國立政治大學應用數學系

碩士學位論文

應用深度雙Q網路於股票自動交易系統
Double Deep Q-Network in Automated Stock Trading

指導教授：蔡炎龍　博士

研究生：黃冠棋　撰

中 華 民 國 110 年 12 月

DOI:10.6814/NCCU202200014

致謝

從大學一路到碩士，不知不覺在政大待了七年多，終究到了要畢業的時候。

謝謝炎龍老師在我大學時讓我對深度學習有初步的認識和啟迪，碩士的時候更是不厭其煩的給了我研究的方向和建議，因為老師的教導我才能完成論文和口試；謝謝老大在高微和實變上交會了我很多事情，希望以後還能跟老大一起喝小麥飲料；謝謝張媽在大學時期的照顧，並讓我體會到數學的有趣；謝謝大哥和芷昀助教的幫忙，完成繁瑣的行政程序；謝謝承孝、冠宇和瑄正總在我迷茫的時候給了我很多的幫助和鼓勵；謝謝乾唐在我口試前幫我完成了大大小小的事；謝謝研究生室的大家，和大家的嬉笑打鬧是我在碩士期間最寶貴的回憶；謝謝子瑩的陪伴，我才有動力去完成各個目標；最後謝謝我的家人，總在背後默默的付出，有你們才有今天的我。

DOI:10.6814/NCCU202200014

中文摘要

本篇文章使用了強化學習結合深度學習的技術去訓練自動交易系統，我們分別建立了深度卷積網路和全連接網路去預測動作的 Q 值，並使用 DDQN 的模型去更新我們的動作價值。我們的交易系統每天採用 10 天前的股票資訊，去預測股票的趨勢，並最大化我們的利益。

DDQN 是一種深度強化學習模型，透過建立目標網路和調整誤差函數使得他能夠避免 DQN 的過估計問題，並得到更好的效能，在我們的實驗中，我們得到了一個良好的效果，證明 DDQN 在自動交易系統上是有效的。

關鍵字：深度強化學習、神經網路、Q 學習、深度雙 Q 網路、股票交易

DOI:10.6814/NCCU202200014

Abstract

In this paper, We use the artificial neural network combined with reinforcement learning to train the automated trading system. We construct the CNN model and the fully-connected model to predict the Q-values of the actions and use the algorithm of DDQN to correct the TD error. According to past 10 days data, the system predicts the trend of the stocks and maximize our profit.

DDQN is a deep reinforcement model, which is an improvement of DQN, build the target network and modify loss function to avoid overestimation and get better performance. In our experiment, we get a good result that DDQN is feasible on automated trading systems.

Keywords: Deep Reinforcement Learning, Neural Network, Q-Learning, DDQN, Stocks Trading

DOI:10.6814/NCCU202200014

Contents

DOI:10.6814/NCCU202200014

DOI:10.6814/NCCU202200014

List of Figures

DOI:10.6814/NCCU202200014

Chapter 1

Introduction

In recent years, the development of artificial intelligence, whether in image classification, semantic analysis, recommendation systems, etc., has achieved impressive results. In many fields, AI has even surpassed human performance. For instance, in 2017, AlphaGo [10] defeated the world Go champion Ke Jie, in which deep reinforcement learning played a very crucial role.

Reinforcement learning is a kind of machine learning, which expects an agent to learn the optimal behavior strategy by taking actions in an environment. However, in the actual work, conventional reinforcement learning is unable to cope with complex and large amounts of data. To solve these problems, scientist combined reinforcement learning with deep learning, solved the restriction of tabular form and improved the generalization ability of the model. In 2013, DeepMind combined Q-learning [14] with CNNs [5], which is called DQN [8], exceeded human experts in three games of Atari 2600. Nowadays, deep reinforcement learning has been applied with big success in various fields.

This paper adopted double deep Q-learning [13] to establish an auto-trading system. We chose underlying stocks of Taiwan 50, which contains top fifty companies in Taiwan, as data. To evaluate the capacity of our model, we compared our profit to other trading strategies in practical.

DOI:10.6814/NCCU202200014

Chapter 2

Deep Learning

In traditional machine learning, we need to divide the problem into several parts and artificially extract features from the data, and then use algorithms to make the machine learn. However, designing a large number of complex feature engineering is a daunting task when dealing with cluttered and unstructured data [2]. Deep learning is a branch of machine learning with the structure of artificial neural network and deep layers. It implements an end-to-end [9] approach where the input is the raw data and the output is the final result, reducing a lot of human intervention.

The basic idea of deep learning is to find the best function based on the training data, by which the answer can be obtained. For example, If you want to know whether a stock will go up or down on a certain day in the future,

$$f(\text{“ Date ”}) = \text{“ The close price of the stock. ”}$$

In practice, both input and output of the function must be vectors. The neural network operates a series of weighted calculations from the input vector, and modifies itself by the difference of output values and real answers. Through this progress, we can find the suitable function solving our problems.

In conclusion, we divide deep learning into three steps. First of all, construct the model. The function set would be determined by this structure of network. Next, we choose the loss function which can identify the function is good or not. Lastly, train the model according to the training data. Thus, the model can be modified itself and get the best function from its function set.

DOI:10.6814/NCCU202200014

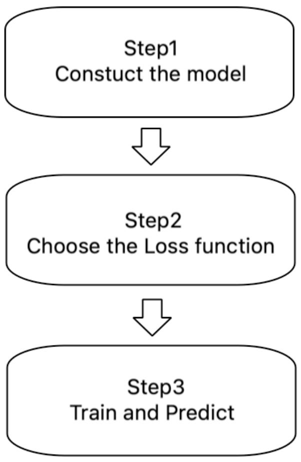


Figure 2.1: Three Steps of Deep Learning

2.1 Neurons and Neural Networks

Artificial neural network [15] is the core structure of deep learning. its name and concept inspired by the human brain, and mimics the way biological neurons send signals to each other. This section will introduce how do neural networks work.

A neuron in neural network is a simple function that usually consists of linear and non-linear components. As shown in Figure 2.2, $x_1,\ x_2,\ ...,\ x_n$ are the inputs and $w_1,\ w_2,\ ...,\ w_n$ are the corresponding weights. We multiply them one by one and add up total, and plus a bias b. Lastly, an activation function σ, which is the non-linear part, sends out the output $h = \sigma(\sum_{i=1}^{n} w_i x_i + b)$ for the neuron.

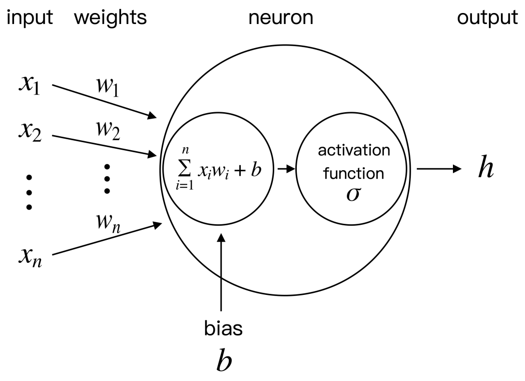


Figure 2.2: A Neuron in Neural Networks

DOI:10.6814/NCCU202200014

As the following example, we can clearly understand the process of the operation. Suppose $x_1,\ x_2,\ x_3$ are $3,\ (-1),\ 1$; $w_1,\ w_2,\ w_3$ are $2,\ (-3),\ (-2)$; and b is 2. Let the activation function σ be ReLU which will output directly if input is positive number, otherwise, it will output zero. First, we multiply each x_i and w_i where i is $1,\ 2,\ 3$, and add up. We get $\sum_{i=1}^{n} w_i x_i = 3 \times 2 + (-1) \times (-3) + 1 \times (-2) = 7$. Next, we plus the bias $b = 2$, then we get $7 + 2 = 9$. Note that the weights and the bias would be modified while training the model. Finally, send the number into the activation function $\sigma(9) = 9$. Hence, the output $h = 9$.

We can build a neural network by connecting individual neurons. As below Figure 2.3, it is a simple neural network. We can divide it into three parts. The first layer which is on the left side called input layer, the last layer which is on the right side called output layer, and the rest of layers are called hidden layers. Each neuron receives the output from all neurons of the previous layers as input, and its own output sends to all neurons of next layer. This process keeps one direction from input layer to output layer. We call it fully connected feed-forward neural network.

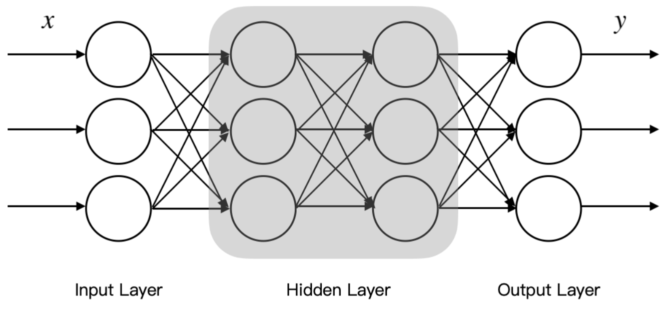


Figure 2.3: Neural Network

When the neural network was constructed, the function set is determined. Hence, if the structure of the model is bad, the functions in this function set would be not work.

2.2 Activation Function

In reality, most of the problems can not be predicted linearly. We need some non-linear functions, called activation function [6], to make our neurons have non-linear relationship. There are some common activation functions as following.

DOI:10.6814/NCCU202200014

1. Sigmoid Function

   Equation:

$$f(x) = \frac{1}{1+e^{-x}}$$

   Range: $(0, 1)$

   Graph:

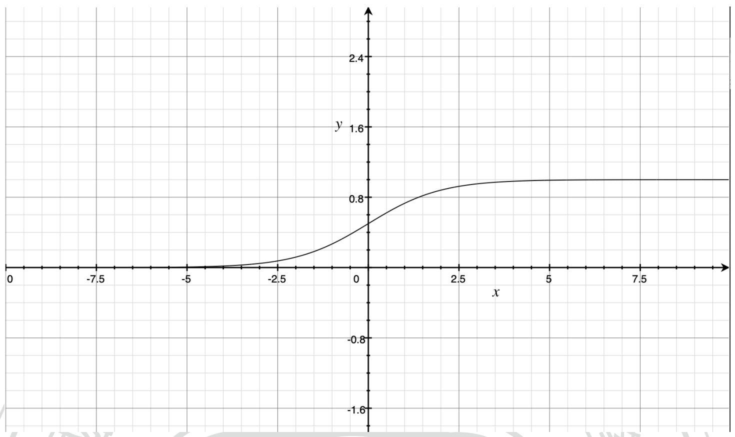


Figure 2.4: Sigmoid Function

2. Hyperbolic tangent (tanh)

   Equation:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

   Range: $(-1, 1)$

   Graph:

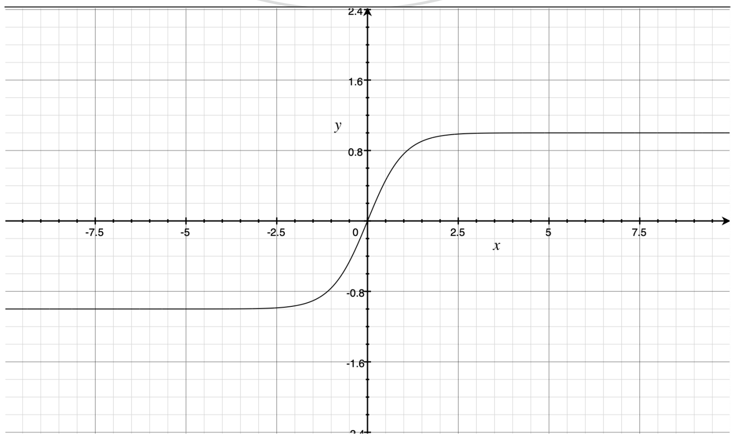


Figure 2.5: Hyperbolic Tangent (tanh)

DOI:10.6814/NCCU202200014

3. Rectified linear unit (ReLU)

   Equation:

$$f(x) = \max(x, 0)$$

   Range: $[0, \infty)$

   Graph:

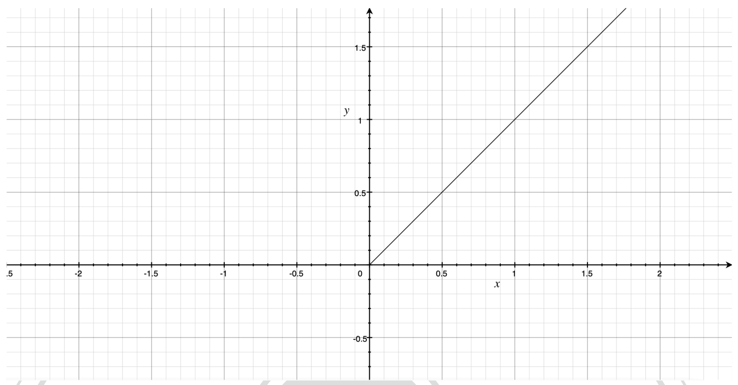


Figure 2.6: Rectified Linear Units (ReLU)

2.3 Loss Function

After deciding the structure of the model, there are lots of undetermined coefficients called parameters, consisting of weights and bias, denoted by θ. Then the network is a function set based on θ, denoted by $\{f(x|\theta)\}$ where x is the input. As mentioned earlier, the purpose of deep learning is to find an optimal function. To find the optimal function, we would like to find out the best parameters θ^*. Hence, we need a judgment criterion to rate our functions.

Here, we define y represents the actual answer, and $f(x|\theta)$ represents the output value. The judgment criterion, called loss function $L(\theta)$, estimates the distance between y and $f(x|\theta)$ under the parameter θ. Thus, through the loss function, we can evaluate the network with the θ. Then we can obtain θ^* as long as we minimize the loss function. As below, we show some simple loss functions.

1. Mean Absolute Error (MAE)

   Equation:

$$\text{MAE}(\theta) = \frac{1}{k} \sum_{i=1}^{k} ||y_i - f(x_i|\theta)||$$

DOI:10.6814/NCCU202200014

where k is the total number of data

2. Mean squared error (MSE)

   Equation:

$$\text{MSE}(\theta) = \frac{1}{2k}\sum_{i=1}^{k} ||y_i - f(x_i|\theta)||^2$$

   where k is the total number of data

3. Binary cross-entropy(H)

   Equation:

$$\text{H}(\theta) = -\frac{1}{k}\sum_{i=1}^{k}[y_i \log(f(x_i|\theta)) + (1-y_i)\log(f(x_i|\theta))]$$

   where k is the total number of data, and each $y_i = 0$ or 1

2.4 Gradient Descent Method

Gradient Descent Method is an approach to minimize the loss function. The core concept is taking steps in the opposite direction of the gradient of the function. We take a look at below Figure 2.7 as simple example. This is a graph for $L(\theta) = \theta^2 - \theta + 1$. Suppose the initial point located at $(a, L(a))$, and we hope to move it to the minimum location of $L(\theta)$. First, we calculate $L'(a)$, and then multiply it by the learning rate η. Lastly, move $(a, L(a))$ to the new spot $(a - \eta L'(a), L(a - \eta L'(a)))$. η is usually a small positive number, and we'll introduce it later. Since $L'(a) > 0$ and $\eta > 0$, we shift the initial point left to reduce $L(\theta)$. Repeating these steps, until the gradient of the function is zero. Likewise, the operation at $(b, L(b))$ is the same.

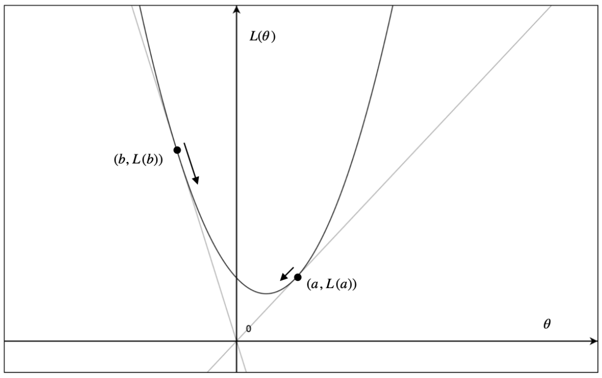


Figure 2.7: $L(\theta) = \theta^2 - \theta + 1$

DOI:10.6814/NCCU202200014

If there are lots of parameters of the function, we can calculate the partial derivatives of each parameters, and do the same operations on each one. Given $\theta = \{w_1, w_2, ..., w_m, b_1, b_2, ..., b_n\}$ is the parameters of the network with initial random values. The new θ will be as below. Actually, the gradient decent calculated easily by chain rule from the last layer to the beginning. This approach is applied in most of the models, called back propagation [3].

$$\theta^{\text{new}} = \begin{bmatrix} {w_1}^{\text{new}} \\ {w_2}^{\text{new}} \\ \vdots \\ {w_m}^{\text{new}} \\ {b_1}^{\text{new}} \\ {b_2}^{\text{new}} \\ \vdots \\ {b_n}^{\text{new}} \end{bmatrix} = \begin{bmatrix} w_1 - \eta\frac{\partial L}{\partial w_1} \\ w_2 - \eta\frac{\partial L}{\partial w_2} \\ \vdots \\ w_m - \eta\frac{\partial L}{\partial w_m} \\ b_1 - \eta\frac{\partial L}{\partial b_1} \\ b_2 - \eta\frac{\partial L}{\partial b_2} \\ \vdots \\ b_n - \eta\frac{\partial L}{\partial b_n} \end{bmatrix}$$

The learning rate η is a hyperparameter set by human. It controls the speed of the movement. If η is too big, the point may cross the lowest spot and getting worse. On the other hand, if η is too small, we'll waste lots of time on the moves and even get stuck in the local minimum. There are many optimizers can control η automatically. The most well-known are Adagrad, RMSprop or Adam.

DOI:10.6814/NCCU202200014

Chapter 3

Convolutional Neural Network (CNN)

Before the development of CNN, there were two problems with image processing in neural networks: first, the image data was too large, making the training too inefficient; the other was that the information of images easily lost through the digitization process, and the original features could not be retained. CNN is a method good at dimension reduction, which could reduce a large number of parameters while retaining the original features.

The original CNN is composed of three parts: convolutional layer, pooling layer, and fully-connected layer. The convolutional layer is used to extract local features from the image; the pooling layer is used to significantly reduce the numbers of parameters; and the fully-connected layer is used to output the required result.

1. Convolutional Layer

   In convolutional layer, we decide some convolutions (or called filters) and use them to slide over the image. This process helps us to extract some features from the image. There's an example as Figure 3.1, we use a 3×3 convolution to slide over the 5×5 image to get a 3×3 feature map.

DOI:10.6814/NCCU202200014

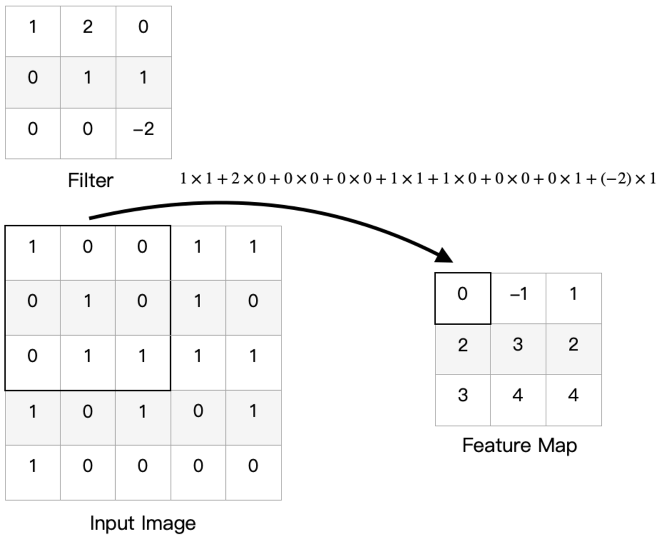


Figure 3.1: Convolutional Layer

2. Pooling Layer

   Even after doing the convolution, the image is still large. Hence, we use the pooling layer to reduce dimension. The most common pooling method is maximum pooling. As Figure 3.2, we slide over the feature map and find the maximum value to get the pooled feature map. Pooling layers can reduce the dimensionality of data more efficient than convolutional layers, which can not only greatly reduce the amount of operations, but also effectively avoid overfitting.

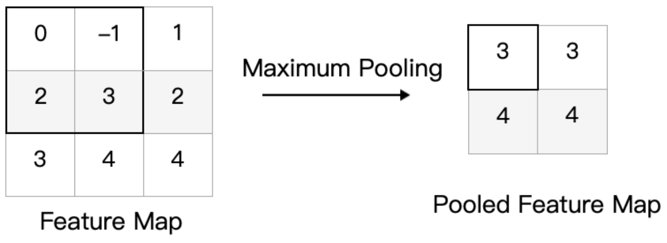


Figure 3.2: Pooling Layer

3. Fully-Connected Layer

DOI:10.6814/NCCU202200014

The data processed by the convolutional layers and the pooling layers are finally flattened and fed to the fully-connected layer to obtain the desired result.

Typically, our CNN network has several convolutional and pooling layers interleaved until the parameter space is small enough. Finally, we flatten the vectors and feed them to the fully-connected layer to output the result. Compared to other image classification algorithms, CNNs use relatively little preprocessing and work well in various field.

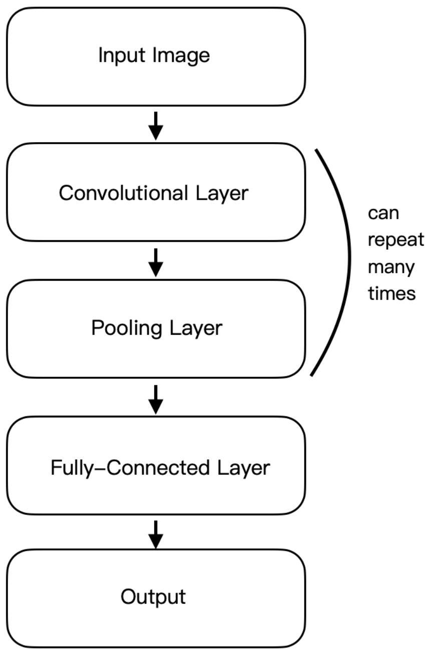


Figure 3.3: The Process of CNN

DOI:10.6814/NCCU202200014

Chapter 4

Reinforcement Learning

Machine learning can be divided into three main categories: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning deals with labeled training data. The machine learns from these known output data and expects to get the correct answer when it comes to unseen input data. As the concept we mentioned in previous chapter, it automatically create a function that maps inputs to outputs.

Unsupervised learning learns some hidden structures from a large amount of unlabeled data. A common example of this approach is data clustering. In reality, we can cluster the consumer groups and design corresponding products for different group.

Reinforcement learning is the third method between supervised learning and unsupervised learning. It does not have labeled data, but not without relevant information. The main purpose of reinforcement learning is to solve the problem of sequential decision making. In more detail, reinforcement learning expects the agent to learn itself over time in an environment and make the optimal behavioral strategies.

4.1 Introduction

Before we dive into the details, let's give an example to get a clearer picture. Assuming a grid world with a number of treasures and traps scattered throughout. A robot decides which direction to go by observing. Our goal is to collect as much treasure as possible while avoiding traps. Reinforcement learning is an approach for the robot to learn the behavioral strategies on their own. We will introduce terms that reinforcement learning settings through the above

DOI:10.6814/NCCU202200014

examples.

1. Agent

   Agent is the intelligence that explore and learn in the environment.

2. Environment

   Environment is the space in which the agent is located.

3. Reward

   Reward is the value that an agent obtains from the environment from time to time. The purpose of the reward is to provide an agent with information about its success or not. The values can be some integers or some vectors, as we decide.

4. Observation

   Observation refers to information about the environment that the agent can perceive in addition to the reward. In general, the agent can only observe a limited domain of the environment, just like humans.

5. Action

   The agent interacts with the environment via actions. The actions can be classified in two types. One type is discrete and can be described as whether to do a certain action. The other is continuous, which is generally represented by vectors.

As above example, the agent is the robot which is located in the grid world. And we call the entire grid world the environment. The agent can observe information within three grids. Through this observation, the agent takes an action to move up. After moving to the new spot, the agent gets a treasure and a new observation. In the following figure, we can clearly understand the interaction between the agent and the environment in reinforcement learning.

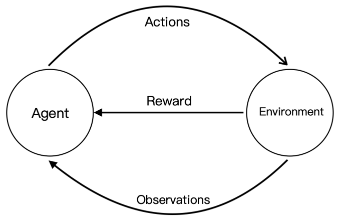


Figure 4.1: The interaction between agent and enviorment

DOI:10.6814/NCCU202200014

4.2 Markov Decision Processes

In order to deal with a series of interactions between agent and environment, we construct mathematical models for reinforcement learning through Markov Decision Processes to simplify the problem.

1. Markov Process

   By observing the environment, the agent would get a state. All possible states in the environment form a finite state space S. Through a series of observations, we get a series of states from the state space. The state change is called Markov Process $(S, P_{ss'})$ if it satisfies the condition:

   $$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t].$$

   This means the transition probability from the pervious state S_t to the next state S_{t+1} does not change at any time t. With Markov Process, we have the transition matrix $P_{ss'}$ to describe the transition probabilities between the states s and s'. For example, assume that the weather changes with sunny and rainy days. If it is sunny today, tomorrow will be 40% sunny and 60% rainy. On the other hand, if today is rainy, tomorrow will be 20% sunny and 80% rainy. We can organize these transitions into the following table.

	sunny	rainy
sunny	0.4	0.6
rainy	0.2	0.8

   In practice, it it almost impossible to know the exact transition probabilities. We can only observe the environment to get a series of the states, which is called episode {sunny, rainy, sunny, sunny, rainy, sunny, rainy, rainy}. When we have more and more observations, we can estimate the transition probabilities.

2. Markov Reward Process

   We extend Markov Process to Markov Reward Process $(S, P_{ss'}, R, \gamma)$ by added reward R and discount factor γ. In MRP, the agent will receive a reward whenever the states

DOI:10.6814/NCCU202200014

transferred. For every episode $\{s_1, r_1, s_2, r_2, ..., s_T\}$, the return at time t is defined as:

$$G_t = r_{t+1} + \gamma r_{t+2} + ... = \sum_{k=0}^{T-t} \gamma^k r_{t+k+1}.$$

The discount factor represents the foresight of the agent. When $\gamma = 1$, the return G_t will be the sum of the sequential rewards, which means that the future is as important as the present; When $\gamma = 0$, G_t will be the present reward, which means not caring about the future at all.

3. Markov Decision Process

We add the action space A in Markov Decision Process $(S, A, P^a_{ss'}, R, \gamma)$. To describe the action decisions of the agent, we define the policy π:

$$\pi(a|s) = P[A_t = a|S_t = s].$$

The policy $\pi(a|s)$ describes the probabilities of the actions based on the states.

Lastly, we define the state value function $V_\pi(s)$ and the state-action value function $Q_\pi(s, a)$. $V_\pi(s)$ is the expected return with policy π at state s, and $Q_\pi(s, a)$ is the expected return with policy π at state s and an action a.

$$V_\pi(s) = E_\pi[G_t|S_t = s]$$

$$Q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$$

Given a policy π to an agent, it can interact with environment and get the episode $\{s_1, a_1, r_1, s_2, a_2, r_2, ..., s_T\}$. Although we can use G_t to represent the value of state s_t, it is not subjective because the episode is specific. Hence, we prefer to use $V_\pi(s)$ or $Q_\pi(s, a)$ which is the expected value with many episodes.

In dynamic programming, we iteratively calculate the value function by the Bellman equation:

$$V_\pi(s) = \sum_{a\in A} \pi(a|s)(\sum_{s'\in S} P^a_{ss'}(r^a_{ss'} + \gamma V_\pi(s'))).$$

DOI:10.6814/NCCU202200014

After we finish all the paths, we can get the value function of all the states by recording the transition probabilities and rewards of all Markov chain. This way of updating values is called full-width backup. However, when the environment is too complex to construct whole transition probabilities table, the dynamic programming won't work. In next section, we'll introduce the methods to deal with this problem.

4.3 Monte Carlo Method and Temporal Difference

Monte Carlo Method uses sample backup. At the end of each episode, Monte Carlo Method calculates the value of each states and makes a weighted average. As the episodes become more and more, our estimation becomes closer to the actual expected value. In practice, we use the following formula to update the estimates.

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(G_t - Q(s_t, a_t))$$

α is a step-size parameter, influence the degree of smoothing of estimation updates.

The adaptability to the environment is the advantage of Monte Carlo Method. We only rely on the interactions between the agent and the environment. However, in the previous experimental results, the estimation of the state values would deviate significantly from the actual value. In addition, the estimates cannot be updated until the end of the episode.

Instead of estimating at the end of each episodes, Temporal Difference updates the estimation at each step. At each time $t+1$, TD method updates the estimations immediately with following formula.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1} - Q(s_t, a_t))]$$

Both Monte Carlo Method and Temporal Difference don't need to construct the whole transition model for environment. They update the values from the experience of the episodes. This concept is called model-free. In the face of complex environments, a model-free approach will be more effective than a model-based one.

DOI:10.6814/NCCU202200014

4.4 Q-Learning

Q-Learning is one kind of TD method. We first initialize a Q-table to record the values of the state-action pairs, and then updates the Q-table for each step of each episode according to the following formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \text{max}_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

It considers the maximum valuation among all the next state-action value function $Q(s_{t+1}, a_{t+1})$ to update $Q(s_t, a_t)$. The next state-action Q values are calculated without choosing the next action. For a clear understanding of Q-Learning, the pseudocode algorithm is shown as below:

Algorithm 1 Q-Learning Algorithm

```
Initialize $Q(s, a)$ arbitrarily
for each episode do
    Initialize state $s$
    for each step of the episode do
        Choose an action $a_t$ according to $Q$ function and state $s_t$
        Take an action $a_t$; observe reward $r_{t+1}$ and next state $s_{t+1}$
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \text{max}_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
        $s_t \leftarrow s_{t+1}$
    end for
end for
```

DOI:10.6814/NCCU202200014

Chapter 5

Deep Reinforcement Learning

Practical tasks often have high-dimensional inputs, which are difficult to be processed with traditional reinforcement learning. There are two ways to solve this problem. One is to use manual feature engineering to reduce the dimensionality of the input data, which is usually a tough project. The other approach is to use deep learning to take advantage of computation to do the work automatically.

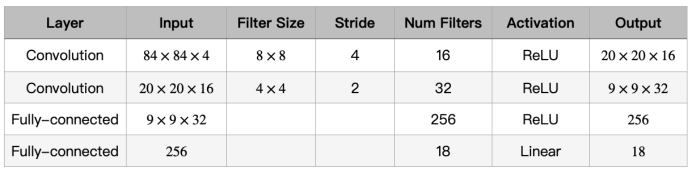
5.1 Deep Q-Learning Network (DQN)

In traditional Q-Learning, we update Q-table iterative according to each step of episodes. However, when the state and action space are too large, Q-table is difficult to be recorded and established. In 2013, DeepMind published that Deep Q-learning Network called DQN. Actually it is a combination of Q-Learning and neural network, turning Q-table into Q-Network. Solving the curse of dimensionality by representing Q as a function.

$$Q(s, a; \theta) \approx Q(s, a)$$

θ is the parameter of the model. The structure of the model is the connection of two convolutional layers and two fully connected layers. The input is four frames of 84×84 images and the output is 18 Q-value of the actions.

DOI:10.6814/NCCU202200014



Layer	Input	Filter Size	Stride	Num Filters	Activation	Output
Convolution	$84 \times 84 \times 4$	8×8	4	16	ReLU	$20 \times 20 \times 16$
Convolution	$20 \times 20 \times 16$	4×4	2	32	ReLU	$9 \times 9 \times 32$
Fully-connected	$9 \times 9 \times 32$			256	ReLU	256
Fully-connected	256			18	Linear	18

Figure 5.1: The composed of DQN

After the network is established, the loss function is set as $L(\theta) = E[(r+\gamma max_{a'} Q(s', a'; \theta) - Q(s, a; \theta))]$ with stochastic gradient decent. Using ϵ-greedy method to generate the transitions (s_t, a_t, r_t, s_{t+1}), i.e., do random action with probability ϵ and do $\text{argmax}_a Q(s_t, a; \theta)$ at time t. And store the transitions into "Replay Memory" as training data to train the model.

Replay memory is designed to eliminate correlations between data. It builds a store and collects the transitions. When we want to update the Q-values, we sample some transitions for calculation. It has the advantage of helping the network to converge.

Algorithm 2 Deep Q-learning

```
Initialize replay memory $D$ to capacity $N$
initialize state-action value function $Q$ with random weights
**for** episode = 1, $M$ **do**
    Initialize sequence $s_1 = x_1$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    **for** $t = 1, T$ **do**
        With probability $\epsilon$ select a random action $a_t$
        Otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and process $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi(t), a_t, r_t, \phi_{t+1})$ in $D$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j, & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \text{max}_{a'} Q(\phi_{j+1}, a'; \theta), & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
    **end for**
**end for**
```

We structurally divide this pseudocode algorithm into three layers.

1. First Layer

   Initialize replay memory D with capacity N to store the transitions, and initialize network with parameter θ. Then, do M episodes.

DOI:10.6814/NCCU202200014

2. Second Layer

   For each episode, initialize the sequence and do the pre-processing of input. Then, do T steps for the episode.

3. Third Layer

   Choose an action a_t with ϵ-greedy method, and store the transition into replay memory. Lastly, Sample some transitions from replay memory, and optimize the loss function with gradient descend.

There are some improvements of DQN as following:

1. Nature DQN

   In 2015, DeepMind improves DQN with "Target Network", named "Nature DQN". Target network is used to provide target value y_i for main network to learn. The reason is that in 2013 DQN, the target value y_i is provided by main network, which leads to a constant change of labels. If we use the constantly changing labels to adjust our network, the value estimation could easily lose control. Target network was so effective in smoothing convergence of the network that it was used in all later models. The loss function change into the following formula:

$$L(\theta) = (r + \gamma \text{max}_{a'} Q(s', a'; \theta^-)) - Q(s, a; \theta),$$

   where θ^- is the parameter of target network and θ is the parameter of main network. And we adjust the parameter of target network equal to the parameter of main network every C steps.

2. Double DQN (DDQN)

   Double Deep Q-Learning is designed for solving overestimation problems. In DQN, we use the maximum value of next state-action Q-value to evaluate the state values. However, the objective state value should be the expected value of all next state-action Q-value in this state. Therefore, the problem of overestimation occurs. In order to deal with this problem, we use different networks to select action and evaluate the Q-value. Then the

DOI:10.6814/NCCU202200014

loss function changes to the following form:

$$L(\theta) = (r + \gamma Q(s', \text{argmax}_{a'} Q(s', a', \theta); \theta^-)) - Q(s, a; \theta))^2.$$

3. Dueling DQN

   The dueling network separate Q-network into two estimators: one for the state value function V and the other for the advantage function A. It considers that a value function Q is contributed by the value of the state itself and the action a under the state, i.e., $Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a)$. The V function is independent of the action and represents the value of the state; the advantage function A is related to the action and represents how good the action is in the state. The dueling network separately calculates the V function and the A function, and combines them into a single Q function at the final layer.

4. Priorized Experience Replay

   Priorized Experience Replay is a method that solves the problem of inefficient convergence of Q-Learning. According to the formula, $r + \gamma max_{a'} Q(s', a) - Q(s, a)$, the transitions with larger TD error will be selected from the memory buffer for updating priority.

5.2 Policy Gradient

In DQN, we use an indirect method to find the optimal policy with the Q-value, which is called value-base. Instead of a strategy, we obtain a Q-table that compare $Q(s, a)$ each time according to the state s to get a higher valuation of a, and then connect the pieces to form a complete strategy. Now, we present a direct method, called Policy Gradient, for network learning to find the optimal policy.

In Policy Gradient, we want to get a best policy directly. The policy with parameter θ is defined as the following form:

$$a = \pi(s|\theta) \text{ or } a = \mu(s|\theta),$$

where π is the stochastic policy output the distribution of action, and μ is the deterministic policy output one action.

DOI:10.6814/NCCU202200014

To evaluate the goodness of a policy, we compare the total rewards obtained. Assume the policy π guides the agent go through the episode $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_t, a_t, r_t)$. We define the objective performance function to evaluate the policy:

$$J(\theta) = E[r_0 + r_1 + r_2 + ... + r_t|\pi_\theta].$$

Now, if we can calculate the gradient of this function, we can do gradient ascent and find the maximum value, which will optimize our network. In 1999, the paper [12] publish that the stochastic policy gradient is:

$$\nabla J_\theta(\pi_\theta) = E_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \text{log}\pi_\theta(a|s)Q^\pi(s,a)].$$

And in 2014, the paper [11] demonstrated that deterministic policy gradient is:

$$\nabla J_\theta(\mu_\theta) = E_{s\sim\rho^\mu}[\nabla_\theta\mu_\theta(s)\nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}].$$

ρ is the distribution of the state under the policy.

In 2015, Google published a new structure of neural network called DDPG [7]. It combines the skills of DQN and DPG and adopts Actor-Critic method. In Actor-Critic method, the model is divided into two parts: value-based and policy-based networks. The policy network is called actor, which outputs the action. And the value network is called critic, which is used to evaluate the goodness of the action chosen by actor, and outputs the TD error to update the actor and critic networks.

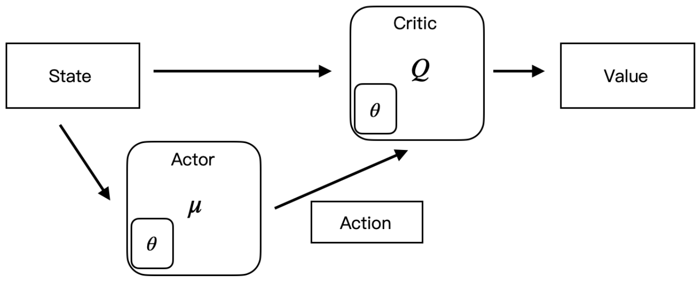


Figure 5.2: The Architecture of DDPG

The main concept of DDPG:

DOI:10.6814/NCCU202200014

1. DPG

   Deterministic policy gradient theorem supports the mathematical theory for DDPG.

2. DQN

   DDPG also uses replay memory and target network to train the network stably.

3. Behavior Policy

   In the training process, we insert the random noise into the decision of the action. The decision of the action is changed from a deterministic process to a stochastic process, which allows the agent to better explore environment.

$$a_t = \mu(s_t|\theta^\mu) + N_t$$

DOI:10.6814/NCCU202200014

Chapter 6

Automated Trading System

In this paper, we will use Double Deep Q-Learning with CNN model [1] [4] and fully-connected model to allow the agent to buy stocks automatically. We expect the agent can learn the trend of the stocks and maximize the profit.

6.1 Dataset Preparation

We chose the stocks in Taiwan50, which contains the top 50 largest and most representative stocks in Taiwan, as our data. We randomly select 40 stocks in Taiwan50 from 2015 to 2019 as training datasets and 2020 as validation datasets. And the rest 10 stocks in 2020 are used as testing datasets. There are five feature of each data: Open, High, Low, Close, Volume. Open is the opening price at which a security first trades when an exchange opens for a day; High is the highest price for the stocks in a trading day; Close is the closing price that is the last price at which a security traded during the regular trading day; Volume is the number of shares of a security traded during a given period of time. In order to eliminate the differences of scale range between the stocks, we use Feature Scaling method to normalize our data.

$$\frac{x_i - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

X_{max} and X_{min} are maximum value and minimum value in the set. Feature Scaling transform the values in the set into $[0, 1]$. Normalization allows the values of data at different scales to be adjusted to a nominally common scale that is meaningful for comparison.

DOI:10.6814/NCCU202200014

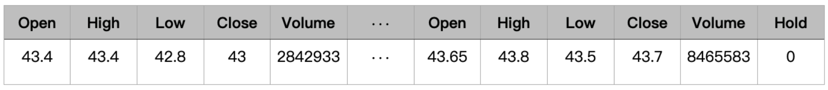
6.2 Trading System Settlement

1. Environment

   The environment represents the entire financial market. We do an action in the environment and it will return a reward. In this paper, the environment is the 50 stocks of Taiwan50 and the agent will interact with the environment maximize the profit.

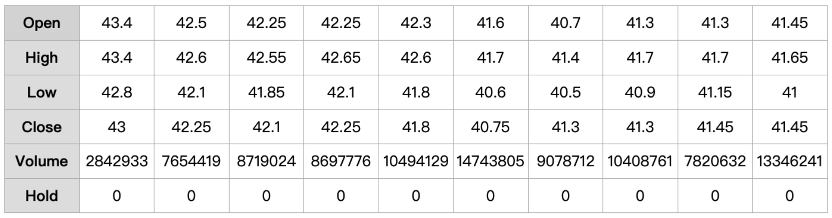
2. State

   The states are observations from the environment that are input to the neural network. We set each state contain pass 10 days of Open, High, Low, Close, Volume and the number of stocks held at that time. In fully-connected mode, our states are size of 51 $\times$ 1 matrix; and in CNN mode, the states are size of 6 $\times$ 10 matrix. The states are like below figures.



Open	High	Low	Close	Volume	···	Open	High	Low	Close	Volume	Hold
43.4	43.4	42.8	43	2842933	···	43.65	43.8	43.5	43.7	8465583	0

Figure 6.1: State Input to Fully-Connected Model



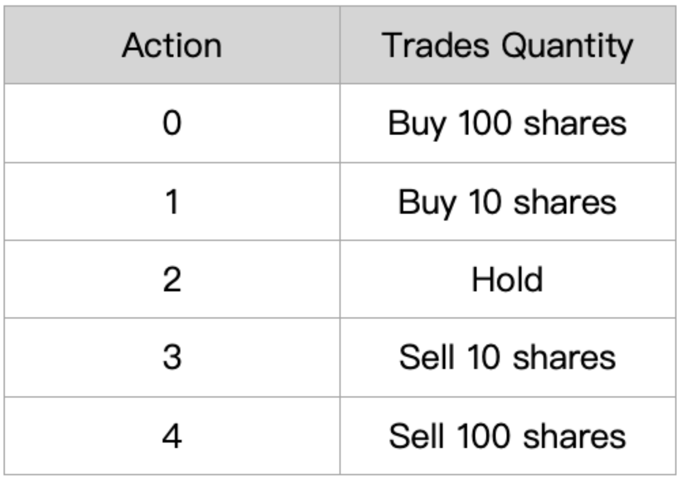
Open	43.4	42.5	42.25	42.25	42.3	41.6	40.7	41.3	41.3	41.45
High	43.4	42.6	42.55	42.65	42.6	41.7	41.4	41.7	41.7	41.65
Low	42.8	42.1	41.85	42.1	41.8	40.6	40.5	40.9	41.15	41
Close	43	42.25	42.1	42.25	41.8	40.75	41.3	41.3	41.45	41.45
Volume	2842933	7654419	8719024	8697776	10494129	14743805	9078712	10408761	7820632	13346241
Hold	0	0	0	0	0	0	0	0	0	0

Figure 6.2: State Input to CNN Model

3. Action

   We suppose that agent can act five actions: buy 100 shares, buy 10 shares, hold, sell 10 shares, and sell 100 shares. If the agent buys or sells a stock, we will trade at the closing price of the current bar.

DOI:10.6814/NCCU202200014



Action	Trades Quantity
0	Buy 100 shares
1	Buy 10 shares
2	Hold
3	Sell 10 shares
4	Sell 100 shares

Figure 6.3: Action

4. Reward

   Whenever the agent makes a trade, we will give a negative reward as commission, just like the real situation. When the agent sells the stocks, we will return a reward as profit, which may be positive or negative. On the last day, we will clear all hand-on stocks and calculate a reward to return as profit.

6.3 Initial Parameter Settlement

We set the initial funds as 100,000 and no shares of stocks on hand at the beginning. To enrich our state-action pairs in the replay memory, we will choose a random offset from the beginning of the data and up to 200 steps in an episode in which we use ϵ-greedy with $\epsilon = \frac{1}{0.1\times\text{episode}+1}$ to choose actions. The learning rate is set as 0.001 and the discount rate set as 0.99. In Double Deep Q-Learning, we have the skill ”separate target network”, means that the target weight which compute Q-values of actions, are different with updating weight. We set the ” target update step” as every 2 episode, which means the target weight would equal updating weight every 2 episode.

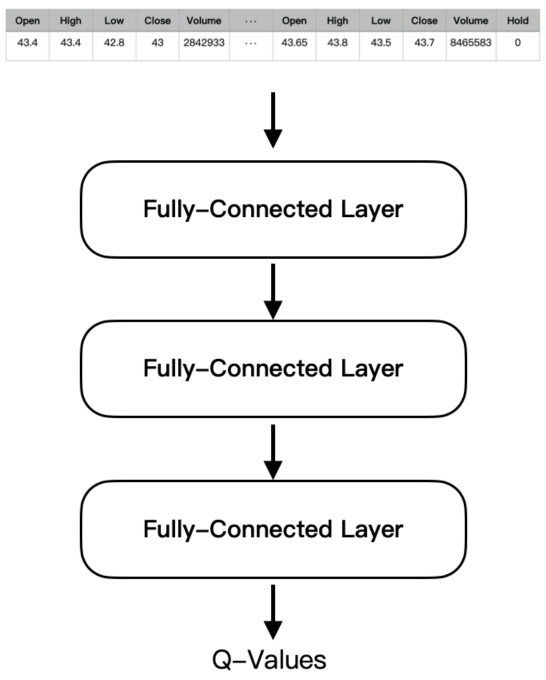
DOI:10.6814/NCCU202200014

6.4 Neural Network

We have two types of network: One is a CNN network, which contains three convolutional layers and two fully-connected layers; the other is a fully-connected network, which consists of three fully-connected layers. We compared these two architectures of DDQN and the buy-hold policy.

6.4.1 CNN in DDQN

The first layer is convolution. The kernel size and the activation function are set as 6×4 and ReLU function. The first layer of filter are 32, which output 32 channels to next layer. The second layer is also a convolution. Because the output channel from above layer is 32, the input channel in second layer is 32. We set the kernel size in second layer as 32×2, the activation function is also ReLU, and the second layer of filter are 64. The third layer is still a convolution with kernel size 64×2, ReLU function and 64 filters. The fourth and the fifth layer are fully-connected layer with ReLU function. We set them as 512 units and finally output a 5-dimension vector which represents the Q-value of the action.



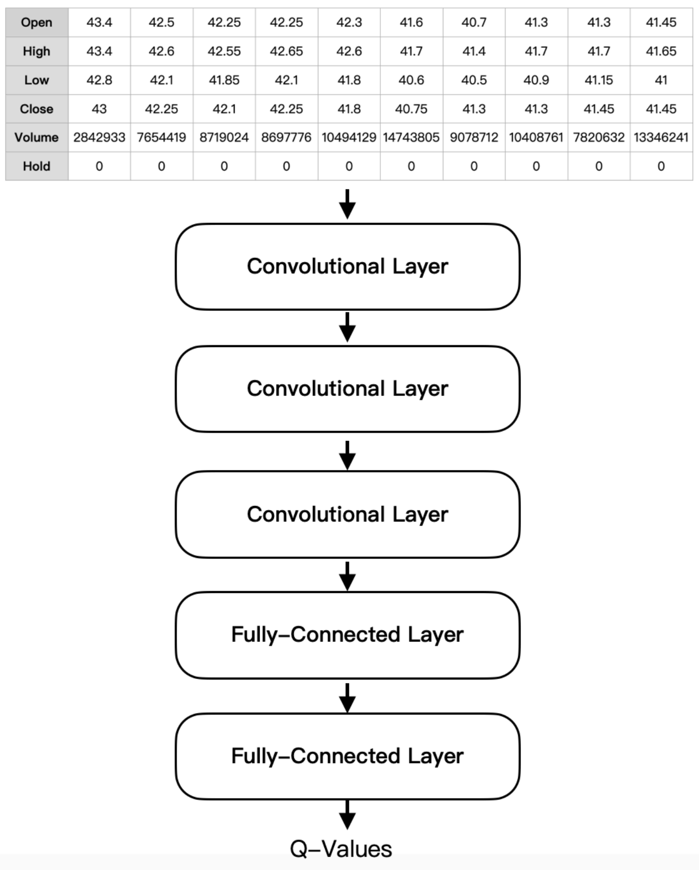
Open	High	Low	Close	Volume	···	Open	High	Low	Close	Volume	Hold
43.4	43.4	42.8	43	2842933	···	43.65	43.8	43.5	43.7	8465583	0

Figure 6.4: Fully-Connected Model

DOI:10.6814/NCCU202200014

6.4.2 Fully-Connected Network in DDQN

We use three fully-connected layers with the activation function are ReLU. The units are 512 in each three layers and finally outputs a 5-dimension vector.



Open	43.4	42.5	42.25	42.25	42.3	41.6	40.7	41.3	41.3	41.45
High	43.4	42.6	42.55	42.65	42.6	41.7	41.4	41.7	41.7	41.65
Low	42.8	42.1	41.85	42.1	41.8	40.6	40.5	40.9	41.15	41
Close	43	42.25	42.1	42.25	41.8	40.75	41.3	41.3	41.45	41.45
Volume	2842933	7654419	8719024	8697776	10494129	14743805	9078712	10408761	7820632	13346241
Hold	0	0	0	0	0	0	0	0	0	0

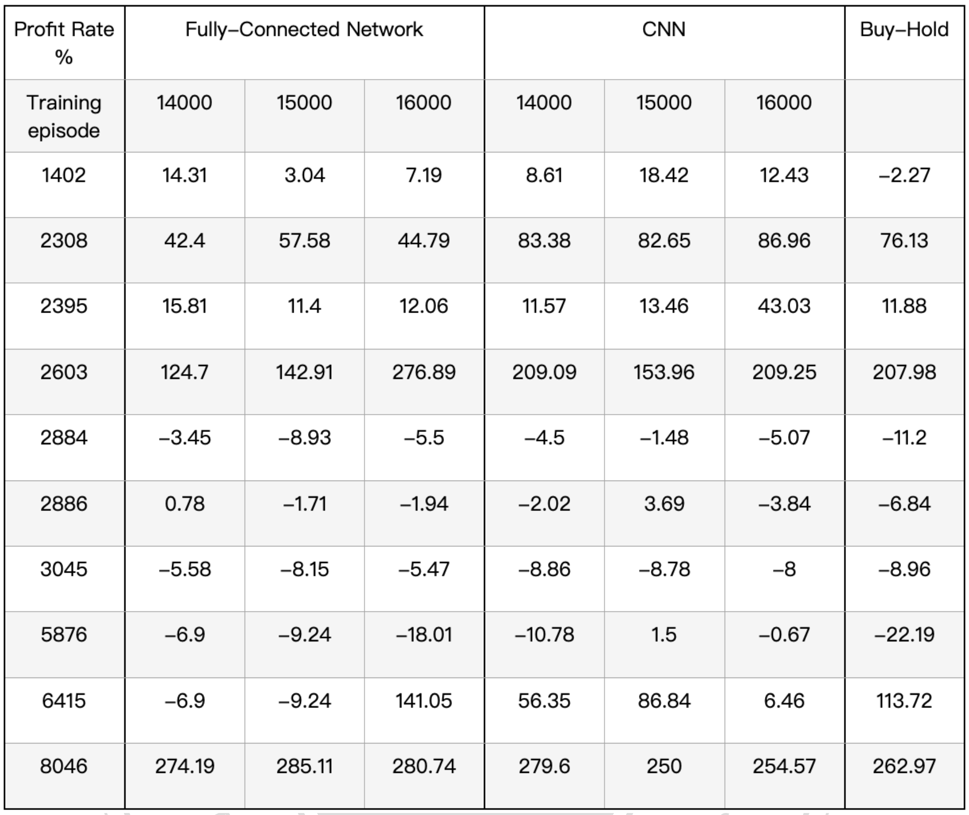
Figure 6.5: CNN Model

6.5 Result

The result is shown as following figure. We choose training episode = 14000, 15000, and 16000, where the validation result is convergence, to compare with buy-hold policy. The buy-hold policy means that we take all money to buy the stocks in the beginning and sell all in the end. We can find that whether CNN or fully-connected model are better than buy-hold policy when the value of buy-hold policy is negative. It means that our DDQN networks specializes in down-trending stocks. In CNN mode, we have six profit rates higher than buy-hold policy; three profit rates are almost even; and one profit rate is less than buy-hold policy. On the other hand, in fully-connected network mode, we have five higher profit rates; two are almost even;

DOI:10.6814/NCCU202200014

three are less than buy-hold. Both DDQN with CNN and DDQN with fully-connected network have better result than basic reward rate. And CNN mode has more stable and better result than fully-connected network mode. We attribute this to CNN's excellent performance in feature capture.



Profit Rate %	Fully−Connected Network			CNN			Buy−Hold
Training episode	14000	15000	16000	14000	15000	16000	
1402	14.31	3.04	7.19	8.61	18.42	12.43	−2.27
2308	42.4	57.58	44.79	83.38	82.65	86.96	76.13
2395	15.81	11.4	12.06	11.57	13.46	43.03	11.88
2603	124.7	142.91	276.89	209.09	153.96	209.25	207.98
2884	−3.45	−8.93	−5.5	−4.5	−1.48	−5.07	−11.2
2886	0.78	−1.71	−1.94	−2.02	3.69	−3.84	−6.84
3045	−5.58	−8.15	−5.47	−8.86	−8.78	−8	−8.96
5876	−6.9	−9.24	−18.01	−10.78	1.5	−0.67	−22.19
6415	−6.9	−9.24	141.05	56.35	86.84	6.46	113.72
8046	274.19	285.11	280.74	279.6	250	254.57	262.97

Figure 6.6: Result

DOI:10.6814/NCCU202200014

Chapter 7

Conclusion

In this paper, we construct the CNN and fully-connected network with DDQN to predict the trend of the stocks. In particular, we use different stocks from the test data, hoping to predict the unseen stocks from the existing stock information. To compared with the performance of CNN and fully-connected network, we select three nearly episodes and find that CNN has more stable and precise prediction. The result shows that CNN is good at feature capture and work in stocks trading. Although there's some stocks are not ideal in our automated trading system, most of them get good performance, the automated trading system with DDQN could achieve good outcome.

To get the better profit rate, there are two points can be improved. One is to use more skills of deep reinforcement learning, for example, priorized experience replay, dueling network, etc. The other point is to change the algorithms of reinforcement learning. Policy gradient method is also a way worth trying.

DOI:10.6814/NCCU202200014

Bibliography

[1] Fei-Ting Chen. Convolutional deep q-learning for etf automated trading system, 2017.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Machine learning basics. *Deep learning*, 1(7):98–164, 2016.

[3] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[4] Yu-Ping Huang. A comparison of deep reinforcement learning models: The case of stock automated trading system, 2021.

[5] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[7] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[9] Jerome H Saltzer, David P Reed, and David D Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.

DOI:10.6814/NCCU202200014

[10] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[11] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[12] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[13] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[14] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[15] Bayya Yegnanarayana. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

DOI:10.6814/NCCU202200014