Learning Financial Asset-Specific Trading Rules via Deep Reinforcement Learning

Mehran Taghian[a,b], Ahmad Asadi[a,b], Reza Safabakhsh[a,b,*]

[a] *Deep Learning Lab, Computer Engineering Department*
[b] *Amirkabir University of Technology, Hafez St., Tehran, Iran.*

Abstract

Generating asset-specific trading signals based on the financial conditions of the assets is one of the challenging problems in automated trading. Various asset trading rules are proposed experimentally based on different technical analysis techniques. However, these kind of trading strategies are profitable, extracting new asset-specific trading rules from vast historical data to increase total return and decrease the risk of portfolios is difficult for human experts. Recently, various deep reinforcement learning (DRL) methods are employed to learn the new trading rules for each asset. In this paper, a novel DRL model with various feature extraction modules is proposed. The effect of different input representations on the performance of the models is investigated and the performance of DRL-based models in different markets and asset situations is studied. The proposed model in this work outperformed the other state-of-the-art models in learning single asset-specific trading rules and obtained a total return of almost 262% in two years on a specific asset while the best state-of-the-art model get 78% on the same asset in the same time period.

Keywords: Reinforcement learning, Deep Q-learning, Single Stock trading, Trading strategy,

1. Introduction

In the face of growing demand by the investors for automated trading of financial assets in different markets, designing a model capable of learning appropriate trading rules and strategies attracted the attention of researchers from finance and artificial intelligence. The power of computers to process a vast historical data of the financial assets and to model price fluctuations and learn fitted trading rules, raises the idea of designing applications that can generate appropriate trading signals on a specific financial asset at each time step.

*Corresponding author
Email addresses: mehrantaghian@aut.ac.ir (Mehran Taghian), ahmad.asadi@aut.ac.ir (Ahmad Asadi), safa@aut.ac.ir (Reza Safabakhsh)

Following the idea of replacing human traders with computer programs, a lot of research is conducted. The first issue to be addressed was that which information source should be fed to the models to enable them to make acceptable decisions. Technical analysis tools were the best candidates. Many researchers tried to verify that technical trading rules are appropriate for making investment decisions on financial assets. Different technical trading rules such as moving-average [1], trading rage break [1], and more complex indicators such as Japanese Ichimoku Kinkohyo [2] were investigated and proved that the technical strategies are able to make profit. Further investigations were accomplished to verify the performance of technical strategies in different markets [3] [4].

While a lot of research is done to verify the performance of technical trading strategies in different financial markets, the essence of learning different rules for different markets became obvious. For example, the trading rules that were performing well in predicting stock price movements in the emerging markets, have less explanatory power in more developed markets [5]. Also, the appropriate indicators to capture the sell signal do not properly capture the buy signal on the same asset [6].

The first attempts to propose a model to learn trading rules on single specific financial assets was based on genetic programming. Genetic programming was widely used to learn technical trading rules for different indices like S&P 500 index[7], to learn appropriate trading rules to benefit from short-term price fluctuations [8], to learn noise-tolerant rules based on a large number of technical indicators [9], and to learn the trading rules based on popular technical indicators like MACD [10].

One of the most important weaknesses of genetic algorithms is that they are not able to evolve after task execution, while the reinforcement learning (RL) based models are able to change during the task execution. Therefore, many researches tried to combine the RL method with genetic algorithms to benefit from the advantages of both of them [11] [12] [13].

Nowadays, considering the brilliant performance of RL based models in trading financial assets, the proposed models for learning stock trading rules are mainly based on the RL techniques. There are three types of RL techniques which are used in the proposed models: 1) Value-based methods in which the agent first estimates the value of each action in each state and then selects the action with highest value at each state, like Q-learning, 2) Policy-based methods in which the agent directly learns the policy function, like Policy Gradient (PG), 3) Actor-critic methods in which the actor generates an action at each time-step and the critic measures the quality of the generated action.

The following challenges should be faced in the models to learn trading rules for a specific financial asset:

1. Dynamic environments: The financial markets are extremely dynamic and the proposed models should be able to adapt themselves quickly with the changes in the market. The proposed models should be able to continuously learn from the model and improve their parameters and performance.

2. Models should fit on each asset in different markets: The proposed models should be able to learn appropriate trading rules and strategies for different assets in different markets using the asset-specific history of price data.
3. Feature extraction: One of the most important parts of the proposed models to learn a good trading rule for a specific financial asset, is the feature extraction phase. The quality of the extracted features directly influences the performance of the learned trading rules.

In this paper, we propose a deep reinforcement learning based model to generate single asset trading signals which outperforms the state-of-the-art methods. Furthermore different neural network structures are proposed for the feature extraction phase and the performance of each neural network is evaluated. In addition, the performance of DRL based models in learning asset-specific trading rules is studied.

2. Related Work

The proposed approaches for single stock timing and trading strategies can be divided into ~~two categories:~~ 1) Knowledge-based methods in which trading strategies are designed based on mathematics or the experiences of human experts, 2) machine learning methods in which the strategies are learned from the available historical data [14]. Lee et al. [15] showed that the knowledge-based methods are not portable enough to be used as a general trading strategy in financial markets. The limitations of human reasoning may result in poor planning, hence, the system qualities of knowledge-based methods should be carefully investigated before the execution. On the other hand, since the machine-learning approaches learn the trading strategies based on the provided historical data of each asset, they can extract more profitable patterns that human traders cannot find out conveniently.

Among different methods and techniques in machine learning, genetic algorithms and reinforcement learning are those which are used to learn single stock trading rules more frequently. Allen et al. [7] proposed one of the first methods based on genetic algorithms to learn trading rules for the S&P 500 index using daily prices. Even though, the method proposed by Allen et al. [7] was unable to earn consistent excess returns over the buy-and-hold strategy after considering trading costs, it was a start point for continuing research on learning more profitable trading rules by genetic algorithms. Potvin et al. [8] proposed a model based on genetic programming to exploit the short-term fluctuations on the individual stocks which was able to make profitable trades rather than the buy-and-hold strategy. Mallick et al. [10] also proposed a model based on genetic programming to automatically generate trading rules on the single stocks in different market scenarios. The model proposed by Mallick et al. [10] was also to ensure a positive dollar return on thirty component stocks of the Dow Jones Industrial Average index. Chien et al. [9] proposed a model based on genetic algorithm which was able to create an associative classifier to classify each time step to one of sell or buy situations.

Better performances gained by employing genetic algorithms for single stock timings motivated the community to improve the quality of such models. Chen et al. [16] proposed a time adapting genetic network programming model which was able to cope with the temporal behavior of asset prices. Chen et al. [17] employed the genetic relation algorithm and considered the correlation coefficient between stock brands as the edges in a graph structure to pick up the most efficient portfolio. Michell et al. [18] proposed a combination of the fuzzy inference system and strongly typed genetic programming to improve the efficiency of the genetic programming techniques. Michell et al. [19] also proposed a model based on strongly typed genetic programming to generate single stock trading rules focusing the fitness function on a ternary decision based on the return prediction of the corresponding stock.

One of the most important weaknesses of the genetic algorithms is that they cannot perform well in dynamic environments. One approach to improve the tolerance of such models in financial markets, which are extremely dynamic, is to combine them with different reinforcement learning techniques. Chen et al. [11] combined the genetic network programming (GNP) with SARSA learning algorithm in order to enable the evolution-based method to change the program during the task execution. Yang et al. [12] added some subroutines to GNP-SARSA algorithm which is able to call a corresponding subprogram during the execution. Fischer et al. [13] also proposed a GNP-SARSA based model with plural subroutines with different structure in which each subroutine node could define its own input and output nodes.

The reinforcement learning has the following three major benefits over other machine-learning approaches which attracted the attention of the community:

1. It needs no prior knowledge of the environment to learn the trading rules.
2. It is able to continuously adapt itself to the new situations of the environment.
3. It considers long-term benefits rather than immediate returns.

The above-mentioned advantages of the reinforcement learning encouraged the research community to think of the RL method combined with the deep neural networks, which are able to extract rich features from the environment, as an stand alone approach to learn appropriate trading strategies. Recent research showed that the combination of deep neural networks and reinforcement learning yields an extremely powerful model to learn a good policy without any knowledge about the environment. Mnih et al. [20] described the Deep Reinforcement Learning (DRL) method of Google's DeepMind team which was able to play seven different Atari games and even defeated the human top players in three of them.

Recently, DRL models are widely used to learn a good single stock trading strategy for a given stock based on its historical data. Deng et al. [21] proposed a model based on a recurrent deep neural network trained with reinforcement learning for real-time financial signal representation in an unknown environment. Wang et al. [14] proposed a model based on deep Q-learning to build an

end-to-end system for taking good positions at each trading time step. Xiong et al. [22] employed the Deep Deterministic Policy Gradient (DDPG) technique to learn a dynamic stock trading strategy which outperformed the Dow Jones Industrial Average and the min-variance portfolio allocation. Li et al. [23] examined the performance of three variations of Deep Q-network including typical DQN, Double DQN, and Dueling DQN in learning single stock trading strategies for ten US stocks and concluded that the typical DQN maximizes the decisions benefits over three methods. Luo et al. [24] combined two convolutional neural networks (CNNs) as feature extractors with a DDPG model for learning trading strategies on real stock-index future data. Zarkias et al. [25] proposed a novel price trailing method by reformulating trading as a control problem and leaned trading strategies based on trend following for taking profitable decisions. Zhang et al. [26] employed RL to design trading strategies for future contracts and investigated both discrete and continuous action spaces with a reward function modified by a volatility scaling. The authors showed that the RL method and the modern portfolio theory are equivalent if a linear utility function is used. Theate et al. [27] used Sharpe ratio performance indicator as the reward function in his proposed model.

Research on DRL models for learning trading strategies showed that the performance of the proposed models is highly dependent on the information quality of their inputs. Therefore, some researchers tried to extract temporal dependencies of price time-series to improve their proposed models. Wu et al. [28] used a Gated Recurrent Unit (GRU) to extract temporal dependencies from raw financial data and technical indicators in combination with the DQN and Deterministic Policy Gradient (DPG) models to learn a trading strategy on single stocks. Suchaimanacharoen et al. [29] first predicted the future prices of a currency pair (EUR/USD) using a CNN and then fed the forecasting prices to the Policy Gradient (PG) model to learn a trading strategy in the high frequency trading domain. Lei et al. [30] proposed a time-driven feature-aware jointly deep reinforcement learning model called TFJ-DRL which was able to learn feature representation from highly non-stationary and noisy environments and extract the temporal dependencies in an online manner, simultanously.

Even though, a wide variety of time-series models are employed to provide better representations of price movements from the historical data, the problem still remains unresolved and proposed models are not capable of learning a well-qualified feature vector to consider previous price movement behavior. One of the useful financial representation techniques to demonstrate the price movement behavior in a short period of time, is the Japanese Candlestick charts. Candlestick charting is one of the oldest methods to demonstrate the price rises and falls during a period of time which were proposed first by a Japanese merchant, Munehisa Homma, to predict the changing prices of rice [31]. Hu et al. [32] proposed a novel investment decision strategy using convolutional autoencoder learning stock representation from candlestick charts. Thammakesorn et al. [33] proposed a model for generating stock trading strategies employing features that effectively can recognize good patterns in candlestick charts. Orquin et al. [34] tested the efficiency of EUR/USD pair taking only into consid-

eration the candlestick charts of the prices. Birogul et al. [35] used the famous YOLO (You Only Look Once) object detector to detect the patterns in candlestick charts in order to generate Buy/Sell signals for a stock. Fengqian et al. [36] proposed a novel technique to generate trading strategies on single stocks using candlestick charts. In this study, an RL technique is proposed to learn stock timing given the current pattern detected from the candlestick charts of the price at each time step. The pattern detection in this study is accomplished by clustering similar candlestick patterns using K-means algorithm.

In this work, we investigated the performance of SARSA(λ) as a traditional RL technique for learning more fitted trading rules based on candlestick chart patterns for single stocks. Furthermore, a neural network based on DQN model is proposed to improve the expected return of the learned trading rules. In addition, an extra layer for learning patterns from raw OHLC prices that are better than popular candlestick chart patterns is proposed and the performance of different structure for this layer is studied. In the end, an end-to-end model for learning a trading strategy for each single asset or derivative based on its historical price data is proposed.

3. Proposed Method

3.1. Trading rules based on candlestick chart patterns

A candlestick chart is used to demonstrate the price behavior of a financial asset during a certain time window. A candlestick which is shown in figure 1 is consisted of a line demonstrating the highest and the lowest prices of the asset, and a body demonstrating the first (open) and the last (close) prices during a specific time period. Typically, if the closing price is higher than the opening price, the candlestick is colored in green or white (denoting a bullish pattern) and otherwise it is colored in red or black (denoting a bearish pattern) to show the direction of the price changes. In this work, Each candlestick is a vector θ showing the open, high, low and close prices. Equation (1) shows the candlestick vector at time step t, consisting of OHLC price. For simplicity, the close price of a candle stick is denoted by P.

$$c_t = (p_{open}, p_{high}, p_{low}, p_{close}) \tag{1}$$

In some cases, the candlesticks form popular patterns showing a special emotional situation of the traders that can be used for analysis or trading purposes. Based on these popular patterns, a list of trading rules are extracted empirically to trade single assets in financial markets. A list of such trading rules is reported in tables explained in (Appendix A) [37].

Note that, in almost all of the trading rules based on candlestick charts another parameter denoting the current market trend is used. In this work, the market trend is detected using equation (2) in which the μ_w denotes the w day moving average which is computed by equation (3), and v is a window size for the number of candles before time t to be considered in the calculation of the trend.

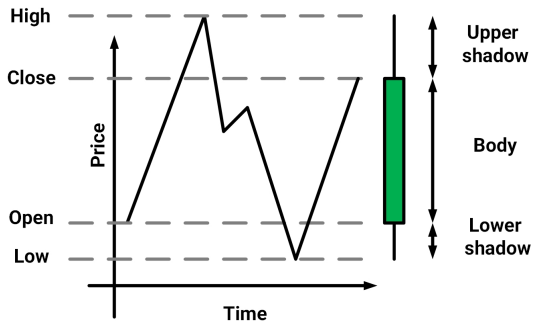


Figure 1: A candlestick representing the price behavior of an asset during a specific time window

$$MT = \begin{cases} \text{uptrend} & \text{if } \forall i \in \{0 \dots v\} \\ & \quad \mu_w(t-i-1) \leq \mu_w(t-i) \\ \text{downtrend} & \text{if } \forall i \in \{0 \dots v\} \\ & \quad \mu_w(t-i-1) \geq \mu_w(t-i) \\ \text{Side} & \text{otherwise} \end{cases} \tag{2}$$

$$\mu_w = \frac{P_{t-w} + P_{t-w+1} + \cdots + P_t}{w} \tag{3}$$

A signaling function ψ_t is defined in equation (A.1) which generates stock signals at each time step based on the occurred candlestick pattern at time t.

3.2. Trading rules learned by SARSA (λ) algorithm

The first learning algorithm which is used to learn fitted trading rules on single assets in this work, is modeled by the SARSA(λ) algorithm. The SARSA algorithm was proposed by Rummery et al. [38] in 1994 as a new modification of Temporal Difference (TD) algorithm. In this work, SARSA(λ) is used which is an off-policy version of the single-step SARSA. In this algorithm the accumulated reward function is formed as in equation (4) [39] in which T denotes the final time step in an episode, R_t denotes the immediate reward at time step t, γ denotes the discount rate, $\hat{q}$ denotes the estimate of the action-value function, S_t denotes the state vector at time step t, A_t denotes the selected action at time t, and w_t denotes the trainable parameters of the model at time step t.

$$\begin{aligned} G_{t:t+n} &= R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, w_{t+n-1}) \\ G_{t:t+n} &= G_t \quad if \;\; t+n > T \end{aligned} \tag{4}$$

The parameter's update rule in the SARSA(λ) algorithm follows the update rule structure in the temporal difference learning methods which is displayed in equation (5), in which η is the algorithm learning rate, δ_t is the temporal difference error for action-value estimation computed by equation (6), and z_t demonstrates the action-value eligibility trace vector computed by equation (7).

$$w_{t+1} = w_t - \eta \delta_t z_t \tag{5}$$

$$\delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w_t) + \hat{q}(S_t, A_t, w_t) \tag{6}$$

$$\begin{aligned} z_{-1} &= 0 \\ z_t &= \gamma \lambda z_{t-1} + \Delta \hat{q}(S_t, A_t, w_t) \end{aligned} \tag{7}$$

The immediate reward R_t here is defined as in equation (8) in which TC is the transaction cost, P_2 is the end price (say after n steps), and P_1 is the current price.

$$R_t = \begin{cases} ((1-TC)^2 \times \frac{P_2}{P_1} - 1) \times 100 & \text{if action} = \text{buy} \\ ((1-TC)^2 \times \frac{P_1}{P_2} - 1) \times 100 & \text{if action} = \text{sell} \end{cases} \tag{8}$$

The SARSA(λ) agent, takes a state vector $s_t \in S$ and an immediate reward R_t from the environment at each time step t, and produces an action $a_{t+1} \in A$ for the next time step. The state space here consists of candlestick vectors with specific types. These types are explained in section 4.5.2. The action space A denotes a set of three discrete actions $A = \{'sell', 'buy', 'idle'\}$. Algorithm 1 briefly explains the *Value Iteration* used to train.

Algorithm 1 The SARSA(λ) algorithm used in this project

```
define n as in n-step SARSA
i = 0
while i < size(data) - n do
    current-state = data[i]
    action = ε-Greedy(current-state) if current-state ≠ 'idle' else 'None'
    next-state = data[i + n]
    next-action = Greedy(next-state) if next-state ≠ 'idle' else 'None'
    Q[current-state][action] = (1 − α) × Q[current-state][action] − α ×
    (Reward_n(i) + γ^n × Q[next-state][next-action])
end while
```

3.3. Trading rules learned by Deep Q-network agent

Even though the SARSA(λ) model is able to learn better signaling rules when a certain popular candlestick pattern has occurred on a single specific financial asset, it is not possible to learn more general rules for cases that an unpopular candlestick pattern has occurred or more than one pattern appears in the input. To solve this problem, it is required to employ deep neural networks which are powerful in extracting rich feature vectors and take complex trading decisions.

In order to address this issue, a model based on the architecture of the DQN[20] is proposed which can extract rich feature vectors from the input time-series and learn good trading signals for the corresponding asset. The architecture of the proposed model is displayed in Figure 2.

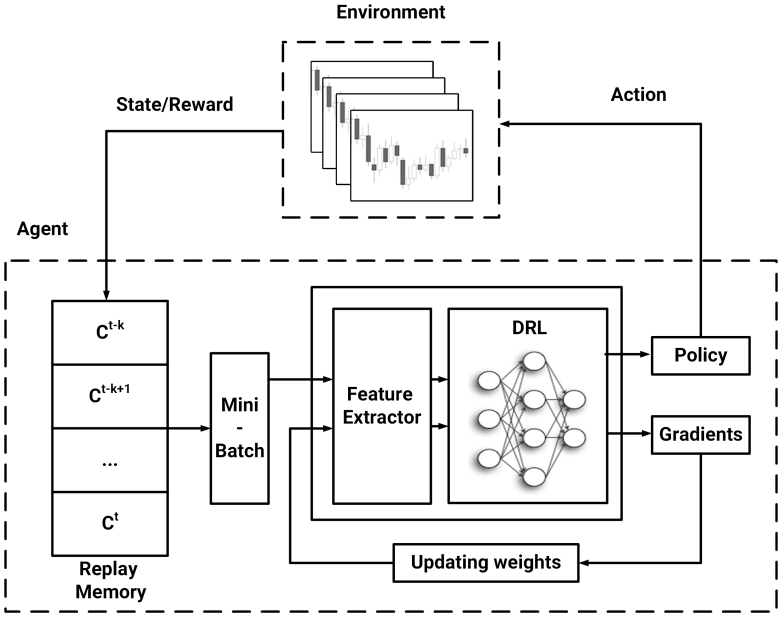


Figure 2: In this architecture, at each time step, the state is given by the environment, and the agent takes action according to the state and receives the reward and next state. The quadruples ($CurrentState, Action, Reward, NextState$) are saved in the replay memory. For the optimization part in each iteration, after the aforementioned step, a batch of quadruples ($CurrentState, Action, Reward, NextState$) is selected for the training. The replay memory has a specific capacity and after it is filled, a random quadruple is substituted with a new quadruple

The proposed model consists of two modules: feature extraction, and decision making. In this architecture, the input time-series of raw OHLC prices is passed to the first module in order to extract the candlestick pattern and form a suitable feature vector based on the corresponding candlestick. The extracted feature vector is then passed to the decision making module which is designed based on the DQN network. The output of this module is the action that agent

should take at the next time-step. It is worth noting that the next state vector and the corresponding immediate reward are passed to the decision making module directly from the environment. The architectures of these modules are discussed in detail in the following sections.

3.3.1. Decision making

In order to strengthen the proposed model to learn more powerful rules which can generate appropriate signals when an unpopular candlestick pattern is occurred, it is necessary to use a deep neural network to estimate the state action-value function (Q function) instead of using a lookup table-based approach. The decision making module in the method proposed here is structured based on the deep reinforcement learning frameworks proposed by Moody et al. [40] and Mnih et al. [20]. Suppose the target Q function is parameterized with a set of network weights Θ. According to the Bellman equation, it is possible to train the corresponding network by equation (9), in which the L_Θ denotes the network's loss function, $Q_\Theta(S, A)$ denotes the target Q function, and $\hat{Q_\Theta}(S, A)$ denotes the estimated Q function at each time step.

$$L(\Theta) = E((Q_\Theta(S, A) - \hat{Q_\Theta}(S, A))^2) \tag{9}$$

The $\hat{Q_\Theta}(S, A)$ function can be estimated using the Bellman equation as in equation (10).

$$\hat{Q_\Theta}(S_t, A_t) = R_t + \gamma \, argmax_{\hat{A}} Q_\Theta(S_{t+1}, A_{t+1}) \tag{10}$$

In order to stabilize the training process, a frozen copy of the network is created at the end of each training iteration in which the weights are not being changed during each training iteration. The value of the $Q_\Theta(S_{t+1}, A_{t+1})$ in equation (10) then, is estimated with applying this frozen network to the selected action at the current time step.

The neural network which is used to estimate the Q function, consists of three fully connected layers. The first layer, transforms the input space to a 128-dimensional latent space, the second layer transforms the 128-dimensional space to a second 256- dimensional latent space, and the last layer converts the seconds space to a 3-dimensional action space. In order to stabilizing the network and avoiding overfitting, a batch normalization layer is used between the layers. In addition, a *Softmax* layer is used to generate a probability distribution over the action space at the last layer of the network.

3.3.2. Feature extraction

The feature extraction phase of the proposed model is accomplished by a neural network which is trained jointly with the decision making module. The back propagated error in decision making module is directly used as the error at the last layer of the feature extraction module.

Different network structures including MLP, CNN, and GRU architectures are used to find the best model to extract rich features for signal generation. Since the input here is not a long sequence of asset prices (in most cases the

candlestick pattern of the current time step and in some cases the candlestick patterns appeared at the last three time step are used as input), simpler networks reached better results.

The output of the feature extractor is concatenated with the market trend extracted by equation (3) and the resulting vector is passed to the decision making module to generate stock trading signals.

3.3.3. Input vectors

To study the effect of different input representations on the performance of the model, three types of inputs are provided and fed separately to the proposed model. In the first method, a list of popular candlestick patterns are provided and at each time step a binary vector specifying the appeared candlestick pattern is computed. In the second form, the percentage of the upper shadow, the body, and the lower shadow of the candlestick are computed as displayed in equation (11) and passed to the model. In the third method, the raw values of the OHLC prices are fed to the model. In the third method, the feature extraction module is responsible for learning a good representation of the input.

$$\begin{aligned} upper &= \frac{p_h - max(p_c, p_o)}{p_h - p_l} \\ lower &= \frac{min(p_c, p_o) - p_l}{p_h - p_l} \\ body &= \frac{|p_c - p_o|}{p_h - p_l} \end{aligned} \tag{11}$$

3.3.4. Training algorithm

The process of training the model is displayed in algorithm 2.

Algorithm 2 Deep Q-Learning Algorithm used for training

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
for episode from 1 to M **do**
    Initialize sequence s_1 and preprocessed sequence $\phi_1 = \phi(s1)$
    for t from 1 to T **do**
        With probability ϵ select a random action a_t
        Otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
        Execute action a_t and observe reward r_t and state s_{t+1}
        Set $s_{t+1} = s_t, a_t$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$
        Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

$$Set\, y_j = \begin{cases} r_j & \text{if episode terminates at step j + 1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$$

        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
        Every C steps reset $\hat{Q} = Q$
    end for
end for

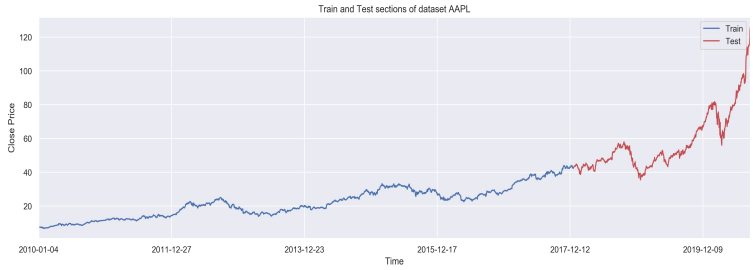
4. Experimental Results

4.1. Datasets

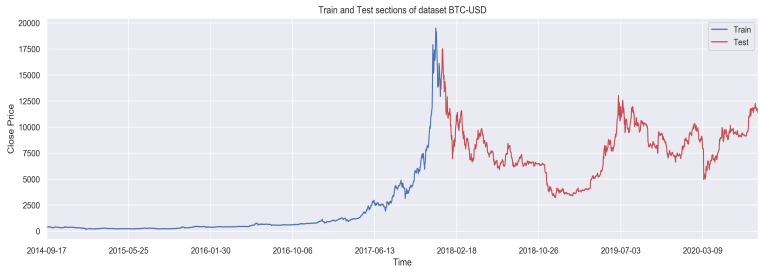
The proposed methods are tested on the real-world financial data including the stocks, currency pairs, and crypto-currencies data. For the stock data we select the historical data of AAPL, GOOGL, and KSS. For crypto-currencies data, we chose BTC/USD which is the price of Bitcoin. All the data used in this paper are available at *Yahoo Finance* and *Google Finance*. All of the candlesticks are created in daily time window according to table 1, the price history of each asset is divided into two parts, which is displayed in figure 3.

Table 1: Data used along with train-test split dates

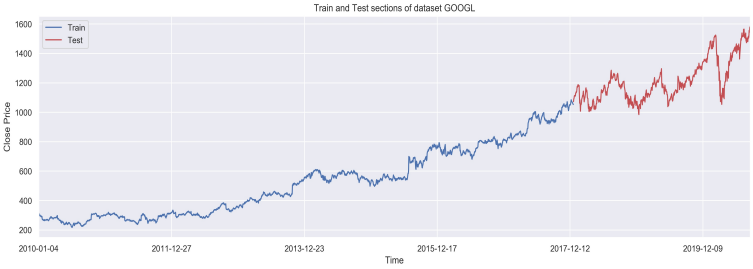
Data	Begin Date	Split Point	End Date
GOOGL	2010/01/01	2018/01/01	2020/08/24
AAPL	2010/01/01	2018/01/01	2020/08/24
KSS	1999/01/01	2018/01/01	2020/08/24
BTC-USD	2014/09/17	2018/01/01	2020/08/26



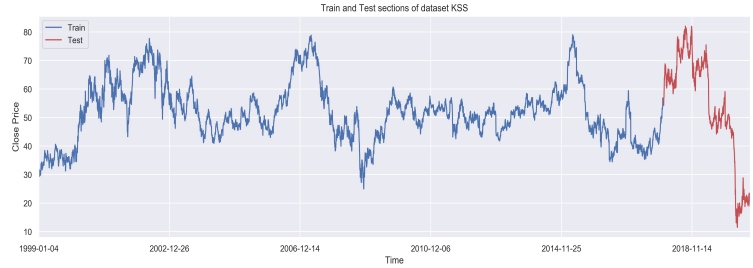
(a) Price history of AAPL stock used to train and test the model.



(b) Price history of BTC/USD pair used to train and test the model.



(c) Price history of GOOGL stock used to train and test the model.



(d) Price history of KSS stock used to train and test the model.

Figure 3: Price histories used to test the models. The blue sections are used for training and the red parts are used as testing sets.

Considering the available price histories from figure 3, extremely different assets with highly variant price movement behaviors appear in the testing data. The history used to train is illustrated in blue and the rest of the data used to test the model is illustrated in red. The AAPL and the GOOGL data have ascending trends in both training and testing parts. The BTC/USD dataset has an ascending training history while the testing part is started with an extremely descending behavior which is turned to a side trend after that. The KSS stock has a side trend in training while in the testing set a descending trend has appeared. The interesting point about the BTC/USD pair is that the model is given an ascending trend during the training, while it is tested on a descending trend while testing. Hence, the performance of the model on the KSS stock shows the generalizability of learned trading rules. Different price movement behaviors in training and testing are used to evaluate the ability of the model to learn good trading rules in even previously unknown market situations.

4.2. Evaluation metrics

The proposed model is evaluated with respect to two types of evaluation metrics: 1) metrics related to the profitability of the learned training rules, 2) metrics related to the implied risk of investment based on the learned trading rules. More details about the evaluation metrics are as follows.

i) Daily returns: The sequence of daily returns are computed by equation (12), and their average value over the testing and training period is reported. In these equations Ω_t denotes the total value of the portfolio at time step t.

$$AR_t = \frac{\Omega_t - \Omega_{t-1}}{\Omega_{t-1}} \tag{12}$$

ii) Total return: The ratio of the capital growth during testing and training time. The total return is computed by equation (13) in which Ω_0 denotes the initial investment and Ω_T denotes the value of the portfolio at the end of the period.

$$TT = \frac{\Omega_T - \Omega_0}{\Omega_0} \tag{13}$$

iii) Value at risk: The value at risk (VaR) is a metric to measure the quality level of a financial risk within a portfolio during a specific period of time. VaR typically is measured with a confidence ratio $\alpha \in (-1, 1)$ and measures the probability of gaining a return less than α in the corresponding time period. The higher the value of the VaR_α with a fixed value of α, the higher the level of financial risk of the portfolio. There exists two main approaches to compute VaR_α: 1) using the closed form which assumes the probability distribution of the daily returns of the portfolio follows a Normal standard distribution, 2) using the historical estimation method which is a non-parametric method and assumes no prior knowledge about the portfolio's daily returns. In this paper, we used the closed form method. To calculate VaR_α, we used Monte Carlo simulation by developing

a model for future stock price returns and running multiple hypothetical trials through the model. The mean μ and standard deviation σ of the returns are calculated, then 1000 simulations run to generate random outputs with a normal distribution $N(\mu, \sigma)$. Then the α percent lowest value of the outputs is selected and reported as VaR_α.

iv) Daily returns volatility: The volatility of the daily returns tells us about the financial risk level of the trading rules. The volatility is estimated using the standard deviation of the time series of the daily returns of the portfolio and is computed by equation equation (14).

$$\sigma_p = \sqrt{\frac{1}{T-1}\Sigma_{i=1}^{T}(AR_i - \mu(AR_{1:T}))^2} \tag{14}$$

v) Sharpe ratio: The Sharpe ratio (SR) was proposed first by Sharpe et al. [41] to measure the reward-to-variability ratio of the mutual funds. This metric displays the average return earned in excess of the risk-free rate per unit total risk and is computed here by equation (15) in which R_f is the return of the *risk-free* asset, and $E\{R_p\}$ is the expected value of the portfolio value. Here we assumed that $R_f = 0$.

$$SR = \frac{E\{R_p\} - R_f}{\sigma_p} \tag{15}$$

vi) Profit curve: The profit curve is a qualitative metric which reflects the profits gained by the model at each time step. In this paper, the profit curves of different models are drawn within a single chart in a specific period of time to compare the performance of these models with respect to their profitability during the investment period.

vii) Decision curve: In this curve, the trading signals to trade each asset is demonstrated over the raw price curve of that asset. This chart gives insight about the quality of decision making power of each model on each financial asset.

4.3. Baseline models

Whenever possible, the proposed models in this paper are compared with the state-of-the-art models of learning single asset trading rules. Since the implementations of the most of these models are not accessible, comparison with each baseline model is accomplished just in cases that the model performance metrics are reported in the corresponding paper. The list of the used baseline models is as follows.

i) **Buy and Hold (B&H)**: The B&H is one of the most widely used benchmark strategies to compare the performance of the proposed model with. In this strategy, the investor selects an asset and buys it at the first time step of the investment and holds it to the end of the period regardless of its price fluctuations.

ii) **GDQN**: Proposed by Wu et al. [28], using the concatenation of the technical indicators and raw OHLC price data of last 9 time steps as the input, a two-layered stacked structure of GRUs as the feature extractor and the DQN as the decision making module.

iii) **DQT**: Proposed by Wang et al. [14], implementing online Q-learning algorithm to maximize the long-term profit of the investment using the learned rules on a single financial asset. The reward function here is formed computing the accumulated wealth over the last n days.

iv) **DDPG**: Proposed by Xiong et al. [22] using Deep Deterministic Policy Gradient(DDPG) as the Deep Reinforcement Leaning approach to obtain an adaptive trading strategy. Then the performance of the model is evaluated and compared with Dow Jones Industrial Average and the traditional min-variance portfolio allocation strategy.

4.4. Details on training the models

All of the models are trained using the Adam optimizer. The mini-batch training is also conducted using a batch size of 10, and the replay memory size is set to 20. The only regularization used in the experiments is the *Batch Normalization*. The transaction cost is set to zero during the training process; however, it may be non-zero during the evaluation.

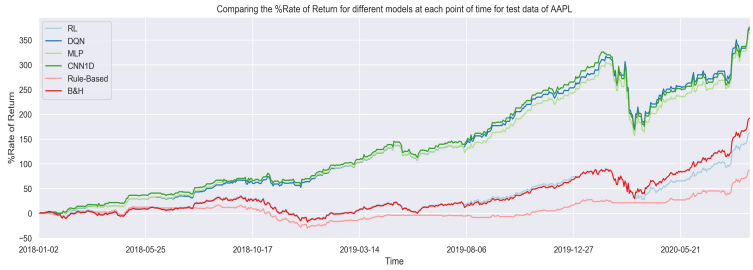
4.5. General evaluations

4.5.1. Feature extractors

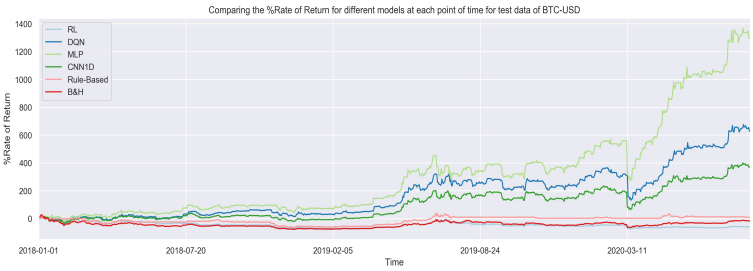
Figure 4 illustrates the profit curve of different models including the experimental trading rules based on candlestick chart patterns (Rule-Based), the benchmark Buy and Hold (B&H) strategy, the typical SARSA(λ) with the optimal $\lambda = 10$ (RL), the proposed DRL method without any feature extraction layer (DQN), the DRL model with a two-layered fully connected module as the feature extractor (MLP), the DRL model with a 1-dimension convolutional layer as feature extractor (CNN1D), and the DRL model with a 2-dimension convolutional layer as feature extractor (CNN2D). Considering the experimental results, following conclusions are drawn.

i) Almost in all of the four datasets, the performance of the SARSA agent, rule-based agent, and the B&H strategy are similar while all of the DRL models, which are trained to learn trading rules specific to each of the datasets, performed much better than them. This means that learning **asset-specific** trading rules and strategies makes more profit than seeking a model to learn general trading rules which are profitable on different assets.

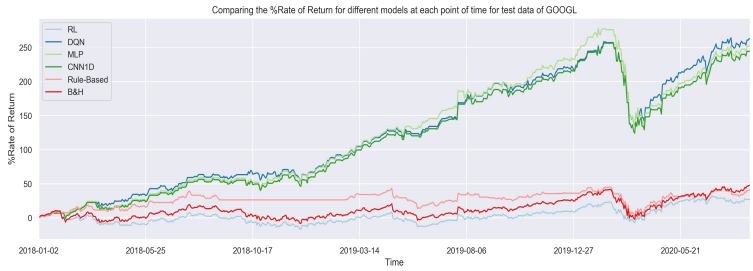
ii) Between the RL-based models, models based on deep neural networks show a better performance than the SARSA model. The main difference between these models is that, the SARSA model tries to generalize the parameters of the available candlestick pattern based trading rules while the deep RL models are able to extract different new forms of rules. Furthermore, the



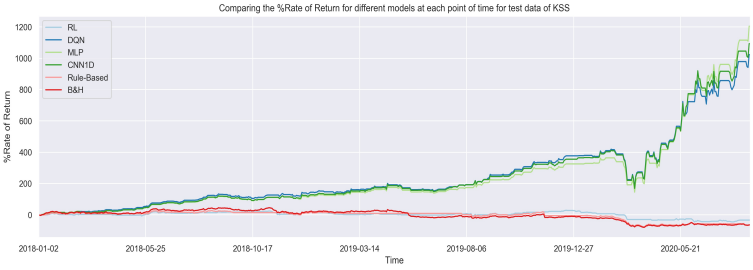
(a) Performance of different models on AAPL.



(b) Performance of different models on BTC/USD pair.



(c) Performance of different models on GOOGL.



(d) Performance of different models on KSS.

Figure 4: Profit curves of different models on each financial asset without considering different input types.

extracted trading rules with deep RL models are more profitable than the experimental rules.

The performance of different models with respect to the introduced metrics are reported and compared with the state-of-the-art models in learning single stock trading rules in table 4 on AAPL stock, table 2 on BTC/USD pair, table 3 on GOOGL stock, and figure tabletbl:KSS on KSS stock. According to the results reported in these tables, the model with the MLP feature extractor reached better total return during the testing period. One of the important points in the reported results is that the rule-based agent achieved the lowest financial risk level among the models. It is obvious that the other models learned more risk-taking trading rules because their reward function is designed based on a risk-neutral investor in which no risk-related term is appeared. The observation verifies that the form of the reward function directly impacts the behavior of the learned trading rules. If the weight of the risk-related term in the financial function is high, the learned trading rules will be appropriate for risk-averse investors.

The other important point in the results reported in tables 2, 3, 4, and 5 is that in cases that the training set is similar to the testing set, the one dimensional convolutional feature extractor performs tightly similar to the model with MLP feature extractor, while in other datasets the MLP model outperforms the CNN1D. Also, according to table 3 which illustrates the performance of models on the GOOGL with side ascending trend, the GRU model performs better than, MLP model with respect to the daily average metric. The important result here is that the feature extraction module can evidently affect the model performance and the feature extraction should be one of the most important focuses of the future research in this area.

4.5.2. Input types

Different input types provide various representations from the same information. In this experiment, we investigated the effect of using different input representations on the performance of the proposed models. The following four types of input representations are used and figure 5 illustrates the results of each model with different input types on each dataset.

i) Pattern: A binary vector representing the appearance of each introduced popular candlestick patterns.
ii) Vanilla: The raw OHLC prices.
iii) Candle-rep: The representation of a candlestick's parts including a quadruple $<u, l, b, \nu>$ denoting the percentages of the upper shadow u, the lower shadow l, the body b, and the direction of the candlestick ν (bullish or bearish).
iv) Windowed: The time-series of raw OHLC prices including the only last 3-time steps.

Since, in the popular candlestick patterns the longest sequential pattern consists of three time-steps on average (like Morning Star pattern), the inputs

Table 2: Performance of different models on BTC/USD.

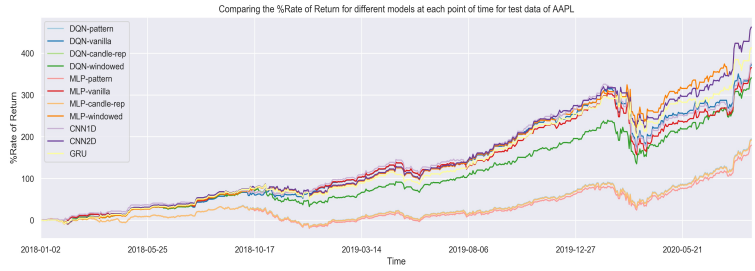
Agent	Arithmetic Return	Average Daily Return	Daily Return Variance	Time Weighted Return	Total Return	Sharpe Ratio	Value At Risk	Volatility	Initial Investment	Final Portfolio Value
Rule-Based	45.078	0.04	**7.06**	0.000	7 %	0.016	**-4.330**	**82**	1000.000	1076
B&H	60	0.06	15.78	-0.000	-16 %	0.016	-6.480	123	1000.000	830
RL	-40	-0.04	10.73	-0.001	-65 %	-0.017	-5.438	101	1000.000	340
DQN-pattern	57	0.06	15.31	-0.000	-16 %	0.015	-6.384	121	1000.000	830
DQN-vanilla	261	0.27	12.63	0.002	628 %	0.076	-5.583	110	1000.000	7287
DQN-candle-rep	50	0.05	15.68	-0.000	-24 %	0.013	-6.470	123	1000.000	757
DQN-windowed	221	0.22	12.83	0.002	384 %	0.064	-5.671	111	1000.000	4843
MLP-pattern	50	0.05	15.68	-0.000	-24 %	0.013	-6.470	123	1000.000	757
MLP-vanilla	**323**	**0.33**	12	**0.003**	**1295** %	**0.097**	-5.372	107	1000.000	**13959**
MLP-candle-rep	50	0.05	15.68	-0.000	-24 %	0.013	-6.470	123	1000.000	757
MLP-windowed	221	0.22	12.62	0.002	389 %	0.065	-5.622	110	1000.000	4892
CNN1D	219	0.22	13	0.002	370 %	0.063	-5.716	112	1000.000	4702
CNN2D	212	0.22	13.230	0.002	334 %	0.061	-5.770	113	1000.000	4348
GRU	165	0.17	13.356	0.001	168 %	0.047	-5.847	113	1000.000	2686

of the models are provided only from the last 3 time-steps. In the case that the input is provided from the last 3 time-steps, three feature extraction models are used: 1) A CNN model with 1-dimensional kernels along side the time axis (CNN1D), 2) a CNN model with 2-dimensional kernels computing an average over the spatial and temporal features (CNN2D), and 3) a GRU model to extract temporal relationships into a single feature vector.

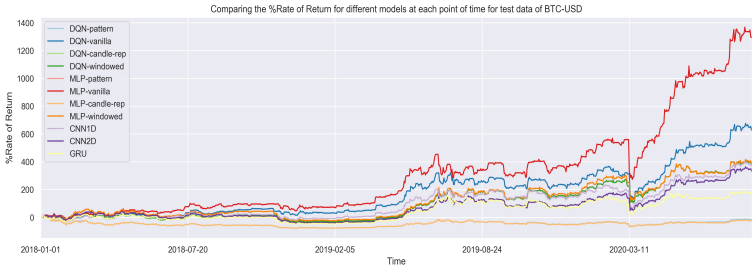
The performance of the time-series models on the assets in which the testing data follows the training data trends (GOOGL and AAPL) is obviously better than the other models. While, in the other datasets, the raw OHLC representation outperforms the other representations.

We expected that the performance of the models that use temporal relationships of the input time-series be better than the performance of the models using just the last time-step inputs; however, the experimental results obviously indicate that the models working only on the input of the last time-step perform better than the others.

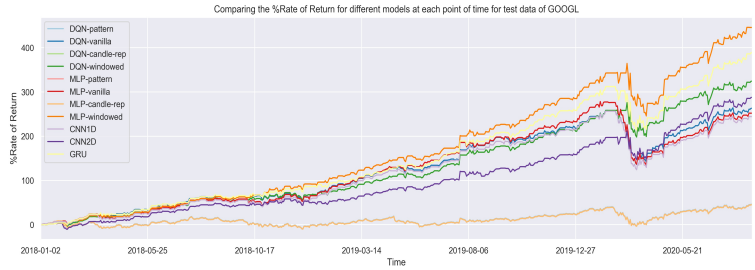
We believe that the short-time price fluctuations makes the decision making for the models difficult. In order to improve the performance of the models the length of the input time-series should be large enough to illustrate a trend in the price history of the asset. Furthermore, typically the frequency of appearance of the 3-step candlestick patterns in the input time-series is notably lower than the other patterns and the models cannot see enough data to learn appropriate rules for such cases. In summary, this observation illustrates that working on learning single asset trading signals should be limited to either only the last time-step data or a large enough time-series indicating at least a price trend in the input.



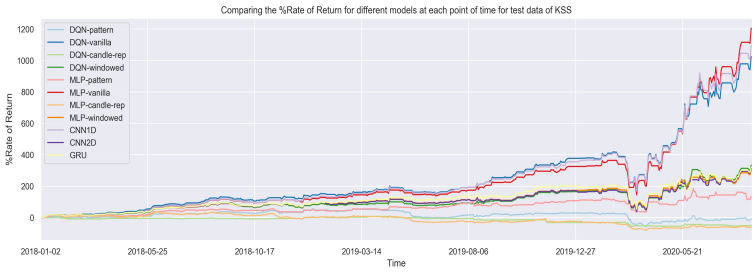
(a) Performance of different models on AAPL.



(b) Performance of different models on BTC/USD pair.



(c) Performance of different models on GOOGL.



(d) Performance of different models on KSS.

Figure 5: Profit curves of different models with different input types on each asset.

Table 3: Performance of different models on GOOGL.

Agent	Arithmetic Return	Average Daily Return	Daily Return Variance	Time Weighted Return	Total Return	Sharpe Ratio	Value At Risk	Volatility	Initial Investment	Final Portfolio Value
Rule-Based	46	0.07	2.51	0.001	40 %	0.040	-2.54	40.9	1000	1408
B&H	51	0.07	3.77	0.001	47 %	0.040	-3.12	50.1	1000	1475
RL	39	0.06	3.58	0.000	27 %	0.028	-3.05	48.8	1000	1270
DQN-pattern	50	0.07	3.77	0.001	46 %	0.039	-3.12	50.1	1000	1462
DQN-vanilla	137	0.20	2.59	0.002	263 %	0.128	-2.44	41.5	1000	3631
DQN-candle-rep	49	0.07	3.77	0.001	45 %	0.039	-3.12	50.1	1000	1452
DQN-windowed	152	0.22	2.45	0.002	324 %	0.147	-2.35	40.4	1000	4243
MLP-pattern	49	0.07	3.77	0.001	45 %	0.039	-3.12	50.1	1000	1452
MLP-vanilla	134	0.20	2.60	0.002	252 %	0.125	-2.45	41.6	1000	3520
MLP-candle-rep	49	0.07	3.77	0.001	45 %	0.039	-3.12	50.1	1000	1452
MLP-windowed	**177**	**0.26**	2.22	**0.003**	**445** %	**0.178**	**-2.19**	**38.5**	1000	**5457**
CNN1D	132	0.19	2.61	0.002	244 %	0.123	-2.46	41.7	1000	3444
CNN2D	144	0.21	2.59	0.002	287 %	0.135	-2.43	41.5	1000	3876
GRU	166	0.25	**2.43**	0.002	388 %	0.161	-2.31	40.2	1000	4888

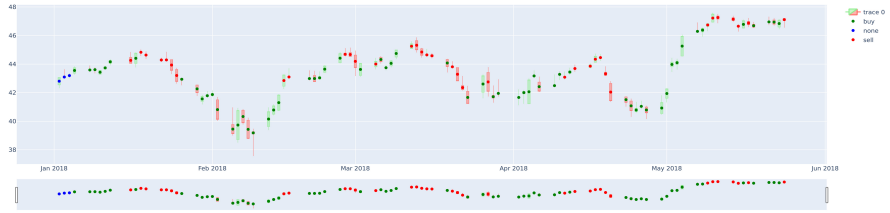
4.5.3. Learned trading rules

Figure 6 illustrates the decisions made by the best model proposed in this paper (MLP) at each time-step for different assets. In order to get a better sense about the trading behavior of this model, only 100 decisions of this model are displayed. The green points demonstrate the 'Buy' action, the red points demonstrate the 'Sell' actions and the blue points demonstrate the 'None' action. Although the actions are displayed right at the generation point in figure 6, signals generated by the model are assumed to be accomplished on the next trading day. The model is assumed to invest all of its available money at the first time that the 'Buy' action is raised. From this point while not 'Sell' action is generated, the bought asset will not be sold. Also, when the first 'Sell' action is observed, the whole of the bought asset will be sold. So, we just evaluate the first actions and do not care about the repetitive actions.

It seems that the learned trading rules, detect the bullish and bearish trends well and generate the 'Buy' signals almost in the first half of bullish trends, and also generate the 'Sell' signals in the second half of the bullish or bearish trends.

4.5.4. Comparing with similar works

Table 6 compares the performance of our best model with the state-of-the-art models using the profit metrics. According to the results reported in table 6a, the performance of the MLP model proposed in this work is significantly better than DQT and RRL on stocks HSI and S&P500 proposed by Wang et al. [14].



(a) AAPL

(b) BTC/USD

(c) GOOGL

(d) KSS

Figure 6: Trading decisions made by each model on each asset.

Table 4: Performance of different models on AAPL.

Agent	Arithmetic Return	Average Daily Return	Daily Return Variance	Time Weighted Return	Total Return	Sharpe Ratio	Value At Risk	Volatility	Initial Investment	Final Portfolio Value
Rule-Based	74	0.11	**2.11**	0.001	86 %	0.072	**-2.28**	**37.5**	1000	1869
B&H	123	0.18	4.71	0.002	191 %	0.085	-3.39	56.0	1000	2919
RL	113	0.17	4.49	0.001	162 %	0.079	-3.32	54.7	1000	2626
DQN-pattern	123	0.18	4.71	0.002	194 %	0.086	-3.38	56.0	1000	2946
DQN-vanilla	165	0.24	3.06	0.002	372 %	0.142	-2.63	45.1	1000	4722
DQN-candle-rep	123	0.18	4.71	0.002	192 %	0.085	-3.39	56.0	1000	2923
DQN-windowed	161	0.24	3.79	0.002	341 %	0.124	-2.96	50.2	1000	4410
MLP-pattern	118	0.17	4.70	0.002	179 %	0.082	-3.39	55.9	1000	2795
MLP-vanilla	164	0.24	3.15	0.002	365 %	0.139	-2.68	45.8	1000	4657
MLP-candle-rep	123	0.18	4.71	0.002	192 %	0.085	-3.39	56.0	1000	2923
MLP-windowed	**184**	**0.27**	3.52	**0.003**	462 %	0.148	-2.81	48.4	1000	5623
CNN1D	166	0.25	3.06	0.002	376 %	0.143	-2.63	45.1	1000	4760
CNN2D	184	**0.27**	3.43	**0.003**	**462** %	**0.150**	-2.77	47.7	1000	**5623**
GRU	174	0.26	3.27	0.002	412 %	0.145	-2.71	46.6	1000	5129

Table 6b represents the Rate of Return (%) of MLP model and models proposed by Wu et al. [28]. Wu et al.'s best performance is on AAPL stock with Rate of Return equals 77.7, but the MLP model gains the Rate of Return 262.3, which is greatly better. Moreover, wherever the models proposed by Wu et al. got a negative return, our model returns a highly positive return, e.g., on stock GE.

Table 6c shows the performance of DDPG, the model presented by Xiong et.al. [22]. The final portfolio value of MLP is better than DDPG, starting with an initial portfolio value of 10000.

As the results indicate, the MLP model performs significantly better than similar models in profitability. As we said before, these papers' codes were not available, and we had to compare the performance according to common metrics.

5. Conclusion

In this paper, we investigated the performance of deep reinforcement learning models in learning financial asset-specific trading rules and strategies. We proposed a novel deep reinforcement learning model based on the Q-learning technique to generate trading signals given different types of input representations. In addition, four feature extraction models were proposed to improve the DRL module performance on decision making: 1) An MLP network consisting of two fully-connected layers, 2) A convolutional model with one-dimensional ker-

Table 5: Performance of different models on KSS.

Agent	Arithmetic Return	Average Daily Return	Daily Return Variance	Time Weighted Return	Total Return	Sharpe Ratio	Value At Risk	Volatility	Initial Investment	Final Portfolio Value
Rule-Based	74	0.11	**2.11**	0.001	86 %	0.072	**-2.28**	**37**	1000	1869
B&H	123	0.18	4.71	0.002	191 %	0.085	-3.39	56	1000	2919
RL	-15	-0.02	4.71	-0.001	-33 %	-0.017	-3.59	56	1000	664
DQN-pattern	31	0.04	12.74	-0.000	-9 %	0.013	-5.83	92	1000	900
DQN-vanilla	272	0.40	9.24	**0.004**	1023 %	0.134	-4.59	78	1000	11236
DQN-candle-rep	-53	-0.08	4.08	-0.001	-49 %	-0.040	-3.40	52	1000	505
DQN-windowed	179	0.27	10.12	0.002	332 %	0.085	-4.97	82	1000	4328
MLP-pattern	106	0.16	6.74	0.001	133 %	0.062	-4.11	67	1000	2331
MLP-vanilla	**286**	**0.43**	9.13	**0.004**	**1204** %	**0.143**	-4.54	78	1000	**13048**
MLP-candle-rep	-42	-0.06	15.63	-0.001	-61 %	-0.016	-6.57	102	1000	389
MLP-windowed	170	0.25	9.55	0.002	303 %	0.083	-4.83	79	1000	4032
CNN1D	278	0.41	9.35	**0.004**	1093 %	0.137	-4.61	78	1000	11932
CNN2D	173	0.260	9.537	0.002	313.508 %	0.084	-4.82	79	1000	4135
GRU	175	0.264	9.746	0.002	320.412 %	0.085	-4.87	80	1000	4204

Table 6: Compare performance with similar works according to the profit

Agents	HSI	S&P500
MLP	13231.2	5032.3
DQT	350	214
RRL	174	141

(a) Compare profitability performance with Wang et. al. [14] based on Accumulated Return(%)

Agents	AAPL	GE	AXP	CSCO	IBM
MLP	262.3	130.4	260.2	259.9	149.2
GDQN	77.7	-10.8	20.0	20.6	4.63
GDPG	82.0	-6.39	24.3	13.6	2.55
Turtle	69.5	-17.0	25.6	-1.41	-11.7

(b) Compare profitability performance with Wu et. al. [28] based on Rate of Return(%)

Agents	DJI
MLP	21580
DDPG	19791
Min-Variance	14369
DJIA	15428

(c) Compare profitability performance with Xiong et. al. [22] based on Final Portfolio Value(Initial Portfolio Value is 10000)

nels applying to OHLC prices, 3) A convolutional model with two-dimensional kernels applying to the last 3 time steps price histories, and 3) A GRU model applying to the last 3 price data. Furthermore, the performance of different models in learning trading rules for different financial assets in various situations were assessed.

Experiments carried out on the performance of the DRL-based models demonstrate the following results:

1. Learning asset-specific trading rules profits more than general experimental rules on different assets. The deep reinforcement models outperform the traditional RL methods, benchmark strategies, and general experimental rules in this task.
2. The MLP feature extractor performs more profitable than the other feature extraction methods in cases that the price behavior of the training data is highly different than the testing data.
3. The Convolutional and GRU feature extractors are potentially able to

learn more profitable trading rules when the price behavior of training and testing sets are similar.

4. In different input representations the short-length price time-series are not good input representations for the DRL models. The raw OHLC prices are more suitable to extract asset-specific trading rules for the deep models than the candlestick charts, or popular experimental candlestick patterns.

This study, can be further improved in the future research in different ways. First, the performance of the time-series based models and feature extractors should be investigated. Second, since the learned trading rules of the models should be changed in testing time when a clear price behavior change is detected, the behavior change detection and rule changing abilities of such models should be investigated. Third, based on the experimental results, it is clear to the authors that proposing more complex feature extraction models, should inevitably improve the models performance.

References

References

[1] W. Brock, J. Lakonishok, B. LeBaron, Simple technical trading rules and the stochastic properties of stock returns, The Journal of finance 47 (5) (1992) 1731–1764.

[2] S. Deng, H. Yu, C. Wei, T. Yang, S. Tatsuro, The profitability of ichimoku kinkohyo based trading rules in stock markets and fx markets, International Journal of Finance & Economics (2020).

[3] K. Grobys, S. Ahmed, N. Sapkota, Technical trading rules in the cryptocurrency market, Finance Research Letters 32 (2020) 101396.

[4] A. Gunasekarage, D. M. Power, The profitability of moving average trading rules in south asian stock markets, Emerging Markets Review 2 (1) (2001) 17–33.

[5] H. Bessembinder, K. Chan, The profitability of technical trading rules in the asian stock markets, Pacific-basin finance journal 3 (2-3) (1995) 257–284.

[6] R. Pramudya, S. Ichsani, Efficiency of technical analysis for the stock trading, International Journal of Finance & Banking Studies 9 (1) (2020) 58–67.

[7] F. Allen, R. Karjalainen, Using genetic algorithms to find technical trading rules, Journal of financial Economics 51 (2) (1999) 245–271.

[8] J.-Y. Potvin, P. Soriano, M. Vallée, Generating trading rules on the stock markets with genetic programming, Computers & Operations Research 31 (7) (2004) 1033–1047.

[9] Y.-W. C. Chien, Y.-L. Chen, Mining associative classification rules with stock trading data–a ga-based method, Knowledge-Based Systems 23 (6) (2010) 605–614.

[10] D. Mallick, V. C. Lee, Y. S. Ong, An empirical study of genetic programming generated trading rules in computerized stock trading service system, in: 2008 International Conference on Service Systems and Service Management, IEEE, 2008, pp. 1–6.

[11] Y. Chen, S. Mabu, K. Hirasawa, J. Hu, Genetic network programming with sarsa learning and its application to creating stock trading rules, in: 2007 IEEE Congress on Evolutionary Computation, IEEE, 2007, pp. 220–227.

[12] Y. Yang, J. Li, S. Mabu, K. Hirasawa, Gnp-sarsa with subroutines for trading rules on stock markets, in: 2010 IEEE International Conference on Systems, Man and Cybernetics, IEEE, 2010, pp. 1161–1165.

[13] T. G. Fischer, Reinforcement learning in financial markets-a survey, Tech. rep., FAU Discussion Papers in Economics (2018).

[14] Y. Wang, D. Wang, S. Zhang, Y. Feng, S. Li, Q. Zhou, Deep q-trading, Cslt.Riit.Tsinghua.Edu.Cn (2017) 1–9.

[15] K. C. Lee, S. Lee, A causal knowledge-based expert system for planning an internet-based stock trading system, Expert Systems with Applications 39 (10) (2012) 8626–8635.

[16] Y. Chen, S. Mabu, K. Hirasawa, A model of portfolio optimization using time adapting genetic network programming, Computers & operations research 37 (10) (2010) 1697–1707.

[17] Y. Chen, S. Mabu, K. Hirasawa, Genetic relation algorithm with guided mutation for the large-scale portfolio optimization, Expert Systems with Applications 38 (4) (2011) 3353–3363.

[18] K. Michell, W. Kristjanpoller, Strongly-typed genetic programming and fuzzy inference system: An embedded approach to model and generate trading rules, Applied Soft Computing 90 (2020) 106169.

[19] K. Michell, W. Kristjanpoller, Generating trading rules on us stock market using strongly typed genetic programming, Soft Computing 24 (5) (2020) 3257–3274.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, nature 518 (7540) (2015) 529–533.

[21] Y. Deng, F. Bao, Y. Kong, Z. Ren, Q. Dai, Deep direct reinforcement learning for financial signal representation and trading, IEEE transactions on neural networks and learning systems 28 (3) (2016) 653–664.

[22] Z. Xiong, X.-Y. Liu, S. Zhong, H. Yang, A. Walid, Practical deep reinforcement learning approach for stock trading, arXiv preprint arXiv:1811.07522 (2018).

[23] Y. Li, P. Ni, V. Chang, Application of deep reinforcement learning in stock trading strategies and stock forecasting, Computing (2019) 1–18.

[24] S. Luo, X. Lin, Z. Zheng, A novel cnn-ddpg based ai-trader: Performance and roles in business operations, Transportation Research Part E: Logistics and Transportation Review 131 (2019) 68–79.

[25] K. S. Zarkias, N. Passalis, A. Tsantekidis, A. Tefas, Deep reinforcement learning for financial trading using price trailing, in: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 3067–3071.

[26] Z. Zhang, S. Zohren, S. Roberts, Deep reinforcement learning for trading, The Journal of Financial Data Science 2 (2) (2020) 25–40.

[27] T. Théate, D. Ernst, An application of deep reinforcement learning to algorithmic trading, arXiv preprint arXiv:2004.06627 (2020).

[28] X. Wu, H. Chen, J. Wang, L. Troiano, V. Loia, H. Fujita, Adaptive stock trading strategies with deep reinforcement learning methods, Information Sciences (2020).

[29] A. Suchaimanacharoen, T. Kasetkasem, S. Marukatat, I. Kumazawa, P. Chavalit, Empowered pg in forex trading, in: 2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), IEEE, 2020, pp. 316–319.

[30] K. Lei, B. Zhang, Y. Li, M. Yang, Y. Shen, Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading, Expert Systems with Applications 140 (2020) 112872.

[31] A. Northcott, The complete guide to using candlestick charting: How to earn high rates of return-safely, Atlantic Publishing Company, 2009.

[32] G. Hu, Y. Hu, K. Yang, Z. Yu, F. Sung, Z. Zhang, F. Xie, J. Liu, N. Robertson, T. Hospedales, et al., Deep stock representation learning: From candlestick charts to investment decisions, in: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2018, pp. 2706–2710.

[33] S. Thammakesorn, O. Sornil, Generating trading strategies based on candlestick chart pattern characteristics, in: Journal of Physics: Conference Series, Vol. 1195, IOP Publishing, 2019, p. 012008.

[34] I. Orquín-Serrano, Predictive power of adaptive candlestick patterns in forex market. eurusd case, Mathematics 8 (5) (2020) 802.

[35] S. Birogul, G. Temür, U. Kose, Yolo object recognition algorithm and "buy-sell decision" model over 2d candlestick charts, IEEE Access 8 (2020) 91894–91915.

[36] D. Fengqian, L. Chao, An adaptive financial trading system using deep reinforcement learning with candlestick decomposing features, IEEE Access 8 (2020) 63666–63678.

[37] G. Keller, R. Litchfield, Candlestick charting explained (2006).

[38] G. A. Rummery, M. Niranjan, On-line Q-learning using connectionist systems, Vol. 37, University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[39] R. S. Sutton, A. G. Barto, et al., Introduction to reinforcement learning, Vol. 135, MIT press Cambridge, 1998.

[40] J. Moody, L. Wu, Y. Liao, M. Saffell, Performance functions and reinforcement learning for trading systems and portfolios, Journal of Forecasting 17 (5-6) (1998) 441–470.

[41] W. F. Sharpe, The sharpe ratio, Journal of portfolio management 21 (1) (1994) 49–58.

Appendix A. Rules by the traditional rule-based agent

In this section, we briefly discuss the details of the rule-based pattern extraction for candlestick charts, and also trading signals for each rule. These patterns along with their rules of extraction and their shapes are represented in (A.8). For each rule, we need to use some predefined hyper-parameters to specify them accurately. The value of these hyper-parameters are chosen according to the $\%Rate\ of\ Return$ of the rule-based method.

- GSL(Gap Significance Level): This is a number between 0 and 1, showing the significance level of the gap (difference between the opening and closing price of 2 consecutive candles). It is multiplied by the length of the candle with largest body of the two candles whose gap length needs to be evaluated.
- CSL(Candle Significance Level): This is a number between 0 and 1, showing if a candle length is significant enough in order to be part of a pattern. This number is multiplied by the maximum-body-length candle in a dataset. Then, the length of the candles in a specific pattern is compared with the max-body-length candle to see if its length is significant enough(not too small).

- PSH(Percentage of Shadow Hammer): This parameter shows that the upper shadow of the hammer should be at most what percentage of the total length.
- UBHL(Upper Bound Hammer Length) & LBHL(Lower Bound Hammer Length): These two parameters show the boundaries that the body of the hammer should be inside them in order to be considered a hammer.

We need to declare some functions for simple calculations on the candle sticks. These functions are defined in (A.7). For each pattern, to produce correct signals, the trend of the stock before that pattern occurs should be calculated. In (A.1) $\psi_t(P_t)$ shows the trading signal at time step t. This function is used in $SignalingFunction$ of (A.8). In (A.1), MT is defined in (2).

$$\psi_t(P_t) = \begin{cases} \text{'Buy'} & \text{if MT is downtrend} \\ \text{'None'}, & \text{if MT is side} \\ \text{'Sell'} & \text{otherwise} \end{cases} \tag{A.1}$$

Table A.7: A list of defined function used in the process of pattern extraction

Number	Function Name	Definition	Return Value	Candle Shape
1	$IsBull(P)$	Is candle P bullish or not	$\begin{cases} \text{'True'} & \text{if } P_{close} > P_{open} \\ \text{'False'} & \text{otherwise} \end{cases}$	
2	$IsBear(P)$	Is candle P bearish or not	$\begin{cases} \text{'True'} & \text{if } P_{open} > P_{close} \\ \text{'False'} & \text{otherwise} \end{cases}$	
3	$TL(P)$	Total length of candle P	$P_{high} - P_{low}$	
4	$BL(P)$	Body length of candle P	$\lvert P_{close} - P_{open} \rvert$	
5	$USL(P)$	Upper shadow length of candle P	$\begin{cases} P_{high} - P_{close} & \text{if } IsBull(P) \\ P_{high} - P_{open} & \text{if } IsBear(P) \end{cases}$	
6	$LSL(P)$	Lower shadow length of candle P	$\begin{cases} P_{open} - P_{low} & \text{if } IsBull(P) \\ P_{close} - P_{low} & \text{if } IsBear(P) \end{cases}$	
7	$IsLS(P)$	Is length of P significant	$BL(P) \geq CSL \times \max_{i \in [0 \ldots n]} P_i$	
8	$MidPoint(P)$	Middle point of P	$\frac{P_{close}+P_{open}}{2}$	
9	$IsDoji(P)$	Is P doji	$\begin{cases} \text{'True'} & \text{if } P_{close} \approx P_{open} \\ \text{'False'} & \text{otherwise} \end{cases}$	
10	$GS(P_1, P_2)$	Gap significance	$GSL \times max(BL(P_1), BL(P_2)))$	

Table A.8: A list of used trading rules based on popular candlestick patterns [37]

Number	Name	Definition	Chart	Signaling Function
1	Hammer / Inverse Hammer	$IsBull(P)$; $\begin{cases}(P_{high} - P_{close}) \leq PSH \times TL(P); & \text{If P is Hammer}\\ (P_{open} - P_{low}) \leq PSH \times TL(P); & \text{If P is Inverse Hammer}\end{cases}$ $LBHL \times TL(P) \leq BL(P) \leq UBHL \times TL(P)$;		buy if downtrend
2	Hanging Man / Shooting Star	$IsBear(P)$; $\begin{cases}(P_{high} - P_{open}) \leq PSH \times TL(P); & \text{If P is Hanging man}\\ (P_{close} - P_{low}) \leq PSH \times TL(P); & \text{If P is Shooting star}\end{cases}$ $LBHL \times TL(P) \leq BL(P) \leq UBHL \times TL(P)$;		sell in uptrend
3	Bullish Engulfing	$IsLS(P_2)$ $P_{2,open} \leq P_{1,close} \leq P_{2,close}$ $P_{2,open} \leq P_{1,open} \leq P_{2,close}$		buy if downtrend
4	Bearish Engulfing	$IsLS(P_2)$ $P_{2,close} \leq P_{1,close} \leq P_{2,open}$ $P_{2,close} \leq P_{1,open} \leq P_{2,open}$		sell if uptrend
5	Bullish Harami	$IsLS(P_1)$; $IsBear(P_1)$; $IsBull(P_2)$ $P_{2,close} \leq P_{1,open}$ $P_{2,open} - P_{1,close} \geq GSL \times BL(P_1)$		buy if downtrend
6	Bearish Harami	$IsLS(P_1)$; $IsBull(P_1)$; $IsBear(P_2)$ $P_{2,close} \geq P_{1,open}$ $P_{1,close} - P_{2,open} \geq GSL \times BL(P_1)$		sell if uptrend
7	Piercing Line	$IsLS(P_1)$; $IsLS(P_2)$; $IsBear(P_1)$; $IsBull(P_2)$ $GS(P_1, P_2) \leq (P_{1,close} - P_{2,open})$ $(P_{2,close} \geq MidPoint(P_1))$		buy if downtrend
8	Dark Cloud Cover	$IsLS(P_1)$; $IsLS(P_2)$; $IsBull(P_1)$; $IsBear(P_2)$ $GS(P_1, P_2) \leq (P_{2,open} - P_{1,close})$ $(P_{2,close} \leq MidPoint(P_1))$		sell if uptrend
9	Morning Star	$IsLS(P_1)$; $IsLS(P_3)$; $IsBear(P_1)$; $IsDoji(P_2)$; $IsBull(P_3)$ $(P_{2,close} \leq P_{3,open})$ $(P_{2,close} \leq P_{1,close})$		buy if downtrend
10	Evening Star	$IsLS(P_1)$; $IsLS(P_3)$; $IsBull(P_1)$; $IsDoji(P_2)$; $IsBear(P_3)$ $(P_{2,close} \geq P_{3,open})$ $(P_{2,close} \geq P_{1,close})$		sell if uptrend

Number	Name	Definition	Chart	Signaling Function
11	Three White Soldiers	$IsLS(P_1);\ IsLS(P_2);\ IsLS(P_3)$ $IsBull(P_1);\ IsBull(P_2);\ IsBull(P_3)$		buy if downtrend
12	Three Black Crows	$IsLS(P_1);\ IsLS(P_2);\ IsLS(P_3)$ $IsBear(P_1);\ IsBear(P_2);\ IsBear(P_3)$		sell if uptrend
13	Rising Three Methods	$IsBull(P_1);\ IsBull(P_5);\ IsBear(P_2);\ IsBear(P_3);\ IsBear(P_4)$ $IsLS(P_1);\ IsLS(P_2);\ IsLS(P_3);\ IsLS(P_4);\ IsLS(P_5)$ $max(P_{2,open}, P_{3,open}, P_{4,open}) \leq P_{5,high}$ $min(P_{2,close}, P_{3,close}, P_{4,close}) \geq P_{1,low}$		none (indecision)
14	Falling Three Methods	$IsBear(P_1);\ IsBear(P_5);\ IsBull(P_2);\ IsBull(P_3);\ IsBull(P_4)$ $IsLS(P_1);\ IsLS(P_2);\ IsLS(P_3);\ IsLS(P_4);\ IsLS(P_5)$ $max(P_{2,close}, P_{3,close}, P_{4,close}) \leq P_{5,high}$ $min(P_{2,open}, P_{3,open}, P_{4,open}) \geq P_{1,low}$		none (indecision)