Research in Industrial Projects for Students

Sponsor

Real AI

Final Report

Applying Q-Learning to Algorithmic Bitcoin Trading

Student Members

Chun Ho Chris Park, *Hong Kong University of Science and Technology*
chpark@connect.ust.hk

Matthew Thomas Sturm, *Massachusetts Institute of Technology*,
mtsturm@mit.edu

Katherine Thai (Project Manager), *Rutgers University–New Brunswick*,
katherine.thai@rutgers.edu

Xinyi Linda Wang, *Hong Kong University of Science and Technology*
xwangcs@connect.ust.hk

Academic Mentor

Queenie Lee, queenie.lee@alumni.stanford.edu

Sponsoring Mentor

Jonathan Yan, jyan@realai.org

August 11, 2018

This project was jointly supported by Real AI and NSF Grant NSF OISE-1460018.

Abstract

Tabular Q-learning and deep Q-learning are machine learning techniques that can learn from experience and be trained to maximize long term reward (or profit) when interacting with an environment. Specifically, they can be trained on historical data and then continue to learn in an online fashion. Bitcoin is a decentralized cryptocurrency that may be traded with US dollars. The Bitcoin market is very volatile and also very young, and thus there may be successful trading strategies that have yet to be exploited. In this paper, we apply tabular Q-learning and two separate implementations of deep Q-learning to the problem of algorithmic trading on the Bitcoin market. All of our Q-learning strategies were able to outperform the buy-and-hold strategy. As compared to the buy-and-hold strategy at a mean profit rate of -27.7%, tabular Q-learning achieved a mean profit rate of 11.8% and deep Q-learning achieved a mean profit rate of 15.0%. In addition, deep Q-learning had a smaller standard deviation in profit rate and performed better than tabular Q-learning when the price of Bitcoin was going down.

Q-learning is a machine learning method used to teach an autonomous agent how to maximize its reward through its choice of actions in a given environment. We apply this technique to algorithmically trade between Bitcoin, a decentralized cryptocurrency, and US dollars. The Bitcoin market is very volatile and also very young, so many successful trading strategies have yet to be exploited. We show that a Q-learning strategy is able to out-perform the benchmark strategy of *buy-and-hold*, where one buys a maximal amount of Bitcoin and then makes no future trades.

We trained a *deep Q-learning* model on a combination of historical price data and a set of features we designed by analyzing the sentiment of Bitcoin-related news articles and correlating them with bearish and bullish markets. While the buy-and-hold strategy has a mean profit rate of -27.7%, our deep Q-learning model achieves a mean profit rate of 15.0% respectively.

Acknowledgments

The students members of this project would like to thank Queenie Lee and Jonathan Yan for their support, guidance, and mentorship throughout the summer. This project would not have been possible without the coordination of the RIPS program by Stacey Beggs and Jorge Balbás of the Institute for Pure and Applied Mathematics (IPAM). We would also like to thank Dr. Albert Ku and Professor Tim Leung of the Hong Kong University of Science and Technology (HKUST) for their organization of RIPS-Hong Kong 2018. Thank you to the Department of Mathematics at HKUST for hosting us and allowing us to use their resources. Lastly, thank you to Dr. Lee Shiu and Dr. Jennie Mui for their generous donation in support of RIPS-Hong Kong and to the National Science Foundation for supporting RIPS.

Contents

Chapter 1

Introduction

This project was undertaken by student participants of the 2018 Research in Industrial Projects for Students (RIPS) Hong Kong program, a collaboration between the Institute for Pure and Applied Mathematics (IPAM) and Hong Kong University of Science and Technology (HKUST). The project was sponsored by Real AI.

1.1 Real AI

Founded in December 2016 by Jonathan Yan, Real AI is an artificial intelligence (AI) research and development company based in Hong Kong. At the moment, the company's focus is using AI to assist in making investment decisions. More generally, Real AI's mission is to ensure that humanity has a bright future with safe and beneficial AI.

1.2 Problem Statement

Cryptocurrencies, also known as digital currencies, have attracted the interests of investors, particularly following the dramatic rise (and subsequent fall) of the price of Bitcoin, the most popular and well known cryptocurrency. While there are many other cryptocurrencies besides Bitcoin, this project focuses only on the Bitcoin market. The Bitcoin market differs from traditional stock markets and even the foreign exchange market; for instance, Bitcoin trades 24 hours, seven days a week. The Bitcoin market is an excellent candidate for algorithmic trading, or trading in which trades are automatically executed by a computer program based on market indicators, but the differences between the Bitcoin market and other markets necessitate the development of new algorithmic trading strategies. With this, our project goal was to develop a Bitcoin trading agent using reinforcement learning and market indicators.

1.3 Approach and Results

To achieve our goal, we developed a Bitcoin trading agent, i.e. a program that automatically executes trades, using reinforcement learning and analysis of Bitcoin-related data. First, we implemented a model using tabular Q-learning. We fed Bitcoin price data as well as data from the Reddit subreddit r/Bitcoin into this model, and compared its performance to a buy-and-hold strategy. Our tabular Q-learning model yielded a maximum profit rate

of between 17.3% and 20.6% and a minimum profit rate of between -15.1% and -13.3% depending on the reward function used. Comparatively, the buy-and-hold strategy had a maximum profit rate of 1.6% and a minimum profit rate of -49.0%. Next, we implemented two different models using deep Q-learning. Bitcoin price data, Reddit data, and indicators calculated from the price data were fed into these models. The first of these models was a simple fully connected network, while the second used a DeepSense network structure [8]. Again, the performances of both models were compared to that of a buy-and-hold strategy. The first model yielded between 18.5% and 33.6% higher returns than the buy-and-hold strategy, depending on the parameters chosen. The second model yielded a maximum profit rate of between 8.3% and 32.4% and a minimum profit rate of between -29.9% and -6.9% depending on the parameter used. Comparatively, the buy-and-hold strategy had a maximum profit rate of 4.2% and a minimum profit rate of -48.3%.

1.4 Paper Outline

This paper begins with a background on reinforcement learning and specifically Q-learning, as well as a summary of relevant previous work in Chapter 2. In Chapter 3, we describe our data collection process and detail our implementations of a tabular Q-learning model and two different deep Q-learning models. Chapter 4 contains the results of our evaluations of the previously discussed implementations. Finally, we summarize our work and offer suggestions for the future direction of the project in Chapter 5.

Chapter 2

Background

2.1 Bitcoin

Bitcoin is a decentralized cryptocurrency. This means that rather than the currency being backed by a bank or a government, Bitcoin transactions are recorded on a decentralized, public ledger that makes use of cryptographic techniques to prevent fraud and theft. While there is a lot to be said about how this currency functions, for the majority of this paper, the most relevant detail is simply that Bitcoin is a currency that can be traded on exchanges with US Dollars analogously to how fiat currencies are traded. One important difference is that the Bitcoin market is quite new, which means both that the market is very volatile and that it is more likely that profitable trading strategies have yet to be discovered or exploited.

2.2 Reinforcement Learning and Q-Learning

Reinforcement learning is a machine learning technique that takes inspiration from some of the ways that biological agents learn. Broadly speaking, this type of learning involves an agent that is interacting with an environment, in which they may take actions, and for which they may get a reward. This might be similar to how a child learns about the world; certain actions, such as touching a hot stove, might have negative consequences and thus the child might learn not to repeat them, and other actions, such as putting sweet food in their mouth, might have positive consequences and thus the child might learn to perform them more in the future.

There are many examples of settings where reinforcement learning could be applicable. The agent could be a robot, the environment could be a set of blocks and bins, the actions could be how the robot moves it joints, and the reward could be whether or not the robot gets a block in a bin. The agent could be a chess player, the environment could be the board, the actions could be moves in the game, and the reward could be whether or not the player wins the game. In this paper, our agent will be the trading agent we are creating, the environment will be the Bitcoin market, our actions will trades or descriptions of what trades should be made, and our reward will be some function depending on our profits.

More formally, in reinforcement learning there is a set of states, S, that the environment may be in and a set of actions, A, that the agent may take. Further, there is a stochastic transition function $P : S \times A \to S$ which determines given the current state and the action chosen what the new state will be. In addition, there is a reward function $R : S \times A \to \mathbb{R}$

which determines given the current state and the action chosen what the reward will be, which is some real number.

Next, the total discounted reward may be defined as

$$V = \sum_{t=0}^{\infty} \gamma^t r_t,$$

where $r_t = R(s_t, a_t)$ is the reward on step t and $\gamma \in [0, 1]$ is the discount factor[1]. The effect of the discount factor is that the agent will put more value in rewards that come sooner rather than later. This makes intuitive sense in the case where rewards are currency because of the existence of interest, but the discount factor may be used even in cases even where there is no inherent reason to prefer reward sooner rather than later because adding the discount factor can help train reinforcement learning agents.

With these definitions, the goal of reinforcement learning then is to learn a policy, $\pi : S \to A$ (which is a map from the current state to the action that should be performed), that maximizes the total discounted reward, V. Naturally, there are many techniques that may be used to find this policy, but the technique we focus on in this paper is Q-learning.

A Q-function may be defined as a map $Q : S \times A \to \mathbb{R}$ where $Q(s, a)$ gives the discounted reward starting from state s that results from first taking action a, and then thereafter acting optimally (taking action a may or may not be optimal). Thus, if we know the Q-function, the optimal policy π is given by $\pi(s) = \text{argmax}_a[Q(s, a)]$. In this way, the problem of learning the optimal policy reduces to learning this Q-function.

One very important equation that this Q-function will satisfy is the Bellman equation, given by

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma \max_a \left[Q(s_{t+1}, a)\right].$$

What this equation is saying is just that the Q-value, $Q(s_t, a_t)$, is given by the immediate reward from taking action a_t, or $R(s_t, a_t)$, plus the accumulated reward from thereafter acting optimally, or $\gamma \max_a [Q(s_{t+1}, a)]$. This follows from the definition of the Q-function. The important thing about this equation is that it relates the Q-value at state s_t to the Q-value at state s_{t+1}, and thus can be used as an iterative update.

Specifically, Q-learning is characterized by the following algorithm:

Algorithm 1: Q-learning

```
Initialize Q arbitrarily except that Q(terminal, ·) = 0;
for each episode do
    Initialize s_0 and t = 0;
    while s_t not terminal do
        Choose a_t from s_t using policy derived from Q (e.g., ε-greedy);
        Take action a_t, observe r_t and s_{t+1};
        Q(s_t, a_t) ← (1 − α)Q(s_t, a_t) + α (r_t + γ max_a [Q(s_{t+1}, a)]);
        t ← t + 1;
    end
end
```

The basic idea is that the Q-function may be initialized arbitrarily, and that the agent may then explore the space, taking actions and observing the new states and rewards that result, and then update the Q-values according to the Bellman equation. There are a few more details of this algorithm that will not be covered in depth here, such as the learning

[1]The number of steps need not be infinite, and in our case, the sum will be finite.

rate, α, but under the right conditions, the Q-values of our function will begin to converge to the true optimal Q-values after enough episodes of the algorithm.

In this project, we implement trading agents that make use of both tabular Q-learning and deep Q-learning. Tabular Q-learning is a type of Q-learning that may be used when the state space S and action space A are finite and relatively small. In this case, algorithm 1 may be used where the Q-values are kept in a table and are updated according to the algorithm. However, it is often the case in many real world applications that the state space is either infinite or very large. In the case of a trading agent, it is most natural to treat the price of an asset, and thus the state space of the environment, as continuous. In these cases, tabular Q-learning is no longer applicable, and one must use deep Q-learning. The key difference is that deep Q-learning approximates the Q-function with a neural network, referred to as the Q-network, rather than a table of values. This allows deep Q-learning to handle continuous input. The exact implementations of deep Q-learning vary and will be elaborated on in chapter 3.

2.3 Previous Work

There has certainly been previous work that has focused on similar ideas to those in our paper. In [3], Jiang, Xu, and Liang use deep reinforcement learning to train an agent for the portfolio management problem. This problem is a little different than the one we study, in that the portfolio management problem attempts to find the optimal way to divide wealth among n different assets over time. In their case, they chose 12 cryptocurrencies, one of which was Bitcoin. In our case, our problem represents a special case of the portfolio management problem when $n = 2$ and when those two assets are Bitcoin and US Dollars. Jiang et al. tested three different implementations that make use of convolutions neural networks, a basic recurrent neural network, and a long short term memory. This paper considered the problem in a very realistic setting where commission fees were paid on trades and where currencies were selected only based on data used to train (and not on data used to test). Despite this, they were able to achieve very good performance as compared to other portfolio management strategies and were able to achieve 4-fold returns in 50 days.

Another paper in this field is [7], in which Wang et al. use deep Q-learning to build a trading agent. Specifically they use a fully connect feed-forward neural network to approximate the Q-function and chose their actions space to consist of three actions, buy, hold, or sell. Specifically, they traded on Hong Kong and US markets where HSI and SP500 were the trading targets. Wang et al. were able to use this deep Q-trading agent to outperform both the buy-and-hold strategy as well as a strategy that was learned by a recurrent reinforcement learning approach.

While neither Jiang et al. nor Wang et al. traded between Bitcoin and US Dollars, there have been several papers that did. In [2], Hegazy and Mumford analyze several trading strategies that run on the Bitcoin market. Specifically, they examine strategies that make use of weighted linear regression, boosted classifiers, Gaussian discriminant analysis, logistic regression, and recurrent reinforcement learning. Hegazy and Mumford tested these strategies both with and without a trading fee and attempted to make their analysis realistic with respect to the Bitcoin market. Of the trading strategies they examine, they find that the strategy given by recurrent reinforcement learning was the most effective both with and without a commission fee.

Finally, in [4], Madan, Saluja, Zhao use machine learning techniques to predict Bitcoin price. In particular, the data they used were features generated from Bitcoin price data as

well as the Bitcoin blockchain data over the course of five years and the algorithms they tested made use of binomial logistic regression, support vector machines, and random forest algorithms. With these, they were able to achieve an accuracy of 98.7% when predicting the sign of Bitcoin price changes.

Chapter 3

Implementation

3.1 Data Collection

3.1.1 Price Data

Our data for Bitcoin price was supplied by Kaggle (an online platform that supplies data among other services). In particular, we obtained two data sets, one supplying historical price data for the Bitstamp exchange, and another for the Coinbase exchange. Both were the price data for USD/BTC trades, and the data supplied minute by minute history with the epoch time, open price, high price, low price, close price, and weighted price for that minute. Data was collected from 2015-01-19 01:00 (UTC) to 2018-06-25 00:00 (UTC). These dates were simply the widest date range at the time for which these two exchanges had data without interruptions.

From our price data, we calculated the Moving Average Convergence/Divergence (MACD), a commonly used trading indicator that describes both trends and momentum (rate-of-change) in price data. Developed in the late 1970s by Gerald Appel, the MACD is computed by subtracting a longer period exponential moving average (EMA) from a shorter period EMA. By convention, these are the 26 day EMA and 12 day EMA. However, the data we trained and tested our models on was in intervals of 30 minutes instead of one day. When used traditionally as an indicator, the MACD is plotted with a third EMA, conventionally the 9 day EMA, which is used to generate buy and sell signals. Since we fed the MACD directly into our deep learning models, we did not need this third parameter.

Let MACD *a*-*b* be the MACD calculated by subtracting the EMA with rolling window size a half hour intervals from the EMA with rolling window size b half hour intervals. We computed MACD 26-12, MACD 48-12, and MACD 1140-48. Since we have one data point for each half hour, we believe the difference between the windows for the long and short term EMAs for MACD 26-12 is too small. We believe that MACD 1140-48 (Note that 1140 half hour periods equal 23.75 days and 48 half hour periods equal one day.) will work best with our models because it has the least noise.

3.1.2 Reddit Data

Reddit[1] is a social media site where users can submit many different forms of content (e.g. text, images, links, videos) to specific communities known as subreddits. Other users can

[1] https://www.reddit.com

upvote or downvote these submissions, or posts. These votes determine the position of the post on the main page of the subreddit. Posts with many upvotes are promoted to the top of the main page. Using the Reddit Pushshift API[2], we collected the number of new posts in the Bitcoin subreddit, known as r/Bitcoin[3], containing the word "hack" in the past minute for the same time span as our price data. Note that " hack " had to appear as is in the body of the reddit submission, meaning submissions containing words in which only the substring "hack" appears (e.g. "hacking" or "Radioshack") were not included.

3.1.3 Sentiment Analysis

Current events can have dramatic effects on global markets, and algorithmic trading agents that take into account only price data cannot predict sharp rises or falls in price due to breaking news. In order to address this issue, we sought to supplement the policy learned by our reinforcement learning models with an indicator based on the sentiment of Bitcoin-related news headlines. We wanted to classify each headline as bullish, bearish, or neutral towards the price of Bitcoin, that is, whether a headline indicated that the price would rise, fall, or stay the same.

Dataset

The data was collected from two different sources: Hodl Hodl News and Crypto Panic. Both are news aggregation websites. Hodl Hodl News aggregates only Bitcoin-related news and opinion articles and has tagged some news articles as bearish or bullish (with the untagged news articles being called "neutral"). The classification method employed by Hodl Hodl News is not disclosed. Crypto Panic aggregates news on several cryptocurrencies, but headlines can be filtered by cryptocurrency and whether they are bearish or bullish towards the price of a cryptocurrency. The classification method employed by Crypto Panic is based on voting system in which visitors to the site can indicate whether they believe a headline to be bearish or bullish. Due to the very limited amount of tagged data available, our dataset was small. The table below summarizes our dataset.

	Bullish	Bearish	Neutral	Total
Number of Headlines	651	469	1221	2341

Naive Bayes Classifier

The model was built using the Naive Bayes classifier in the Python Natural Language Toolkit (NLTK) library. The Naive Bayes classifier is based on a conditional probability model. The classifer choses the class for the headline that maximizes the probability that a headline belongs to that class given its features. In our case, those features were simply the words in the headline.

Implementation and Results

All headlines were made lowercase before feature extraction. Features were the words present in a headline after using the `word_tokenize()` method from the NLTK library. Feature sets were vectors containing the features (words) in each headline and a boolean

[2]https://github.com/pushshift/api
[3]https://www.reddit.com/r/Bitcoin

value of True for each feature. 60% of the data was used to train the model, and the remaining 40% was used to test.

The classification accuracy rate was 69.0171%. This was fairly good given that we had to classify headlines into three classes, and we did not refine our features. Future work could focus on developing better features (e.g. word combinations in headlines) and obtaining more tagged data.

3.2 Tabular Q-Learning

For the implementation of tabular Q-learning, we had to fit the continuous price data into discrete form, as the state space had to be finite and discrete in order to run the algorithm. In order to discretize the data, we decided to analyze the price level from different perspectives according to the timeframe. We used four variables for the model, which were relative price of 1 day, 3 days, 7 days, and 30 days, to form a state. We thought that the model could gain insight on which action to take by analyzing where the market stands from different viewpoints.

We sampled the data by hour, processed each price data into different bins according to where it stands compared to the average price in the past timeframe. For example, we would find the mean price of the past 24 hours of the current data point, and set thresholds for different bins according to the normal distribution to classify the current data point. For the final model we implemented, we used three bins $(-1, 0, 1)$ where negative-numbered bins mean that it was more than 0.425 standard deviation below the mean in the price distribution in the past timeframe, and vice versa for positive-numbered bins. Therefore, the state space had four parameters, one for each timeframe, and each parameter could take three values.

For calculating the reward, we tried three different reward functions:

1. $r_t = -(z_t - z_{t-1})a_t$
2. $r_t = -(z_t/z_{t-1} - 1)a_t$
3. $r_t = \left(1 - a_t \frac{z_t - z_{t-1}}{z_{t-1}}\right) \frac{z_{t-1}}{z_{t-n}}$

where z_t is the Bitcoin price at timestep t, and a_t is the action we chose among buy (1), hold (0) and sell (-1).

The first and second reward function simply reflect the price change between the current and previous timestep. The third reward function is borrowed from [7], which reflects the accumulative price over n timesteps instead of the instantaneous reward to reduce the noise.

The tabular Q-learning algorithm is as below, similar to the one in 2.2:

Algorithm 2: Tabular Q-learning

Initialize all entries of the Q-table to 0;
while s_t *not terminal* **do**
  for *each* $a_t \in \{-1, 0, 1\}$ **do**
    Observe r_t and s_{t+1};
    90% of time: $Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_a [Q(s_{t+1}, a)]\right)$;
    10% of time: $Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha\left(r_t + \gamma \operatorname{random}_a [Q(s_{t+1}, a)]\right)$;
    $t \leftarrow t + 1$;
  end
end

Since we are doing per hour trading, executing an action means the time passes by one hour and the state only depends on the Bitcoin price. The states are actually fixed no matter what action we take, so we simply try all the three actions at every state and update all these three entries of the Q-table simultaneously. The agent utilized ϵ-greedy exploration policy with $\epsilon = 0.9$, so it picked a random Q-value of the next state 10% of the time and it took the maximum Q-value of the next state 90% of the time. The learning rate α was set to 0.8, and discount factor γ to 0.9.

3.3 Deep Q-Learning

Our group implemented two different versions of Bitcoin trading agent using deep Q-learning. The details on the implementations of each model will be elaborated on in sections 3.3.1 and 3.3.2. This section aims to explain the general concepts that are needed for the implementation of deep Q-learning algorithms.

Deep Q-learning is different from the previous tabular version of Q-learning in that the Q-values that are being learned are not stored in a table, but are parameterized by a neural network. The goal of deep Q-learning is to learn this neural network that estimates the Q-values according to the input vectors describing the states. The general outline of the deep Q-learning algorithm will be to observe each state, take an action according to the ϵ-greedy policy, observe the reward, and learn the neural network from the observed data. Experimental results from Mnih et all in [6] suggest that RMSProp for optimization algorithm and Huber loss as the loss function yield optimal results, so this is what our model used. Besides from this general idea of deep Q-learning, there are two methods which play a vital role in optimizing the learning process, which are experience replay and target network update.

First of all, experience replay is implemented by keeping a memory of experiences observed from the environment, randomly sampling a mini-batch from the memory, and fitting the neural network to the data in the mini-batch. The memory is a series of experiences, $e_t = (s_t, a_t, r_t, s_{t+1})$, gained at time t, which is a vector of the current state, action taken, reward, and next state. The memory is implemented by using a double-ended queue, and each time a new experience is stored into the memory, the oldest experience is eliminated. After a given amount of time, a new experience is stored, the mini-batch is sampled from a uniform random distribution of the memory, and the neural network is fit into the data in the mini-batch.

There are a few reasons why experience replay is vital for deep Q-learning. Firstly, it allows the network to learn from past experience repeatedly. This is quite helpful for making the neural network converge to the right values, especially when the number of data available is limited. Secondly, this method removes the local correlations in the data, therefore partly fulfilling the assumption that the data is sampled independently from an identical distribution, which is held for most convergence proofs in supervised learning.

Target network update is implemented through keeping a separate neural network for calculating the target values, and only periodically updating that network. Previously, it was shown that the target Q-value to fit the network into is given by the equation $Q(s_t, a_t) = r_t + \gamma \max_a[Q(s_{t+1}, a)]$. Here, we examine that the maximum estimated future Q-value is calculated through the neural network. The problem that arises from this is that each time the network is updated to some input and output vector, the changes are generalized over all states, and the target values for every state keeps changing. This is a very big problem for converging the neural network, because the values that we are trying

to learn is not fixed. Therefore, keeping a separate target network and only updating this periodically gives room for the model to learn the target value, which will be fixed for some time period. Appropriate adjustment for this time period is needed, as updating the target network too slowly will prevent the model from taking into account newer Q-values.

3.3.1 Using a Fully Connected Q-Network

The first neural network structure is a very simple one, consisting of four densely connected layers with 24 units in each layer, and ReLU activation. The idea for this model was to provide a relatively small, simple network with state inputs that are already descriptive of the environment. Therefore, the state inputs that were provided were rolling means, rate of price change, and MACD indicator. The data set used to train this model was taken every 30 minutes, as we thought that per-minute data would introduce too much noise to the model. As the data set was quite small, the network sampled a mini-batch and learned the network every time it observed a new data from the environment. The target network was updated every 80 observations. The action space was the percentage of total value held in Bitcoins, ranging from 0% to 100%, in 10% increments. The reward function used at first was the current portfolio value minus that of five hours ago. Later on, we changed the reward to the difference between portfolio value before and after the current action, which reflects more on the immediate reward.

3.3.2 Using DeepSense Network Structure

We have also implemented a more complex neural network, referring to the DeepSense structure introduced in [8]. The DeepSense is originally designed for processing time-series inputs from sensors and it is good at reducing undescribable noise. Since the data we use to train our bitcoin trading agent is also time-series data, it is reasonable to use the DeepSense network in our model.

The inputs of the whole model have five channels: price difference between the current and previous price, high price in the past 30 minutes, low price in the past 30 minutes, MACD and the Reddit data (see 3.1.2). We regard the current data and the previous 180 data as a whole, which form a state s_t.

The final output is an integer, the index of the action chosen. At first, we designed the action a_t as the percentage of the current total asset in Bitcoin market, and the possible values were: $\{0\%, 5\%, 10\%, ..., 95\%, 100\%\}$. But after testing, we found the outputs were highly unstable, and the performance was poor, so we suspected that the reason was that the action space was too large and the agent was not able to fully explore it. So we changed the action space back to {buy, hold, sell}, and used the same trading strategy as in 3.2. Then the results stabilized and the performance improved, which will be further elaborate in 4.2.2.

We first rearranged the inputs into the shape (9,20) to feed into the DeepSense network. The first few layers are two 2D-convolutional layers, each followed by a dropout layer, which ignore a neuron with a certain probability.

Then we created two gated recurrent units (GRU), which are similar to the long short-term memory but without an output gate and have been shown to exhibit better performance on smaller datasets. We wrapped each cell with a dropout mask to reduce the tendency to overfit (see [1]). Each GRU outputs the predicted state of timestep t and receive the output of the previous layer and the state of timestep $t-1$ as inputs. We connect

these two GRU together and only keep the output the last cell since it outputs the most recent state.

Since we sampled 24 data in each episode and used it from the first to the last, there are some data remaining unused in the current episode during each epoch. To keep this information, we concatenate it at the end of the output of the GRU layer and feed it into three fully-connected layers, each followed by a dropout layer. The output of these fully-connected layer are 3 floating point numbers, which are the Q-values of corresponding action. We chose the maximum as the predicted action of the current state.

As the network is used for the memory replay, it is more like supervised learning inside the reinforcement learning algorithm. For the Q-learning part, the reward function we used at first in this model is:

$$r_t = \frac{\text{profit}_t}{\text{profit}_{t-1}} - 1 \tag{3.1}$$

To calculate the profit, we kept a portfolio file and rest the portfolio when we are using a new episode.

But there is a problem of using this reward function. The Q-values are the total discounted sum of the immediate rewards, and it is kept throughout the training and testing process. But the portfolio is rest from time to time, which means the immediate rewards changes when the agent revisit the same state. And this makes the states of an agent even more complicated and contributes to the unstability of the results.

So we change the reward function back to the second one in 3.2, which is:

$$r_t = -(z_t/z_{t-1} - 1)a_t \tag{3.2}$$

A brief outline of the training algorithm is as follow:

Algorithm 3: deep Q-learning

Initialize all the weight of the DeepSense network randomly;
for *each randomly chosen episode of size 24* **do**
  Rest the portfolio;
  while s_t *not terminal* **do**
    Choose a_t from s_t using ϵ-greedy policy;
    Take action a_t, observe r_t and s_{t+1};
    Save s_t, r_t, a_t to the memory;
    if t (mod 4) *is 0* **then**
      Train the DeepSense network:
      Inputs: randomly sampled data from the memory;
      Target Q values: $Q_t(s_t, a_t) \leftarrow r_t + \gamma \max_a [Q(s_{t+1}, a)]$;
      ($Q(s_{t+1}, a)$ is learned from the DeepSense network)
    end
    if t (mod 500) *is 0* **then**
      Update target network: copy the weights from the training network to target network.
    end
    $t \leftarrow t + 1$;
  end
end

Where ϵ is decreasing during the training process, ranging from 1.0 to 0.1.

Chapter 4

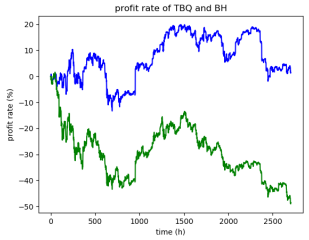
Results

4.1 Tabular Q-Learning

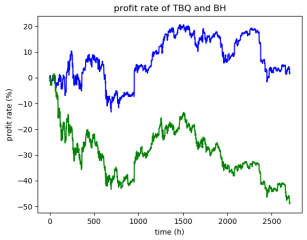
To evaluate the performance of the tabular-Q learning (TBQ) model implemented in 3.2, we compared it with the buy-and-hold strategy. Since our model only outputs 3 actions (buy, hold and sell), the following strategy is used to carry out the actions:

If the action is sell, sell bitcoins to make sure that at most 10% of the total assets are in the bitcoin market. If the action is buy, buy bitcoins to make sure that at least 90% of the total assets are in the bitcoin market. If the action is hold, do nothing.

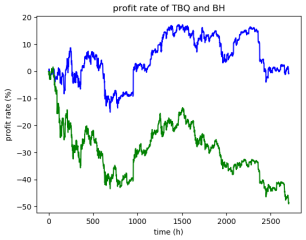
The profit rate is the ratio of profit to the initial assets. The following figure shows the profit rate of TBQ model versus buy-and-hold (BH) strategy using the per hour Bitcoin price from 2017-07-13 21:30 (UTC) to 2018-03-03 17:30 (UTC) (the first 70% of data) as training set and the Bitcoin price from 2018-03-03 18:30 (UTC) to 2018-06-24 10:30 (UTC) (the last 30% of data) as testing data. The testing results of using the three different reward functions in 3.2 are as follows:



(a) $r_t = -(z_t - z_{t-1})a_t$



(b) $r_t = -(z_t/z_{t-1} - 1)a_t$



(c) $r_t = \left(1 - a_t \frac{z_t - z_{t-1}}{z_{t-1}}\right) \frac{z_{t-1}}{z_{t-n}}$

Figure 4.1

The blue curve represents the profit rate of the TBQ model and the green curve represents the profit rate of the BH strategy. The profit rate is calculated by the current profit divided by the initial total asset. Here is some statistical data of the performance of TBQ:

	max	min	std	end
TBQ ($r_t = -(z_t - z_{t-1})a_t$)	20.5958%	-13.3095%	8.4996%	1.5814%
TBQ ($r_t = -(z_t/z_{t-1} - 1)a_t$)	19.8469%	-13.3095%	8.3790%	1.3323%
TBQ ($r_t = \left(1 - a_t \frac{z_t - z_{t-1}}{z_{t-1}}\right) \frac{z_{t-1}}{z_{t-n}}$)	17.2510%	-15.1136%	8.0421%	-0.8625%
BH	1.6155%	-48.9986%	9.7684%	-48.5189%

In general, TBQ model performs a lot better than BH strategy, and it can be seen that the 3 reward functions produced almost the same result. The success of this implementation is that it produce stable results when repeating the training and the training time is very short, within 1 minute. But we can also see that the volatility of TBQ is almost the same as BH, which means it heavily depends on the Bitcoin market.

The reason of this success possibly is that the relative Bitcoin price regarding to different timeframes can be a very good description of the current state of the trading agent, and is highly related to the possible direction of price changes.

4.2 Deep Q-Learning

4.2.1 Using a Fully Connected Q-Network

Firstly, the training and the testing was done on the last quarter of the total Bitcoin price data. There were a total of 15,000 data points of which 12,000 were used for training and 3,000 used for testing. There were five features used, which were:

1. Price change from 1 day ago
2. Price change from 7 days ago
3. Rolling mean for 1 day
4. Rolling mean for 1 week
5. MACD

For testing, the portfolio was given 1000 USD to start with, and the portfolio value was calculated in USD value.

The first result shown in figure 4.2 was trained and tested using MACD on 24 hours vs. 6 hours. It yielded approximately 18.5% higher returns than buy and hold.

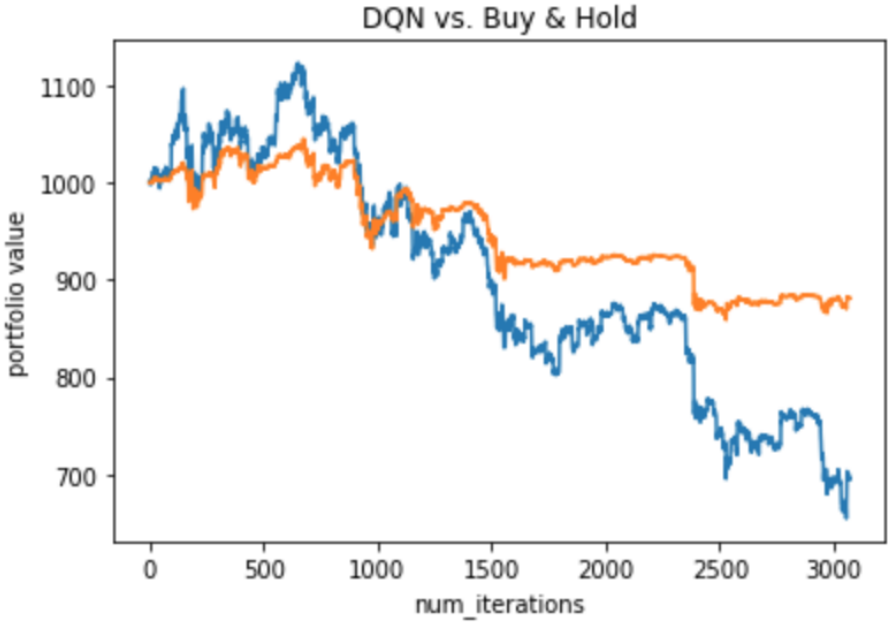


Figure 4.2: Here, the portfolio value using DQN is given in orange and that of Buy & Hold is given in blue

The second result shown in figure 4.3 was trained and tested using MACD on 570 hours vs. 24 hours (approx. 24 days vs. one day). It was speculated that this would yield higher results, as the data was less noisy. As speculated, it yielded higher returns which was approximately 33.6% higher than buy and hold.

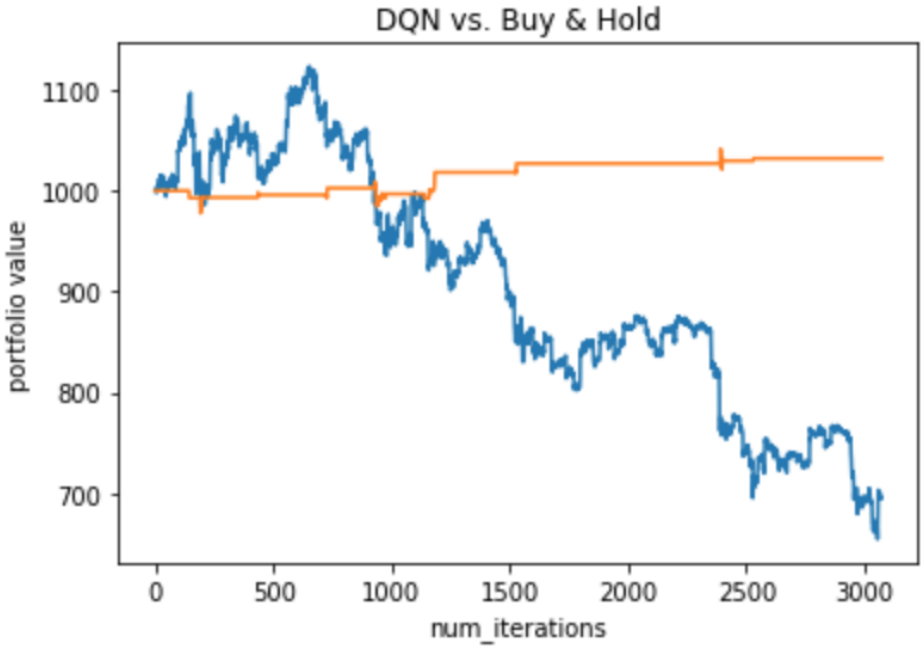


Figure 4.3: Here, the portfolio value using DQN is given in orange and that of Buy & Hold is given in blue

It can be seen that the agent was inclined not to hold any Bitcoins, and only made trades when there would be profit.

The last result shown in figure 4.4 was trained and tested using the same features as the second result, but changed the reward function to difference in portfolio value before and after the action. We wanted to test whether making the reward reflect more on immediate rewards would improve or worsen the performance. In fact, the result was slightly worse than the second, as it achieved 27.7% higher returns than buy and hold.

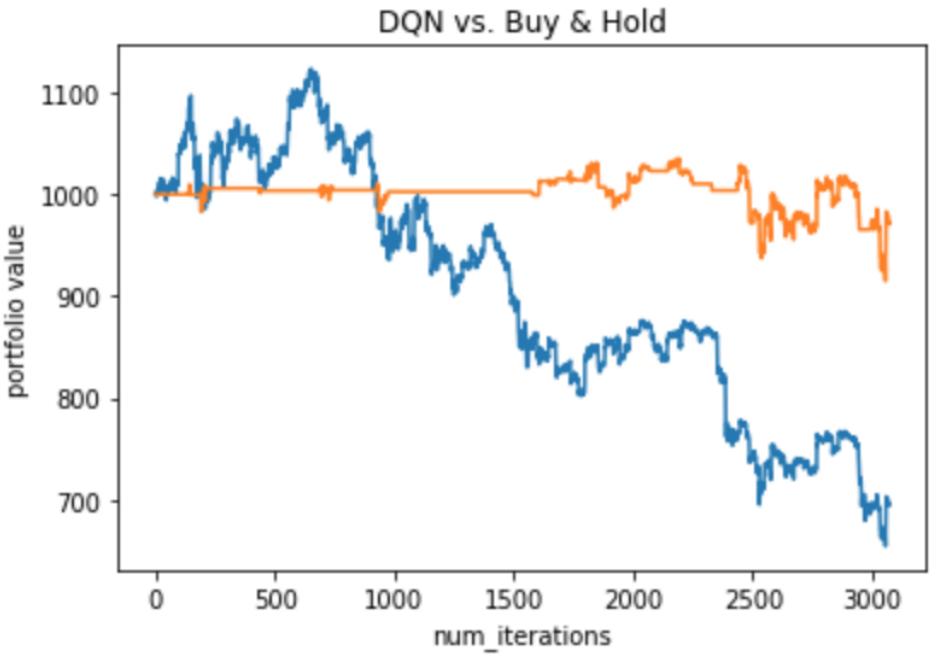


Figure 4.4: Here, the portfolio value using DQN is given in orange and that of Buy & Hold is given in blue

The following is a statistical summary of the performance for each experiment.The table includes maximum, minimum, standard deviation, and end profit rate for the 3 experiments with the DQN model and the buy and hold strategy.

	mean	max	min	std	end
DQN (MACD 48-12)	-5.1466%	4.4733%	-14.1108%	5.329%	-11.8891%
DQN (MACD 1140-48)	1.55%	4.1419%	-2.2143%	1.5266%	3.208%
DQN (immediate reward)	-0.191%	3.5233%	-8.4866%	1.5325%	-2.7034%
BH	-9.6897%	12.2857%	-34.5316%	11.9331%	-30.3943%

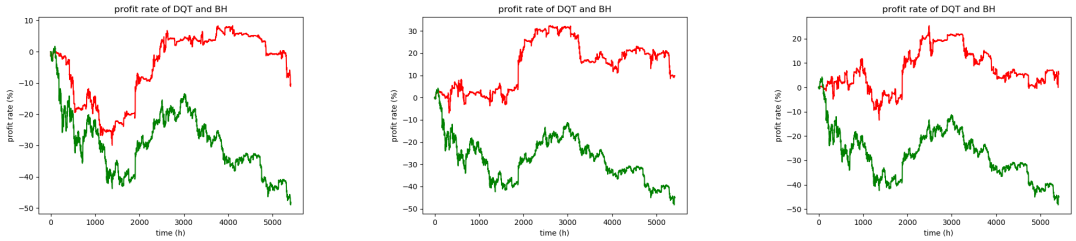
As shown above, the model that was able to achieve the most stable and highest profit rate was DQN with MACD for longer timeframe and prolonged reward function. It seemed to recognize the long-term market trend and refrained from holding Bitcoins, except for buying and selling at sharp spikes in price. When the timeframe for MACD was short, the model seemed to fit into the noisy MACD data and produce worse and unstable results. The results when using immediate reward function resulted in the agent buying large amounts of Bitcoins at the end, especially before sharp spikes in price and ended up in less profit. It seems that it is due to the fact that the agent focuses more on earning immediate profit in the current state, and tends to buy more when prices rise.

4.2.2 Using DeepSense Network Structures

The features used in this implementation, as mentioned in 3.3.2, are price difference between the current and previous price, high price in the past 30 minutes, low price in the past 30 minutes, MACD and the Reddit data (see 3.1.2). They served as the 4 channels input to the DeepSense network and also defined the state of Q-learning.

The testing environment is simulating the online learning environment, which means after each input of several new data points, we do some training to update the model.

The following figure shows the profit rate of this DeepSense Q-learning (DSQ) model versus buy-and-hold (BH) strategy using the per half hour Bitcoin price from 2017-06-17 16:00 (UTC) to 2018-03-03 07:00 (UTC) (the first 70% of data) as training set and the Bitcoin price from 2018-03-03 19:00 (UTC) to 2018-06-24 11:00 (UTC) (the last 30% of data) as testing data. The testing results using MACD 26-12, MACD 48-12 and MACD 1140-48 are as follows:



(a) MACD 26-12 (b) MACD 48-12 (c) MACD 1140-48

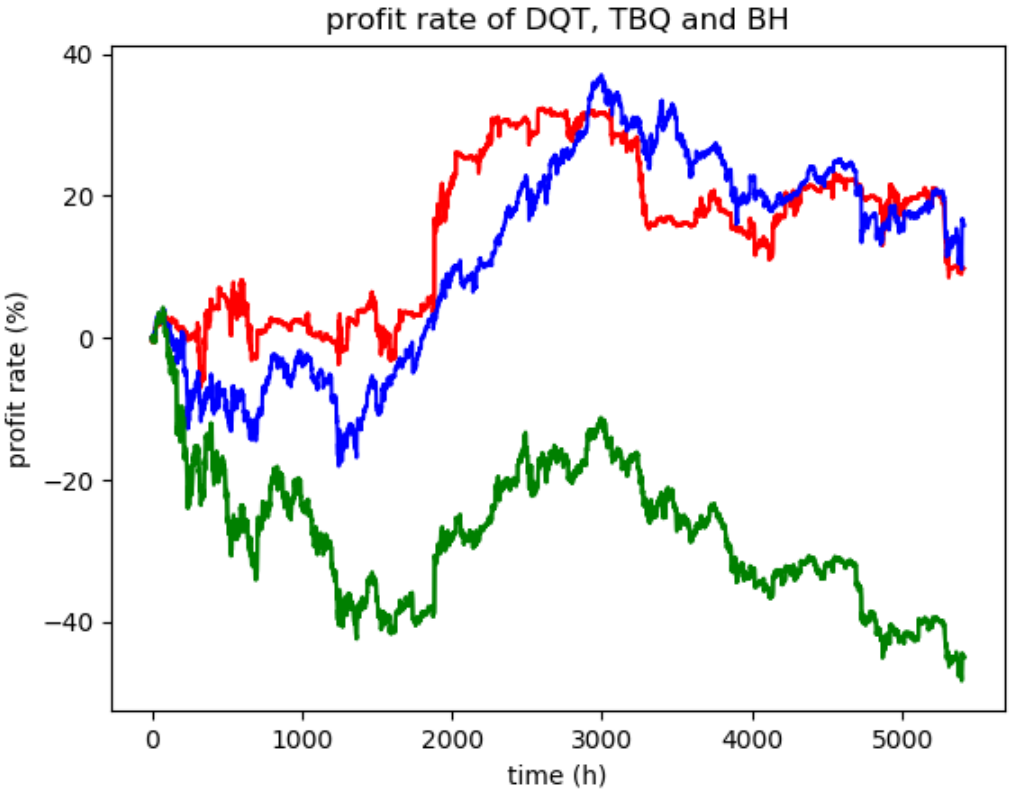
Figure 4.5

Where the red curve represents the profit rate of the DSQ model and the green curve represents the profit rate of the BH strategy. The profit rate is calculated by the current profit divided by the initial total asset. Here is some statistical data of the performance of DSQ:

	mean	max	min	std	end
DSQ (MACD 26-12)	-4.4184 %	8.2951 %	-29.9509 %	10.5540 %	-10.3785 %
DSQ (MACD 48-12)	15.0162 %	32.3667 %	-6.9510 %	10.8045 %	9.7948 %
DSQ (MACD 1140-48)	7.1467 %	25.3342 %	-13.4603 %	8.0943 %	5.4265 %
BH	-27.6702 %	4.2512 %	-48.2802 %	9.9122 %	-45.0118 %

In general, DSQ model performs better than BH strategy. But the problem of this implementation is that its performance is not very stable, which means the results of repeating the training turns out to be different. This is possibly because in the beginning of the training, in order to better explore the whole space of states, the policy we take is highly randomized, so the Q-values in the beginning can be very different when from training to training. Since we only have around 30,000 data in total, the differences in the beginning are very likely to heavily influence the final results. One possible solution is to increase the batch size when doing memory replay, which will increase the training time.

We use the same data set to train the tabular Q-learning (TBQ) model as well, without changing any parameters. The only change is that the per hour Bitcoin price is now per half hour Bitcoin price. Then we plot the TBQ, DSQ (use MACD 26-12) and BH together, the result is as follow:



Where the blue, red and green curves represent the profit rate of the TBQ model, the DSQ model and the BH strategy, respectively.

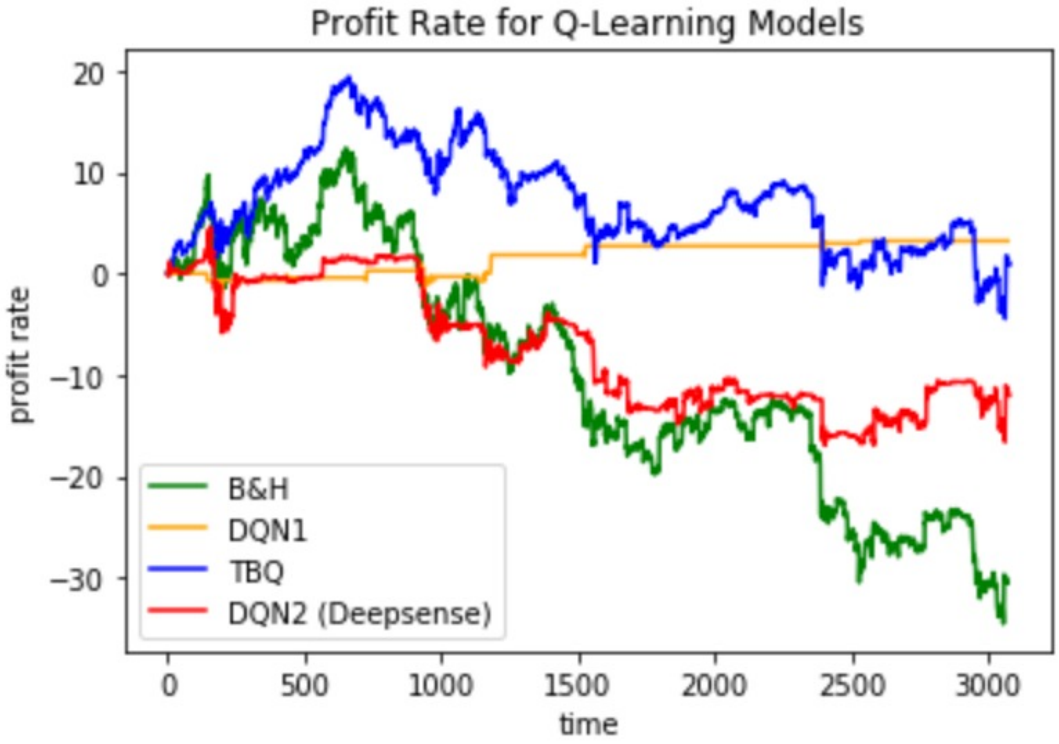
	mean	max	min	std	end
DSQ (MACD 48-12)	15.0162 %	32.3667 %	-6.9510 %	10.8045 %	9.7948 %
TBQ ($r_t = -(z_t/z_{t-1} - 1)a_t$)	11.7895 %	37.0355 %	-18.0724 %	14.4897 %	15.7938 %
BH	-27.6702 %	4.2512 %	-48.2802 %	9.9122 %	-45.0118 %

We can see that although DSQ made less money than TBQ in the end, its standard deviation is smaller than TBQ and it generally performed better than TBQ when the Bitcoin price is going down.

Chapter 5

Conclusion

Reinforcement learning allows an agent to learn according to rewards gained from its environment, which is a very suitable technique for building a Bitcoin trading agent since its ultimate goal is to earn money. In this project, we specifically used Q-learning to achieve this goal. Three different models were implemented: the tabular Q-learning model, the deep Q-leaning model with a fully-connected Q-network, and the deep Q-leaning model with a DeepSense Q-network. The tabular Q-learning model turns out to be the most profitable and the other two deep Q-learning model turns out to be more conservative. The graph below is a comparison between these three models and the buy-and-hold strategy:



All the three models generally perform better than buy-and-hold when the price is dropping which means they actually learned from the past cases. Since it is not possible for any agent to perform as well as the buy-and-hold strategy when the price is rapidly rising

(unless the agent is able to invest the entire portfolio into Bitcoin), the agents generally perform worse than the buy-and-hold strategy.

The challenge of bulding a Bitcoin trading agent is that the Bitcoin market is very volatile and since Bitcoin has only been attracting investment in a recent few years, the usable data set is very small, so it is very hard to have a convergent Q-table or get a stable result when doing deep Q-learning. But the advantage of doing algorithm trading of Bitcoin is that the Bitcoin market is very close to an isolated system and is influenced by less factors compared to the stock market. With more Bitcoin data generated in the future, it is expected that the three trading agent we implemented will perform better and better in the future. Also, we believe that implementing an automated trading agent that integrates deep learning techniques and sentiment analysis models will be a very challenging but profitable task. Although we were not able to entirely integrate these two components in this project, allowing the agent to absorb useful information from the media and make judgments on market trends based on this will prove very useful.

Selected Bibliography Including Cited Works

[1] Y. Gal and Z. Ghahramani, *A theoretically grounded application of dropout in recurrent neural networks*, 2016 Conference on Neural Information Processing Systems, (2016).

[2] K. Hegazy and S. MUMFORD, *Comparitive automated bitcoin trading strategies*, 2016.

[3] Z. Jiang, D. Xu, and J. Liang, *A deep reinforcement learning framework for the financial portfolio management problem*, CoRR, abs/1706.10059 (2017).

[4] I. Madan, *Automated bitcoin trading via machine learning algorithms*, 2014.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, *Playing atari with deep reinforcement learning*, CoRR, abs/1312.5602 (2013).

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, *Human-level control through deep reinforcement learning*, Nature, 518 (2015), pp. 529–533.

[7] Y. Wang, D. Wang, S. Zhang, Y. Feng, S. Li, and Q. Zhou, *Deep q-trading*, Center for Speech and Language Technology, Research Institute of Information Technology, Tsinghua University, (2017).

[8] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, *Deepsense: A unified deep learning framework for time-series mobile sensing data processing*, World Wide Web 2017, (2017).