High	Low
2018-01-02	03:00:00.0000000	2018-01-02 03:00:00.0000000	GBP/USD	1.35129	1.35124	1.35129	1.35124	1.35131	1.35125	1.35129	1.35124	1.35127	1.35127	1.35131	1.35124
2018-01-02	03:01:00.0000000	2018-01-02 03:01:00.0000000	GBP/USD	1.35130	1.35124	1.35139	1.35134	1.35141	1.35135	1.35128	1.35120	1.35127	1.35136	1.35141	1.35120
2018-01-02	03:02:00.0000000	2018-01-02 03:02:00.0000000	GBP/USD	1.35139	1.35134	1.35147	1.35142	1.35151	1.35147	1.35139	1.35133	1.35136	1.35144	1.35151	1.35133
2018-01-02	03:03:00.0000000	2018-01-02 03:03:00.0000000	GBP/USD	1.35147	1.35142	1.35149	1.35144	1.35154	1.35150	1.35146	1.35142	1.35144	1.35147	1.35154	1.35142
2018-01-02	03:04:00.0000000	2018-01-02 03:04:00.0000000	GBP/USD	1.35149	1.35144	1.35149	1.35144	1.35152	1.35147	1.35148	1.35143	1.35147	1.35147	1.35152	1.35143
2018-01-02	03:05:00.0000000	2018-01-02 03:05:00.0000000	GBP/USD	1.35150	1.35144	1.35151	1.35147	1.35151	1.35147	1.35149	1.35143	1.35147	1.35149	1.35151	1.35143

Figure 5.1: The screenshot of the raw data used for the experiment

5.1.1 Data Processing

The raw data is used to calculate the technical indicators mentioned in 3.1.2 and 4.1.3. Figure 5.2 and Figure 5.3 show the sample data for technical indicators (21 features) and the visualisation of the technical indicators' time series that are inserted into the networks respectively.



Mid	HMA_20	HMA_60	HMA_180	Boll_LR_20	Boll_UR_20	Boll_LR_60	Boll_UR_60	Boll_LR_180	Boll_UR_180	CCI_20
1.35227	1.35200	1.35188	1.35167	1.35158	1.35217	1.35127	1.35285	1.35115	1.35229	158.68585
1.35196	1.35201	1.35189	1.35168	1.35160	1.35217	1.35128	1.35206	1.35115	1.35229	107.03723
1.35206	1.35202	1.35191	1.35169	1.35162	1.35219	1.35129	1.35208	1.35115	1.35229	100.89098
1.35213	1.35205	1.35193	1.35170	1.35164	1.35221	1.35129	1.35210	1.35116	1.35230	116.34588
1.35212	1.35207	1.35194	1.35171	1.35166	1.35223	1.35129	1.35212	1.35116	1.35230	123.83377

CCI_60	CCI_180	ATR_20	ATR_60	ATR_180	Pct_Return_20	Pct_Return_60	Pct_Return_180	RSI_20	RSI_60	RSI_180
142.17795	168.64906	0.00021	0.00015	0.00014	0.00040	0.00065	0.00074	66.31847	59.74269	54.49343
115.36225	129.01999	0.00022	0.00016	0.00014	0.00019	0.00044	0.00044	54.88510	54.25268	52.71019
111.34333	121.60304	0.00022	0.00016	0.00014	0.00026	0.00051	0.00046	57.35238	55.65629	53.23146
118.38799	127.89382	0.00021	0.00016	0.00014	0.00031	0.00046	0.00049	59.03251	56.55358	53.57210
123.04637	130.68929	0.00020	0.00016	0.00014	0.00027	0.00048	0.00048	58.68236	56.38663	53.51571

Figure 5.2: The screenshot of the technical indicators (21 features) used for the experiment

Before the state or observation **S** is fed into actor and critic networks, they need to be normalised first. The state **S** consists of two types of data set: account state and technical indicators. For the technical indicators data set, we calculate the global historical mean $\hat{\mu}$ and standard deviation $\hat{\sigma}$ from the whole historical data set of GBP/USD. Then we perform standard normalisation using equation 5.1.1 on the values of all technical indicators. The normalisation for balance $\hat{\boldsymbol{b}}$ and position

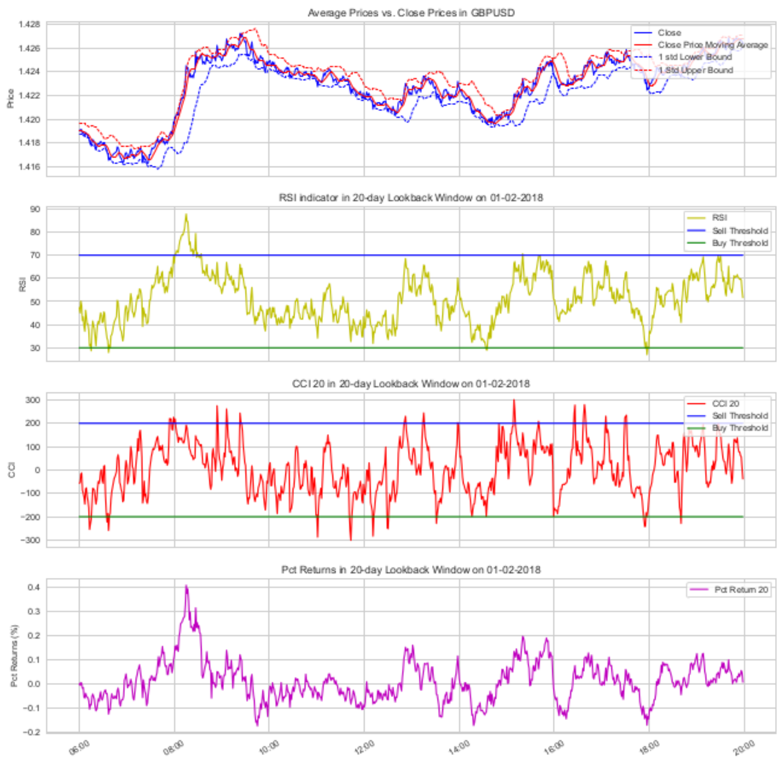


Figure 5.3: The visualisation of time series of technical indicators that are inserted into the networks

$\boldsymbol{\hat{I}}$ are shown in equation 5.1.2 and 5.1.3 respectively.

$$\boldsymbol{Z}_t^{GBPUSD} = \frac{v_t^{GBPUSD} - \mu^{GBPUSD}}{\sigma^{GBPUSD}} \tag{5.1.1}$$

$$\boldsymbol{\hat{b_t}} = \frac{b_t}{K} \tag{5.1.2}$$

$$\boldsymbol{\hat{I_t}} = \frac{I_t * \boldsymbol{\hat{p_t}}}{K} \tag{5.1.3}$$

where $\hat{p}_t$ is the global mean for GBP/USD mid price, K is the initial amount of cash investment for trading. For our case, the trading agent has the principle money $b_0 = 1000000$ (in USD).

5.2 FX Trading Environment Set Up

We use Gym API designed by OpenAI [26] to setup the FX trading environment according to the trading rules and constraints specified in Chapter 4. PyTorch is used to implement the actor-critic networks and Stable-Baseline3 API [27] are used to log the results into Tensorboard.

5.2.1 Environment Steps

As we use only GBP/USD for experiment, we can assume a_t is the buy or sell proportion for GBP/USD. This section describes how the trading agent will respond after receiving action a_t from the actor network. The flow of the trading steps within the simulated environment is displayed in Algorithm 6. We initialise $K = 10000$, $b_0 = \$100000$. Also we assume there is no cap limit on trading bet size, i.e. no clip size for each bet. For the reward function 4.1.1, we use $\epsilon_A = 5$, $\epsilon_W = 100$ and $\epsilon_\alpha = 10$.

Algorithm 6 Action steps for FX trading

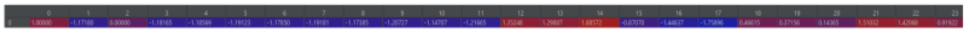
```
1: $K \leftarrow 10000$                                   ▷ Multiplier for GBP/USD
2: $b_0 \leftarrow 1000000$
3: $I_0 \leftarrow 0$                                      ▷ Position GBP/USD = 0 at t=0
4: Initialise $Done \leftarrow False$
5: $t \leftarrow 0$
6: $c \leftarrow 0.001$                                    ▷ Transaction cost fraction
7: procedure STEP($a_t$)                                   ▷ Action function for each step t
8:     if $t < T$ then
9:         $\hat{a_t} \leftarrow a_t * K$
10:        if $\hat{a_t} < 0$ then                         ▷ Sell action
11:            if $I_t > 0$ then
12:                $x_t^S \leftarrow \min(K * a_t * p_t, I_t * p_t) * (1 - c)$,
13:                $y_t^S \leftarrow \min(K * a_t * p_t, I_t * p_t)$
14:                $b_{t+1} \leftarrow b_t + x_t^S$
15:                $I_{t+1} \leftarrow I_t - y_t^S$
16:                $a_t^E \leftarrow -x_t^S / K$
17:            else
18:                $a^E \leftarrow 0$
19:        else if $\hat{a_t} > 0$ then                   ▷ Buy action
20:            if $b_t > 0$ then                           ▷ when cash ¿ 0
21:                $x_t^B \leftarrow \min(K * a_t * p_t * 1 - c, b_t)$,
22:                $y_t^B \leftarrow \min(K * a_t * p_t, x_t^B)$
23:                $b_{t+1} \leftarrow b_t - x_t^B$
24:                $I_{t+1} \leftarrow I_t + y_t^B$
25:                $a_t^E \leftarrow x_t^B / K$
26:            else
27:                $a^E \leftarrow 0$
28:    else if $t = T$ then
29:        $x_t^S \leftarrow I_{T-1} * p_{T-1} * (1 - c)$,    ▷ Close all position
30:        $b_T \leftarrow b_{T-1} + x_t^S$
31:        $I_T \leftarrow 0$
32:        $a_t^E \leftarrow x_t^S / K$
33:        $d \leftarrow True$
34:    $s_{t+1} \leftarrow s_t$
35:    Calculate reward $r_t$
36:    return $s_{t+1}$, $r_t$, $d$, infos
```

5.3 Model Training

We run the training mode of each DDPG model in a high performance computer which has 32 CPU cores and 32GB of RAMs.

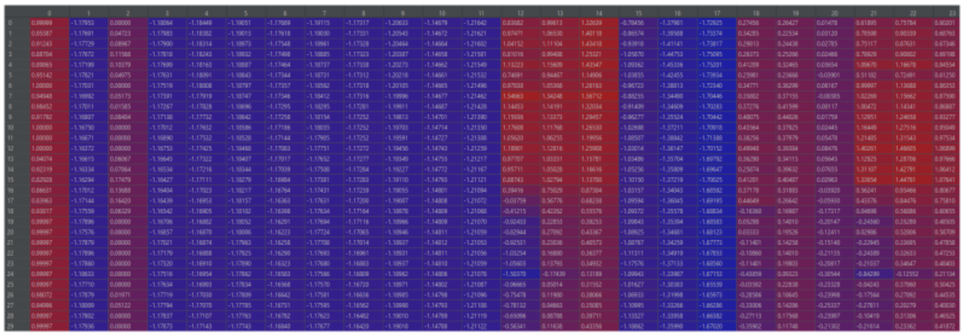
5.3.1 Training Data

To train the model, we take the GBP/USD data from 02-01-2018 to 02-02-2018 as in-sample training data. For each day, only the minute data from 6am to 8pm are taken and there are 840 minute-bar data per day (i.e. 60 bars per hour per for GBP/USD currency pair). The train data is organised as described in 4.1.3, for each technical indicator, we use three lookback windows $W_1 = 20$, $W_2 = 60$ and $W_3 = 180$. Hence, by taking $\alpha = 1$ (one currency pair) and $\beta = 3$, we will have $|\mathbf{S}| = 2 * 1 + 7 * 3 + 1 = 24$ elements in the state space $\mathbb{S}$. We use $T = 30$ consecutive steps to form a 30x24 two-dimensional matrix for CNN state input. Figure 5.4 and 5.5 show the examples for the normalised MLP and CNN input states.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	1.00000	-1.17180	0.00000	-1.18165	-1.18569	-1.19123	-1.17850	-1.19181	-1.17385	-1.20727	-1.14787	-1.21665	1.35248	1.29887	1.88572	-0.87070	-1.44637	-1.75896	0.46615	0.37156	0.14365	1.51032	1.42068	0.91922

Figure 5.4: Example of normalised MLP input



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0.99999	-1.17953	0.00000	-1.18064	-1.18449	-1.19051	-1.17889	-1.19115	-1.17317	-1.20633	-1.14679	-1.21642	0.83682	0.99813	1.32639	-0.78456	-1.37981	-1.72925	0.27456	0.26427	0.01478	0.81895	0.75784	0.80201
1	0.95387	-1.17691	0.04723	-1.17983	-1.18382	-1.19013	-1.17618	-1.19030	-1.17331	-1.20543	-1.14672	-1.21621	0.97471	1.06530	1.40118	-0.86574	-1.39588	-1.73374	0.34285	0.22534	0.03120	0.78598	0.90339	0.68763
2	0.91243	-1.17729	0.08967	-1.17900	-1.18314	-1.18973	-1.17548	-1.18961	-1.17328	-1.20464	-1.14664	-1.21602	1.04152	1.11104	1.43418	-0.93918	-1.41141	-1.73817	0.29813	0.24438	0.02785	0.75117	0.87631	0.67346
3	0.88704	-1.17672	0.11568	-1.17818	-1.18243	-1.18932	-1.17498	-1.18885	-1.17323	-1.20387	-1.14658	-1.21581	0.81016	0.99488	1.25321	-1.05870	-1.44753	-1.75095	0.28373	0.25266	0.02486	0.78829	0.90802	0.69198
4	0.89865	-1.17199	0.10579	-1.17699	-1.18163	-1.18887	-1.17464	-1.18737	-1.17338	-1.20273	-1.14662	-1.21549	1.13223	1.15609	1.43547	-1.09362	-1.45336	-1.75201	0.41209	0.32465	0.03654	1.09670	1.16678	0.84554
5	0.95142	-1.17821	0.04975	-1.17631	-1.18091	-1.18843	-1.17344	-1.18731	-1.17312	-1.20218	-1.14661	-1.21532	0.74691	0.94467	1.14906	-1.03855	-1.42455	-1.73934	0.25981	0.23666	-0.03901	0.51182	0.72491	0.61250
6	1.00000	-1.17031	0.00000	-1.17519	-1.18008	-1.18797	-1.17257	-1.18582	-1.17318	-1.20105	-1.14665	-1.21496	0.97038	1.05368	1.28163	-0.96723	-1.38813	-1.72340	0.34771	0.36299	0.08167	0.99997	1.13688	0.86353
7	0.94948	-1.16992	0.05173	-1.17391	-1.17919	-1.18747	-1.17346	-1.18412	-1.17316	-1.19996	-1.14677	-1.21462	1.54663	1.34248	1.56712	-0.88255	-1.34490	-1.70446	0.35802	0.37155	-0.00385	1.02269	1.15662	0.87590
8	0.98452	-1.17011	0.01585	-1.17267	-1.17828	-1.18696	-1.17295	-1.18295	-1.17281	-1.19911	-1.14687	-1.21428	1.14453	1.14191	1.32034	-0.91409	-1.34609	-1.70283	0.37276	0.41599	0.00117	1.00472	1.14341	0.86887
9	0.91762	-1.16807	0.08404	-1.17138	-1.17732	-1.18642	-1.17258	-1.18154	-1.17252	-1.19813	-1.14701	-1.21390	1.15936	1.13373	1.29457	-0.96277	-1.35524	-1.70442	0.40075	0.44026	0.01759	1.12951	1.24658	0.93277
10	1.00000	-1.16750	0.00000	-1.17012	-1.17632	-1.18586	-1.17186	-1.18035	-1.17252	-1.19703	-1.14714	-1.21350	1.17608	1.11768	1.28530	-1.02698	-1.37211	-1.70818	0.43564	0.37825	0.02445	1.16449	1.27516	0.95049
11	1.00000	-1.16671	0.00000	-1.16890	-1.17532	-1.18528	-1.17144	-1.17905	-1.17252	-1.19591	-1.14727	-1.21308	1.05620	1.06255	1.19956	-1.08507	-1.38842	-1.71388	0.38256	0.37979	0.05478	1.21485	1.31543	0.97534
12	1.00000	-1.16372	0.00000	-1.16753	-1.17425	-1.18488	-1.17083	-1.17751	-1.17272	-1.19456	-1.14743	-1.21259	1.18901	1.12816	1.25908	-1.03014	-1.38147	-1.70152	0.49948	0.39304	0.08478	1.40261	1.48605	1.06899
13	0.94074	-1.16615	0.06067	-1.16645	-1.17322	-1.18407	-1.17017	-1.17652	-1.17277	-1.19349	-1.14755	-1.21217	0.97707	1.03351	1.15781	-1.03486	-1.35704	-1.69792	0.36290	0.34115	0.05645	1.12925	1.28706	0.97666
14	0.92319	-1.16334	0.07864	-1.16534	-1.17216	-1.18344	-1.17039	-1.17508	-1.17284	-1.19227	-1.14772	-1.21167	0.95711	1.05028	1.16616	-1.05256	-1.35809	-1.69647	0.25874	0.39632	0.07655	1.31107	1.42791	1.06412
15	0.82928	-1.16294	0.17479	-1.16427	-1.17111	-1.18279	-1.16984	-1.17381	-1.17283	-1.19110	-1.14793	-1.21121	0.88743	1.02794	1.13700	-1.10150	-1.37219	-1.70025	0.41201	0.40487	0.02963	1.33854	1.44761	1.07841
16	0.86631	-1.17012	0.13688	-1.16404	-1.17023	-1.18217	-1.16764	-1.17431	-1.17239	-1.19055	-1.14801	-1.21094	0.39416	0.75029	0.87504	-1.03157	-1.34043	-1.68992	0.37178	0.31883	-0.03920	0.56241	0.93466	0.80677
17	0.83963	-1.17144	0.16420	-1.16439	-1.16953	-1.18157	-1.16363	-1.17631	-1.17200	-1.19007	-1.14808	-1.21072	-0.03759	0.56776	0.68238	-1.09594	-1.36045	-1.69195	0.44649	0.26642	-0.05930	0.43376	0.84476	0.75810
18	0.93017	-1.17559	0.06329	-1.16542	-1.16905	-1.18102	-1.16398	-1.17634	-1.17164	-1.18970	-1.14809	-1.21060	-0.41215	0.42262	0.55576	-1.09372	-1.35578	-1.68834	-0.16360	0.16907	-0.17317	0.04898	0.56086	0.60655
19	0.99997	-1.17896	0.00000	-1.16706	-1.16882	-1.18052	-1.16291	-1.17694	-1.17116	-1.18986	-1.14809	-1.21070	-0.92433	0.22853	0.38253	-1.09843	-1.35394	-1.68583	0.05298	0.14010	-0.20147	-0.24360	0.35289	0.48505
20	0.99997	-1.17576	0.00000	-1.16857	-1.16870	-1.18006	-1.16223	-1.17724	-1.17065	-1.18946	-1.14811	-1.21059	-0.82944	0.27092	0.43367	-1.08925	-1.34681	-1.68123	0.03333	0.19526	-0.12411	0.02986	0.52806	0.58709
21	0.99997	-1.17879	0.00000	-1.17021	-1.16874	-1.17963	-1.16258	-1.17708	-1.17014	-1.18937	-1.14812	-1.21053	-0.92531	0.23036	0.40573	-1.08787	-1.34259	-1.67773	-0.11401	0.14258	-0.15140	-0.22945	0.33695	0.47858
22	0.99997	-1.17896	0.00000	-1.17179	-1.16888	-1.17925	-1.16290	-1.17693	-1.16961	-1.18931	-1.14811	-1.21056	-1.03254	0.16880	0.36377	-1.11311	-1.34919	-1.67853	-0.10860	0.14010	-0.21135	-0.24389	0.32633	0.47253
23	0.99997	-1.17860	0.00000	-1.17320	-1.16910	-1.17890	-1.16323	-1.17680	-1.16883	-1.18937	-1.14810	-1.21059	-1.05603	0.13793	0.34932	-1.17576	-1.37133	-1.68560	-0.11401	0.19803	-0.20817	-0.21037	0.34647	0.48403
24	0.99997	-1.18633	0.00000	-1.17516	-1.16954	-1.17862	-1.16583	-1.17566	-1.16809	-1.18962	-1.14806	-1.21070	-1.50370	-0.17439	0.13189	-1.09943	-1.33987	-1.67153	-0.43859	0.09323	-0.30544	-0.84299	-0.12552	0.21134
25	0.99997	-1.17710	0.00000	-1.17634	-1.16993	-1.17834	-1.16568	-1.17570	-1.16720	-1.18971	-1.14802	-1.21087	-0.96665	0.05014	0.31552	-1.01627	-1.30583	-1.65539	-0.03592	0.22838	-0.23328	-0.04243	0.37860	0.50425
26	0.98072	-1.17879	0.01971	-1.17719	-1.17030	-1.17809	-1.16642	-1.17581	-1.16636	-1.18985	-1.14798	-1.21096	-0.75478	0.11980	0.38066	-1.06933	-1.31998	-1.65973	-0.28586	0.18645	-0.23998	-0.17564	0.27892	0.44535
27	0.94996	-1.18009	0.05122	-1.17794	-1.17070	-1.17785	-1.16731	-1.17595	-1.16562	-1.18998	-1.14793	-1.21108	-0.78132	0.04863	0.35085	-1.10995	-1.33266	-1.66286	-0.33006	0.14206	-0.25337	-0.27811	0.20279	0.40030
28	0.99997	-1.17802	0.00000	-1.17837	-1.17107	-1.17763	-1.16782	-1.17623	-1.16482	-1.19010	-1.14789	-1.21119	-0.65096	0.08788	0.39711	-1.13327	-1.33958	-1.66382	-0.27113	0.17568	-0.23997	-0.10419	0.31306	0.46525
29	0.99997	-1.17936	0.00000	-1.17873	-1.17143	-1.17743	-1.16840	-1.17677	-1.16420	-1.19018	-1.14788	-1.21122	-0.56341	0.11638	0.43356	-1.18862	-1.35990	-1.67020	-0.35902	0.11748	-0.21302	-0.21614	0.23362	0.41872

Figure 5.5: Example of normalised CNN input

5.3.2 Network Hyperparameters

For all of the batch normalisation layers, we use the default values set by PyTorch. With reference to Table 4.1 and Table 4.2, we use $N_1 = 400$ and $N_2 = 300$ for both actor and critic MLP networks. In our case, we have only one currency pair GBP/USD, then $N_{action} = 1$.

CNN is more complicated and has more parameters than an ordinary MLP network. With reference to Table 4.3 and Table 4.4, we set $N_T = 30$ (30 consecutive time steps), $N_{F1} = 24$ (no. of features in state $\mathbf{S}$), $N_{F2} = 16$ (channel size for convolutional layer 2), $N_3 = 400$ (node size for L2), $N_4 = 300$ (node size for L3) and $N_{action} = 1$. For the kernel settings and with reference to equation 3.2.6, we set the kernel size $l = 3$, stride $s = 1$ and padding $\rho = 0$.

5.3.3 DDPG Hyperparamters

We list the value of hyperparameters for DDPG+MLP and DDPG-CNN models in Table 5.1. For all of the actor and critic networks for DDPG models, we use ADAM as the network optimiser. We denote the learning rates for actor and critic networks as α and β. The memory buffer size $\mathcal{D}$

in DDPG is set to 1000000 and minibatch with batch size $|\mathcal{B}| = 128$ is used in gradient descent. As shown in Algorithm 2, γ is the discounting factor and η is the optimisation step size.

Model	α	β	γ	Memory size	τ	Episode	η
DDPG+MLP	0.00001	0.0001	0.99	1000000	0.001	2000	60
DDPG+MLP+HER	0.00001	0.0001	0.99	1000000	0.001	2000	100
DDPG+CNN	0.000005	0.00005	0.99	1000000	0.001	1000	100
DDPG+CNN+HER	0.000005	0.00005	0.99	1000000	0.001	1500	100

Table 5.1: Hyper-parameters used for DDPG model training with or without HER

Chapter 6

Results & Discussion

In this chapter we first analyse the training results to investigate how the network architecture within DDPG could have impacts on the trading behaviours of the agent and achieve the optimal trading strategies with the constraints set. Then we test the intra-day trading performance by using the test data from 02-02-2018 to 08-02-2018 (5 trading days).

6.1 Analysis of In-Sample Results After Training

As only up to 72 hours are allowed to train one model, we are not able to run 2000 episodes for both DDPG+CNN models (i.e. with and without HER). For DDPG+CNN without HER we run for 1000 episodes and for 1500 episodes with HER. Both DDPG+MLP models run are trained in 2000 episodes.

6.1.1 Actor-Critic Loss

First we take a look at the loss values for the actor-critic network and the rewards after the training. Figure 6.1 illustrates the progress of the reward, actor and critic losses during the network training. For critic loss, we observe that both DDPG+CNN variations have significant lower values compared to DDPG+MLP. Also, it seems that CNNs help to let the DDPG's critic loss converges earlier than MLP networks. DDPG with CNN networks start to converge at around 20 after 1500 episodes, while the DDPG+MLP with HER starts to converge after 1700 episodes and DDPG+MLP without HER has not reached converge even at episode 2000. One possible reason for early convergence with CNN network in critic loss could be due to more temporal information stored within CNN network (one CNN input contains 30 steps of consecutive states) and therefore the Q-network could detect more overlapping patterns between current state and next state and hence less loss is given according to equation 3.3.11.

Surprisingly, we don't observe any convergences for actor loss for all models after episode 2000. Although no convergence is observed, the rate of growth in actor loss is slowing down and we can deduce that there will be convergence if we have increased the episodes for training. The actor loss for DDPG+MLP with HER is significantly larger compared to other models. The explanation for this is, since the actor loss is derived from policy gradient (see equation 3.3.12), some augmented experiences calculated using HER algorithm may cause unusually big gradient changes during the policy gradient update at each episode during the early stage of training. In reward chart, all models' rewards converge in the range of 2000-5000 (excluding DDPG+CNN with HER). As expected, reward for DDPG+MLP with HER should be higher compare to DDPG+MLP without HER due to the augmented experiences with desired goals and better rewards.

6.1.2 Optimal Cumulative PnL

Now, let's take a look at Figure 6.2, which shows the optimal cumulative PnLs, one of the performance benchmarks that the agent should earn if it starts trading daily from 02-01-2018 to 02-02-2018. The cumulative PnLs for both DDPG+MLP models are almost converging at episode 2000. DDPG+MLP with HER has slightly higher cumulative PnL than same model without HER. At episode 1000, there is not much difference in PnLs for DDPG+CNN with and without HER. It seems like the HER algorithm does not have much impact on DDPG+CNN.

6.1.3 Optimal Strategies After Training

After the training, we would like to investigate if the DDPG models with different networks can generate feasible optimal actions within such an infinitely large search space. For each model, we take the last episode of the training results on trading day **01-02-2018** for analysis. For the DDPG+MLP with and without HER, they are trained until episode 2000. For DDPG+CNN with and without HER, they are trained until 1500th and 1000th episodes respectively. Figure 6.3 shows the optimal actions or strategies generated by the actor-critic networks for each model for trading day on 01-02-2018. Then the agent's trading performance triggered by these trading actions is shown in Figure 6.4.

Analysis from 6.00am to 8.00am

In Figure 6.3, we start the analysis from 6.00am to 10.00am. At this time, the GBP/USD prices are moving from some of the lowest prices to the highest prices of the day, then we observe that the actions from DDPG+MLP without HER are volatile, meaning the model fails to spot the best chance to maximise the profit by buying low and selling high. The other three models manage to spot this opportunities to long as much as possible (with action $a_t = 1$) from 6.00am to 8.00am and then short some GBP/USD positions at around 1.425.

Analysis from 8.00am to 6.00pm

Now we analyse the time in between 10.00am to 6.00pm. During this hours, we notice that the actions from DDPG+MLP with HER model are fluctuating, this means that this model is trying hard to maintain the inventory patterns defined by the inventory model equation 4.3.2 without sacrificing much PnL. This can be shown in Figure 6.4 that the cumulative PnLs produced by DDPG+MLP with HER are on par with DDPG+CNN models while maintaining the inventory shape designed for the agent. It's interesting to see that for DDPG+CNN models, the model without HER has more fluctuated actions than with HER but both of them have similar PnL and position profiles. We could deduce that the strategies for DDPG+CNN with HER has to minimise the long-short transactions due to the inventory constraint defined for the agent, but this also minimises the transaction costs. In contrast, the strategies for DDPG+CNN without HER are to long and short positions at the right time to produce the similar PnL profiles since it does not need to control its inventory. Furthermore, DDPG+MLP without HER model starts to reduce the positions from 4.00pm as it tries to follow the inventory model designed for it. However, we don't see the DDPG+CNN with HER model to do the similar.

Analysis from 6.00pm to 8.00pm

After 6.00pm, it's surprise to observe that DDPG+MLP with HER model does not obey the inventory model and build more long positions. We believe that the model thinks it is more important to have higher PnLs than to follow the inventory rules, as observed from the cumulative

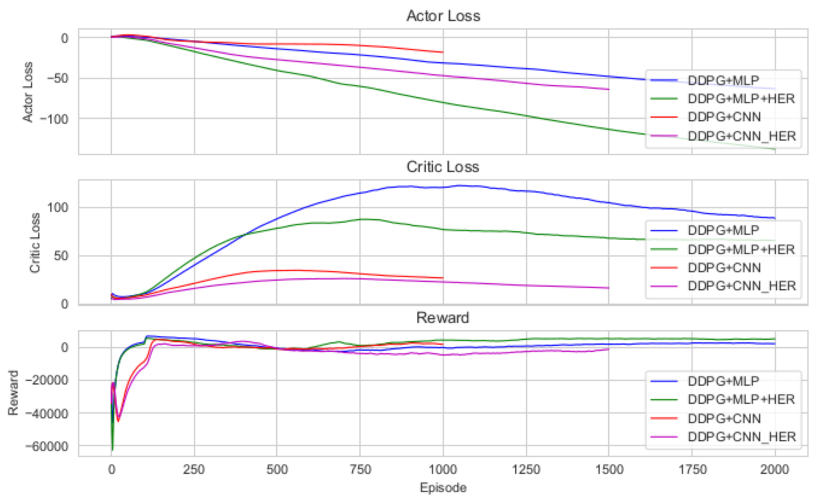


Figure 6.1: The progress of the reward, actor and critic loss during network training

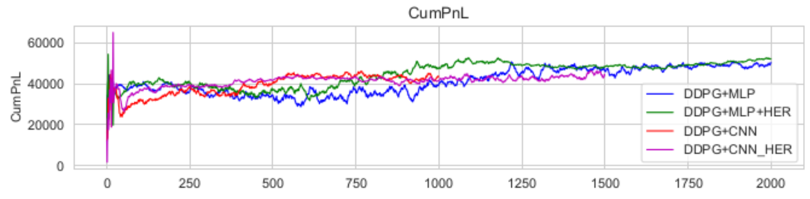


Figure 6.2: The optimal cumulative PnL for each model after training

PnL chart in Figure 6.4. The DDPG+CNN with HER model does not follow the inventory rules too.

Discussion

With the above analysis on different time sessions, without HER, DDPG+CNN model manages to find the optimal actions which provide higher PnLs in lesser episodes (recall that DDPG+CNN models are trained in 1000 episodes only). This is helped by the CNN network's abilities to spot the temporal patterns from the time-series inputs. Now we compare the behaviours of DDPG+MLP and DDPG+CNN with HER, though both models generate reasonable optimal actions, but it's obvious to see that the DDPG+MLP follows the inventory rules more than DDPG+CNN. In other words, combined with the analysis from Actor-Critic Loss section, HER algorithm seems to have lesser impacts to DDPG+CNN as opposed to DDPG+MLP. This could be due to the criteria we defined in Algorithm 5 may not be good enough and thus there are less augmented experiences to be provided to DDPG+CNN model for training.

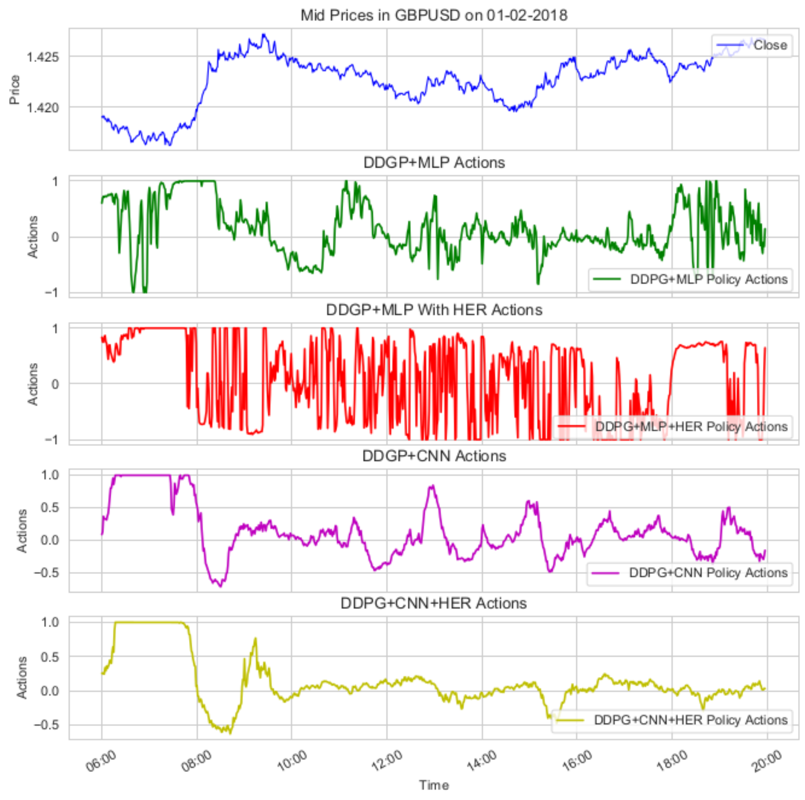


Figure 6.3: The optimal actions for trading day 01-02-2018

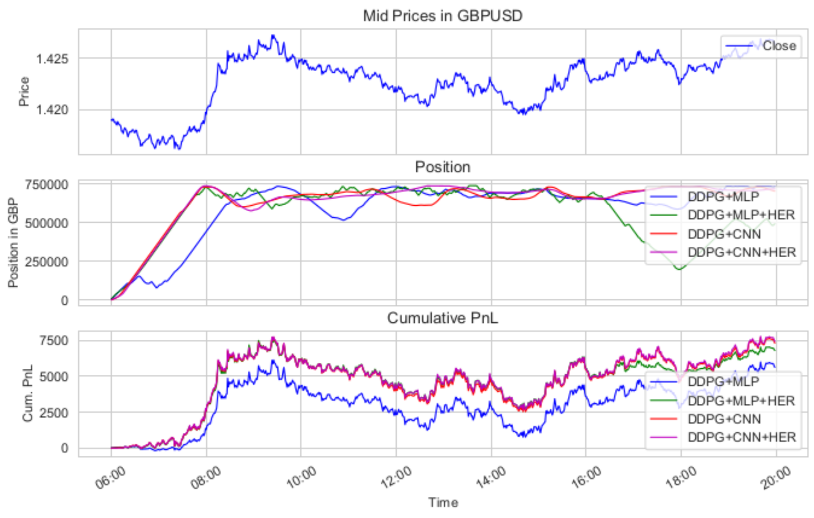


Figure 6.4: The trading behaviour after taking policy actions for trading day 01-02-2018

6.2 Analysis of Out-of-Sample Results

Once all of the models are trained, we use the trained models to run the out-sample intra-day trading from 02-02-2018 to 08-02-2018 (5 working day) to test the trading performance.

6.2.1 Trading Performance Metrics

Four trading metrics are used to evaluate our results: cumulative return (CR), Sharpe ratio (SR), Sortino Ratio (SSR) and Calmer Ratio (CAL). The equations of each metric are denoted as:

$$CR = \frac{(b_T - b_0) * 100}{b_0} \tag{6.2.1}$$

where b_0 and b_T are the initial value and final value of the portfolio assets.

$$SR = \frac{(\mathbb{E}[\mathbf{r}] - \mathbf{r_f})}{\sigma_r} \tag{6.2.2}$$

where $\mathbb{E}[\mathbf{r}]$ is the annualised mean return, $\mathbf{r_f}$ is the risk free or benchmark rate, σ_r is the annualised standard deviation (volatility) of the return.

$$SSR = \frac{(\mathbb{E}[\mathbf{r}] - \mathbf{r_f})}{\sigma_r^-} \tag{6.2.3}$$

where σ_r^- represents the annualised down-side standard deviation.

$$CAL = \frac{\mathbb{E}[\mathbf{r}]}{MDD} \tag{6.2.4}$$

where MDD is the maximum draw-down of the return.

6.2.2 Analysis of Trading Results

Tables 6.1 and 6.2 show the intra-day trading performance by using the out-of-sample data from 02-02-2018 to 08-02-2018. The former table does not include the last minute PnL generated by the end-of-day force close position. Normally when a trader closes the large big positions at the end of the day, this will have big impact on the daily PnL. Hence we use these two tables to compare if there is any impact. If both DDPG+MLP and DDPG+CNN with HER follow the inventory model defined in equation 4.3.2, they will have smaller positions at the end of the day and hence their daily PnL should be less affected.

In Table 6.1, DDPG+CNN with HER are the winners on day 1 and day 2. Surprisingly, DDPG+MLP without HER model is the winner for the rest of the week. Day 3 is the most interesting as both DDPG+MLP models have positive values in all of the metrics. Now we examine the trading results in Table 6.2. As expected, we spot that all performance results are dropped for all of the models. DDPG+MLP with HER is out-performing DDPG+CNN with HER as winner on day 1. DDPG+CNN with HER and DDPG+MLP without HER models stays the same as second day and third day winners respectively. However, on day 4 (07-02-2018), DDPG+MLP with HER is out-performing DDPG+MLP without HER in CR(%) and Sortino Ratio. This means that the inventory control method may help the model to reduce less profit. On day 5, both DDPG+MLP and DDPG+CNN without HER are close with each other, as the former is winning on CR(%) and Sharpe Ratio, whereas the latter is winning on Sortino and Calmer.

Since the unusual performance for DDPG+MLP on 06-02-2018 is spotted, we analyse the trading actions generated by the networks as well as the trading activities that occurred on that day. In Figure 6.5, we detect that there are very high positive correlations between the GBP/USD mid prices and the cumulative PnLs for all of the models. After closer investigation, this is caused the high positions held by all of the agents.

In between 10.00am and 2.00pm, there was big drop in GBP/USD and hence this will also bring down the unrealised PnLs for all of the models (cumulative PnL = cumulative realised PnL + cumulative unrealised PnL). Unluckily for the models with HER, the inventory model (equation 4.3.2) followed by them is totally opposite with the market price patterns, that explains why their cumulative PnL is highly correlated with the GBP/USD market price. As for the other two models without HER, with reference to the actions in Figure 6.6, we discovered that the two models start to build up the positions from 6.00am to 8.00am. At 8.00am, DDPG+CNN without HER has higher positions of £575000 than DDPG+MLP without HER model's £550000. After 8.00am, DDPG+CNN without HER started to offload some positions as it spotted the downward trends but it's not doing enough. When the GBP/USD dropped to 1.385, the lowest of the day, DDPG+CNN without HER still has GBP position around £400000. After 4.00pm, all models start to long for GBP/USD and we see that the higher the position at this time, the more gain they will have until the end of the day. Both DDPG+CNN models have lower positions after 4.00pm and they have lesser long actions compare to DDPG+MLP models. The DDPG+MLP with HER needs to follow the inventroy model to reduce the positions before end of the day. However this isn't the case for DDPG+MLP without HER, this could explain why DDPG+MLP is the winner on 06-02-2018.

6.2.3 Discussion

From the trading performance shown in Table 6.1 and Table 6.2, we observe that the HER algorithm plays some important roles on inventory controls which could have impact on trading performance. We also explain why DDPG+MLP without HER could outperform other models on some occasions. Moreover, we find out that DDPG+MLP models could perform as good as DDPG+CNN in the out-sample tests. We believe that if we could let DDPG+CNN models to be trained with more episodes, then they may perform better.

Date	Model	CR (%)	Sharpe Ratio	Sortino Ratio	Calmer Ratio
02-02-2018	DDPG+MLP	−0.796	−0.856	−1.068	**−0.219**
02-02-2018	DDPG+MLP+HER	−0.743	−0.856	−1.050	−0.227
02-02-2018	DDPG+CNN	−0.794	−0.819	−1.010	−0.224
02-02-2018	DDPG+CNN+HER	**−0.626**	**−0.811**	**−0.984**	−0.227
05-02-2018	DDPG+MLP	−0.582	**−0.704**	−0.902	-0.184
05-02-2018	DDPG+MLP+HER	−0.671	−0.766	−0.946	-0.189
05-02-2018	DDPG+CNN	−0.713	−0.812	−0.995	-0.191
05-02-2018	DDPG+CNN+HER	**−0.553**	−0.709	**−0.889**	**−0.178**
06-02-2018	DDPG+MLP	**0.115**	**0.132**	**0.213**	**0.040**
06-02-2018	DDPG+MLP+HER	0.048	0.050	0.083	0.015
06-02-2018	DDPG+CNN	−0.053	−0.074	−0.115	−0.020
06-02-2018	DDPG+CNN+HER	−0.053	−0.063	−0.097	−0.018
07-02-2018	DDPG+MLP	**−0.399**	**−0.488**	−0.677	**−0.142**
07-02-2018	DDPG+MLP+HER	−0.420	−0.516	**−0.673**	−0.175
07-02-2018	DDPG+CNN	−0.440	−0.559	−0.744	−0.162
07-02-2018	DDPG+CNN+HER	−0.493	−0.610	−0.798	−0.184
08-02-2018	DDPG+MLP	**0.541**	**0.453**	**1.123**	0.391
08-02-2018	DDPG+MLP+HER	0.194	0.126	0.250	0.051
08-02-2018	DDPG+CNN	0.5143	0.417	1.088	**0.473**
08-02-2018	DDPG+CNN+HER	0.188	0.136	0.292	0.069

Table 6.1: Intra-day trading performance using out-of-sample data from 02-02-2018 to 08-02-2018 (5 trading days) EXCLUDING the last minute force close positions

Date	Model	CR (%)	Sharpe Ratio	Sortino Ratio	Calmer Ratio
02-02-2018	DDPG+MLP	−0.863	−0.920	−1.139	−0.237
02-02-2018	DDPG+MLP+HER	−0.743	**−0.856**	**−1.050**	**−0.227**
02-02-2018	DDPG+CNN	−0.892	−0.905	−1.094	−0.251
02-02-2018	DDPG+CNN+HER	**−0.715**	−0.906	−1.071	−0.259
05-02-2018	DDPG+MLP	−0.635	**−0.758**	−0.958	-0.195
05-02-2018	DDPG+MLP+HER	−0.748	−0.842	−1.026	-0.204
05-02-2018	DDPG+CNN	−0.786	−0.877	−1.048	-0.203
05-02-2018	DDPG+CNN+HER	**−0.614**	−0.772	**−0.947**	**−0.193**
06-02-2018	DDPG+MLP	**0.037**	**0.043**	**0.067**	**0.013**
06-02-2018	DDPG+MLP+HER	−0.046	−0.048	−0.077	−0.015
06-02-2018	DDPG+CNN	−0.108	−0.150	−0.229	−0.041
06-02-2018	DDPG+CNN+HER	−0.133	−0.154	−0.231	−0.044
07-02-2018	DDPG+MLP	−0.496	**−0.586**	−0.785	**−0.174**
07-02-2018	DDPG+MLP+HER	**−0.491**	−0.596	**−0.764**	−0.204
07-02-2018	DDPG+CNN	−0.505	−0.635	−0.832	−0.186
07-02-2018	DDPG+CNN+HER	−0.552	−0.676	−0.875	−0.205
08-02-2018	DDPG+MLP	**0.484**	**0.403**	0.976	0.350
08-02-2018	DDPG+MLP+HER	0.124	0.080	0.156	0.032
08-02-2018	DDPG+CNN	0.476	0.386	**0.996**	**0.438**
08-02-2018	DDPG+CNN+HER	0.135	0.097	0.208	0.050

Table 6.2: Intra-day trading performance using out-of-sample data from 02-02-2018 to 08-02-2018 (5 trading days) INCLUDING the last minute force close positions

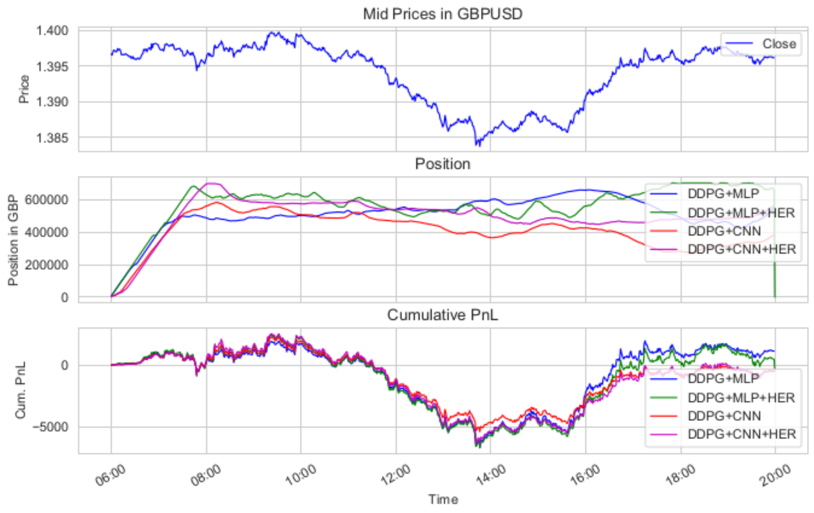


Figure 6.5: The out-of-sample trading behaviour for trading day 06-02-2018

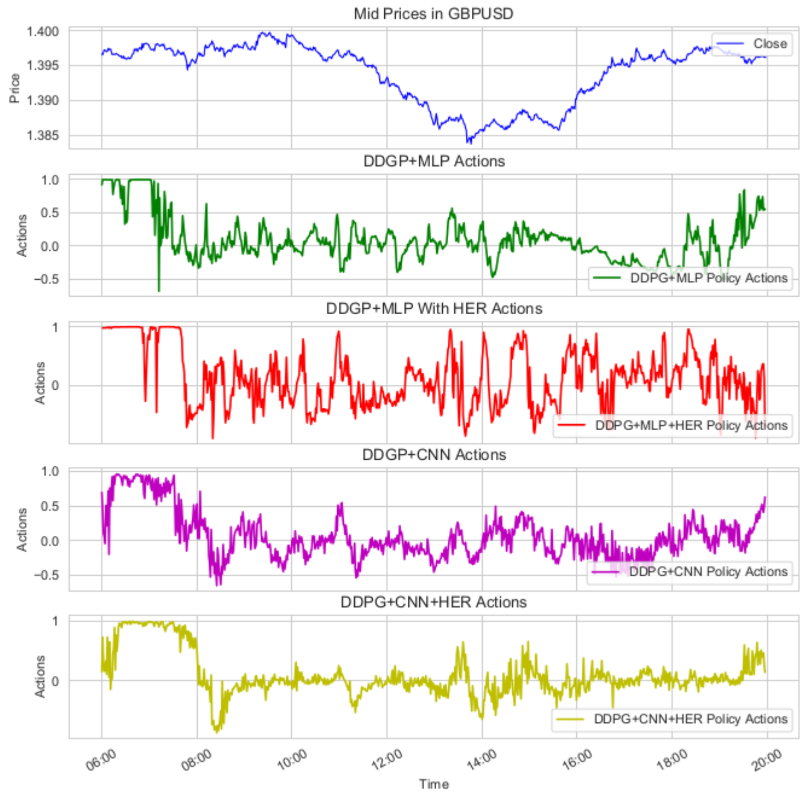


Figure 6.6: The actions for each model for trading day 06-02-2018

Chapter 7

Future Work

Due to time constraints, there are some ideas have not been implemented. In the future, we plan to explore more experiments with the following scenarios.

- **Allow long and short positions in the inventory**: Currently only the inventory with long positions (as shown in Figure 4.1) are tested with DDPG+MLP and DDPG+CNN models. It will be nice to see if the models could perform better if the long and short positions in the inventory are allowed for an agent.
- **More currency pairs**: It will be interesting to find out how would the RL system responds to multiple currency pairs. Could the DDPG model be able to detect the correlations between different pairs of symbols by using different types of networks?
- **Comparison with other RL models**: We could have more experiments by implementing other RL models such as DQN, PPO, TRPO etc. The trading performance of the trading agents, each with different RL models, could be compared.
- **LSTM network in DDPG model**: For our experiment, we have tried MLP and CNN networks within DDPG. How about LSTM network? Could the agent have better trading performance if we have adopted DDPG+LSTM model?
- **Transaction cost control**: We use the HER algorithm for inventory control. Then on top of that, we should be able to use similar techniques to impose the transaction cost control as well to obtain optimal solutions for maximising PnL.
- **Ensemble strategies**: Hongyang et al. [16] proposed to use an ensemble strategy for automated trading. It would be interesting to implement a similar strategy for FX trading.

Chapter 8

Conclusion

The tasks to find out the optimal strategies for FX trading using reinforcement learning, especially with continuous controls, are challenging since it involves multi-disciplinary knowledge such as computing, mathematics, deep learning as well as the trading experiences.

We implemented the DDPG algorithm to work with two different types of deep learning networks MLP and CNN to generate continuous actions for FX trading. In addition, by combining the MLP or CNN with the innovative HER algorithm, we demonstrated that DDPG can be used to provide the continuous optimal strategies for FX intra-day trading with inventory controls. We did some analysis on the training results to compare the optimal actions provided by the all four models and we found out that they can provide the optimal PnLs with the constraints set by us. The abilities to use DDPG to find these optimal strategies are useful in trading world especially in automated trading. Then we compare the trading performance among the four models and to our surprise the DDPG+MLP models perform slightly better than DDPG+CNN models. We believe that if we could improve the training speed for DDPG+CNN models so that it can be trained with more episodes, then they could have better trading performance. Moreover, we demonstrate that the HER algorithm can affect the inventories of DDPG+MLP and DDPG+CNN models and this eventually have some impacts on the models' daily PnLs.

Overall we performed the model implementation well and achieved some satisfactory results. Due to time constraints and hardware limitation, we managed to train the reinforcement learning system with DDPG models by using one month of historical FX market data only. We think that the models could have chance to recognise more market patterns and perform better, if there are more training data to be fed into the system.

Bibliography

[1] John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks, 12(4)*, pages 875–889, 2001.

[2] M. Pakkanen. *Deep Learning Lecture Notes*. Imperial College London, 2020.

[3] Sumit Saha. A comprehensive guide to convolutional neural networks, 2018.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction, 2nd Edition*. The MIT Press, Cambridge, Massachusetts, 2018.

[5] Md. Saiful Islam, Emam Hossain, Abdur Rahman, Mohammad Shahadat Hossain, and Karl Andersson. A review on recent advancements in forex currency prediction. *Algorithms 13(8)*, page 186, 2020.

[6] Victor Talpaert, Ibrahim Sobh, and Ravi Kiran et al. Exploring applications of deep reinforcement learning for real-world autonomous driving systems. 2019.

[7] Abdellatif Alaa, Mhaisen Naram, Chkirbene Zina, Mohamed Amr, Erbad Aiman, and Guizani Mohsen. Reinforcement learning for intelligent healthcare systems: A comprehensive survey. 2021.

[8] Tengteng Zhang and Hongwei Mo. Reinforcement learning for robot research: A comprehensive review and open issues. *International Journal of Advanced Robotic Systems*, pages 1–22, 2021.

[9] Xinghua Lu, Yunsheng Chen, and Ziyue Yuan. A full freedom pose measurement method for industrial robot based on reinforcement learning algorithm. 2021.

[10] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. 2019.

[11] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. *Third IEEE-RAS International Conference on Humanoid Robots*, 2003.

[12] David Silver, Volodymyr Mnih, and Koray Kavukcuoglu et al. Playing atari with deep reinforcement learning. 2013.

[13] Silver D., Huang A., and Maddison C. et al. Mastering the game of go with deep neural networks and tree search. *Nature 529*, page 484–489, 2016.

[14] John Moddy and Matthew Saffell. Reinforcement learning for trading. 1998.

[15] Souradeep Chakraborty. Deep reinforcement learning in financial markets. *Computational Finance (q-fin.CP)*, 2019.

[16] Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. Deep reinforcement learning for automated stock trading: An ensemble strategy. 2020.

[17] Alvaro Cartea, Sebastian Jaimunga, and Leandro Sanchez-Betancourt. Deep reinforcement learning for algorithmic trading. 2021.

[18] Bing Anderson and Shuyun Li. An investigation of the relative strength index. *Banks and Bank Systems*, pages 92–96, 2015.

[19] Mark Leed. Bollinger bands thirty years later. *Statistical Finance (q-fin.ST)*, 2012.

[20] Mansoor Maitah, Petr Prochazka, Michal Cermak, and Karel Sredl. Commodity channel index: Evaluation of trading rule of agricultural commodities. *International Journal of Economics and Financial Issues 6(1)*, pages 176–178, 2016.

[21] Massimo Buscema. Back propagation neural networks. *Substance Use Misuse 33(2)*, pages 233–270, 1998.

[22] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations*, 2017.

[23] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *[Online] http://arxiv.org/abs/1511.08458*, 2015.

[24] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, and etc. Continuous control with deep reinforcement learning. *ICLR 2016*, 2016.

[25] Marcin Andrychowicz, FilipWolski, Alex Ray, and Jonas Schneider et al. Hindsight experience replay. *31st International Conference on Neural Information Processing Systems*, 2018.

[26] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[27] Raffin Antonin, Hill Ashley, Ernestus Maximilian, Gleave Adam, Kanervisto Anssi, and Dormann Noah. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

TONG_TZYY_00402411_THESIS

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49