Journal of Artificial Intelligence and Data Mining (JAIDM), Vol. 11, No. 1, 2023, 103-118.

Shahrood University of Technology

Journal of Artificial Intelligence and Data Mining (JAIDM)
Journal homepage: http://jad.shahroodut.ac.ir

Research paper

A Reinforcement Learning-based Encoder-Decoder Framework for Learning Stock Trading Rules

Mehran Taghian, Ahmad Asadi, and Reza Safabakhsh*

Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran.

Article Info

Article History:
Received 09 June 2022
Revised 05 November 2022
Accepted 07 January 2023

DOI:10.22044/jadm.2023.11979.2347

Keywords:
Deep Reinforcement Learning, Deep Q-Learning, Single Stock Trading, Portfolio Management, Encoder-Decoder Framework.

**Corresponding author: safa@aut.ac.ir (R. Safabakhsh).*

Abstract

The quality of the extracted features from a long-term sequence of raw prices of the instruments greatly affects the performance of the trading rules learned by the machine learning models. Employing a neural encoder-decoder structure to extract informative features from complex input time-series has proved very effective in other popular tasks like neural machine translation and video captioning. In this paper, a novel end-to-end model based on the neural encoder-decoder framework combined with DRL is proposed to learn single instrument trading strategies from a long sequence of raw prices of the instrument. In addition, the effects of different structures for the encoder and various forms of the input sequences on the performance of the learned strategies are investigated. The experimental results show that the proposed model outperforms other state-of-the-art models in highly dynamic environments.

1. Introduction

Forming profitable trading strategies fitted on either a single financial instrument or a set of instruments in a specific market based on a vast historical data is a critical problem for investors. Since the introduction of algorithmic trading [1] and monitoring the trading process by computers, especially at high frequency [2], there has been a widespread interest in designing a powerful model to learn profitable investment strategies.

In the recent years, the machine learning (ML) models and deep neural networks (DNNs) have been widely used for learning profitable investment strategies in both single asset trading and portfolio management problems [3].

Considering the great performance of deep reinforcement learning (DRL) models (deep neural networks trained with the reinforcement learning techniques) in forming investment strategies, the proposed methods for portfolio management are mainly based on the DNN structures and DRL techniques. Ganesh *et al*. [4] have proposed a very-long short-term memory (VLSTM) network to deal with extremely long sequences in financial markets, exploring the importance of VLSTM in the context of high-frequency trading. Arévalo *et al*. [5] have proposed a DNN structure to forecast the next one-minute average price of an instrument given its current time and n-lagged one-minute pseudo-returns to build a trading strategy that buys (sells) when the next predicted average price is above (below) the last closing price. Dixon *et al*. [6] have proposed a DNN model to learn the spatio-temporal model of the input, developing a classifier trained by out-of-sample predictive mean squared error to predict short-term market prices. In our former work [7], the DRL performance in learning single asset-specific trading rules was investigated, and conclusions were that: 1) the quality of extracted features from the input can greatly affect the performance of the learned strategy by DRL models, and 2) proposing a good feature extractor from a long-term historical price data sequence would obviously improve the profitability of the resulting trading strategy. Considering the results from our former work [7], proposing a model to learn good features from a long-term price sequence would

effectively contribute to improve the performance of DRL models.

In this work, we first develop a DRL agent based on a deep Q-learning algorithm to generate trading signals given a sequence of OHLC prices of each instrument. Then we design and implement an encoder-decoder based model to improve the agent's feature extraction performance. In addition, we examine the performance of different DNN structures for the encoder module. The time-series of candlesticks and raw OHLC input types are evaluated, and the performance of models is tested using various stocks with different behaviors. Furthermore, the influence of window size on the agent's performance for the windowed input type is studied. The experimental results show that our model outperforms the state-of-the-art methods.

In the next section of this work, we briefly review the related works of learning financial asset-specific trading strategies, and discuss the advantages and disadvantages of different categories of the proposed methods. Section 3 discusses the model proposed in this paper. The model consists of an encoder part for feature extraction and a decoder part for decision-making. The details of the architecture of both of these parts are discussed in Section 3. Section 4 provides the experimental results, and the conclusions are provided in Section 5.

2. Related Works

Many researchers proposed methods based on reinforcement learning for determining trading strategies. RL is applied in portfolio management first by Moody *et al*. [8] using recurrent reinforcement learning. Suchaimanacharoen *et al*. [9] first predicts the future prices using a CNN, and then feed the output to a policy gradient model along with historical data to empower trading decisions.

Following the work by Mnih *et al*. [10], which introduced the deep Q-learning model and its successful performance in playing Atari games, the researchers have carried out many kinds of research works to apply DRL methods to the stock market environment. Luo *et al*. [11] proposed a DDPG model with two different convolutional neural network (CNN) function approximators. The input state to the model is 18 different technical indicators converted to multiple channels of 1D images fed to the CNN model. Wang *et al*. [12] proposed a novel RL-based investment strategy consisting of three phases: 1) extracting asset representation from multiple time-series using a Long Short-Term Memory with a History Attention (LSTM-HA) network, 2) modeling the interrelationships among assets as well as the asset price rising prior using a Cross-Asset Attention Network (CAAN), and 3) generating portfolio and giving the investment proportion of each asset according to the output winner scores of the attention network. The three components are optimized end-to-end using a sharp ratio-oriented RL. Xiong *et al*. [13] explored the training power of the deep deterministic policy gradient to learn stock trading strategy. Chakole *et al*. [14] proposed a method using the Q-learning algorithm to find the optimal dynamic trading strategy. They introduced two models varied in their representation of the environment, the first of which represents environment states using a finite set of clusters, and the second of which used the candlesticks themselves as the states of the environment. Théate *et al*. [15] presented a solution to the algorithmic trading problem of generating the trading strategy for single stock based on the DQN algorithm with a Sharpe ratio-oriented manner. Brim *et al*. [16] used co-integrated stock market prices, and incorporated DQN to generate pairs of trading strategy.

Having considered the temporal essence of stock market data, some of the researches have combined the temporal feature extraction power of recurrent neural networks with DRL's decision-making ability. Wu *et al*. [17] applied the Gated Recurrent Unit (GRU) to exploit informative features from raw financial data along with technical indicators to represent stock market conditions more robustly. Then they designed a risk-adjusted reward function using the Sortino ratio proposed by Rollinger *et al*. [18]. Based on the state, action, and reward functions designed, they proposed deep Q-learning and deep deterministic policy gradient for quantitative stock trading. Weng *et al*. [19] applied DRL for portfolio management, and in order to distinguish the critical time when the price changes, they proposed using a three-dimensional attention gating network that gave higher weights on rising moments and assets. Moreover, they applied the XGBoost method to quantify the importance of features and output the three most relevant features from historical data to the model: close price, high price, and low price.

In our former work [7], we studied the performance of strategies based on the candlestick patterns, SARSA(λ) algorithm, and deep Q-learning, and concluded that methods based on deep reinforcement learning could generate more adaptive trading strategies specific to each asset.

The encoder-decoder framework is one of the most popular neural structures that is used to solve complex problems in an end-to-end manner [20].
In this work, we dedicate focus on the following issues:

1. Proposing a model based on the encoder-decoder architecture, where the decoder is a DRL agent trained to generate a trading strategy based on the representation of the market produced by the encoder model. The encoder is a neural network structure responsible for exploiting features from the raw input time-series data and generating feature vector.
2. Showing the importance of extracting time dependencies of the input prices, and proposing different encoder structures are proposed to improve the quality of time dependencies extraction.
3. Investigating the impact of window-size on exploiting important features and generating a proper representation of the market.

The rest of this paper is structured as what follows. First, we introduce the details of the proposed method, deep Q-learning method, model architecture, and DNN models used as encoder. Then performances of different encoder models are evaluated using various methods explained in detail in Section 4. Next, the results of experiments are analyzed.

3. Proposed Model

In this section, we briefly describe the problem formulation, and then drill into different proposed model structures.

3.1. Formulation

In the financial markets, a candlestick is used to represent the price fluctuations in a short time period, originating from Japanese rice traders and merchants who used candlesticks to track the market prices [21]. A candlestick consists of 4-price elements, namely high (the highest stock price during a period-e.g. a day), low (the lowest price), open (the stock price at the beginning of the period), and close (the stock price at the end of the period), abbreviated to OHLC. A candlestick's color can be either green/white, representing a bullish candle (open price is lower than close price), or red/black representing a bearish candle (close price is lower than open price). Figure 1 shows a sample candlestick. Equation (1) shows a vector representing a candlestick. This vector consists of the Open, High, Low, and Close prices.

$$C_t = (P_{open}, P_{high}, P_{low}, P_{close}) \quad (1)$$

A candlestick chart is used to demonstrate the behavior of the asset price. According to this concept, the patterns in this chart show the buyers' and sellers' behavior and their influence on the market. Thus these patterns can be used to analyze the price fluctuations and use the analysis to devise trade strategies on a financial asset.

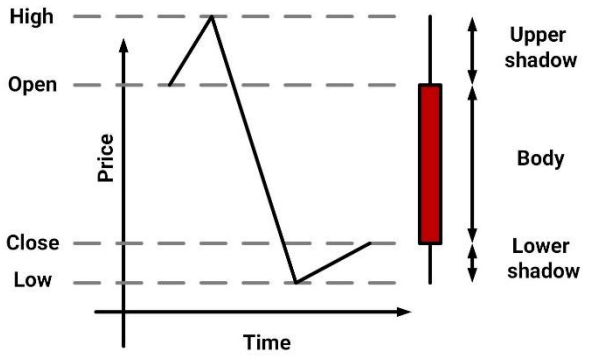


Figure 1. A candlestick representing the price behavior of an asset during a specific time [7].

3.2. Model architecture

The proposed model is based on the encoder-decoder framework, which consists of the following modules:

1. Encoder:
   Encoder is the first neural structure that takes the input of the model and learns a good mapping from the input space to the feature space that minimizes the decoders' loss function.
2. Decoder:
   Decoder is the second neural structure in the encoder-decoder framework that takes the features extracted by the encoder for each input record and generates the appropriate output based on the input feature vector. The gradients of the decoder are back-propagated to the encoder and train its weights along with the weights of the decoder during the training phase.

In the proposed model, the decoder part is a target or policy network used in the deep Q-learning based model proposed by Taghian *et al*. [7] to learn the trading strategies. The encoder part is a deep neural network applied to extract deep features from the candlestick chart representations. These features are categorized into the following two groups:

1. Features directly learned from candlestick representations or raw OHLC data.
2. Features representing the temporal relationships among a sequence of candlesticks inside a time window.

For each category, NN models exist to efficiently analyze and extract those features according to the policy network's performance in trading. The encoder part extracts the features from the input, and provides a state vector (feature vector) for the decoder module, a deep Q-Learning agent that uses this state vector as the state of the environment to produce trading signals. Based on the rewards given to the DRL agent, the model is optimized towards producing higher profits. This optimization is done in an end-to-end form, back-propagating error from the decoder part to the encoder module. As a result, the encoder extracts the features based on the trading performance of the DRL. The model architecture is shown in figure 2.

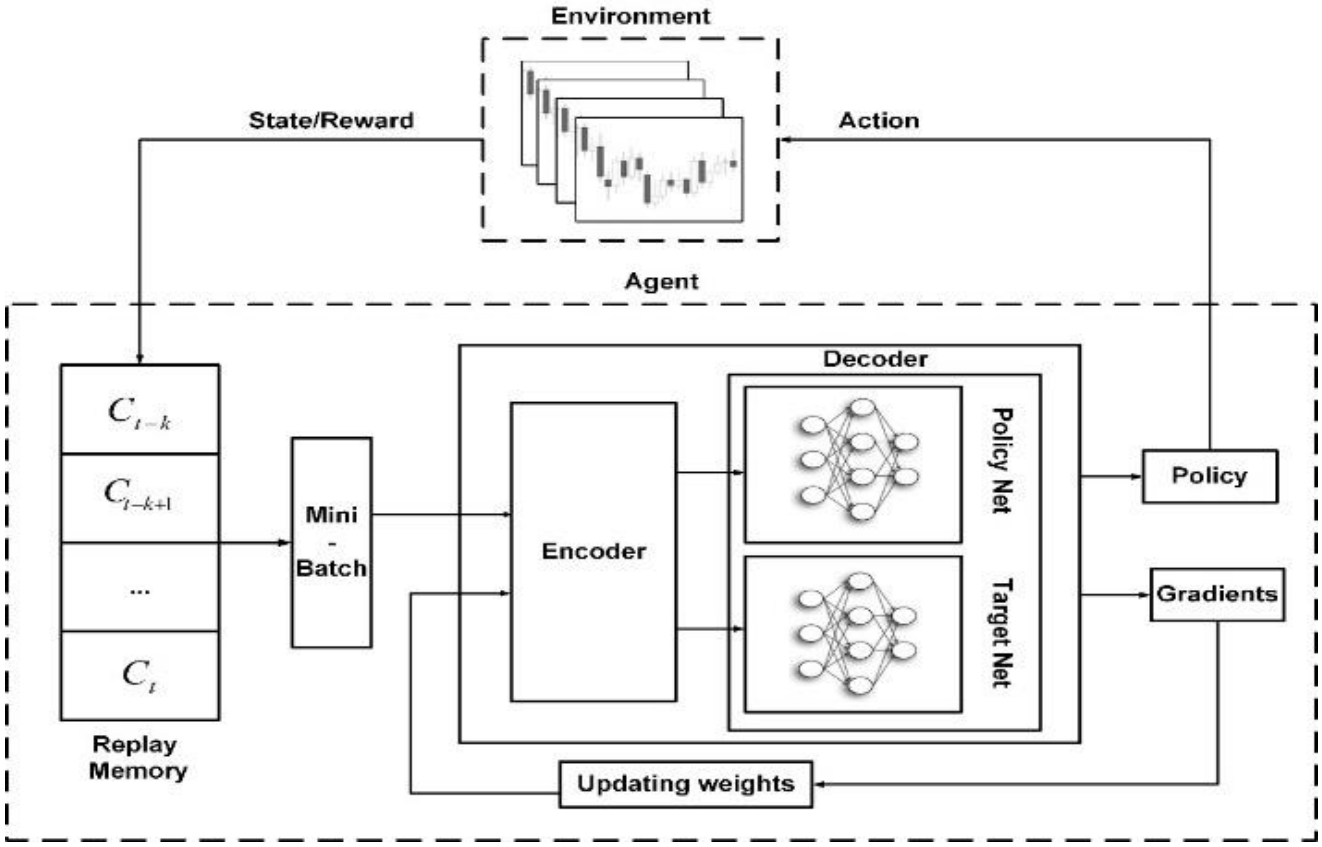


Figure 2. Overall structure of the proposed model.

3.3. The decoder module

The decoder module is the trading agent of our model, and learns to produce trading signals. This module's architecture is based on the deep Q-Learning algorithm proposed by Mnih *et al*. [10] to play Atari games. The module's input is a state vector containing the features of the market at time-step t. These features can be either vanilla input of OHLC prices or the feature vector produced by a feature extractor model. We show the vector space of the input candles with $C = \{c_1, c_2, ..., c_T\}$ where T is the final time-step, and c_i is the candle representation of different time-intervals (here daily). The feature extractor module Φ gets the vector space of candles as input, and generates the vector space of states $S = \{s_1, s_2, ..., s_T\}$.

$$\Phi(C) = S \quad (2)$$

Equation (2) represents Φ, the feature extraction function. This function can be either an identity function (state space is the candlesticks themselves) or a neural network that extracts deep features from the input vector space of candlesticks and produces a vector space of features. Each vector s_i denotes the state of the environment at time-step t. This state is given to the DQN agent to use in taking action. The action space of the DQN agent is $A = \{'buy', 'sell', 'noop'\}$, which are the signals of the trading strategy at each time-step. After taking action, the agent would be given a reward based on the signal produced. The rewards of buying and selling are a bit different. Equation (3) demonstrates the reward function used by the

environment. The os parameter is used alongside action $a='n'$ showing that whether the money has already been invested on the market or not.

$$R_t=\begin{cases}((1-Tc)^2*\frac{p_2}{p_1}-1)*100 & a='b'\vee(a='n'\wedge os=1)\\ ((1-Tc)^2*\frac{p_1}{p_2}-1)*100 & a='s'\vee(a='n'\wedge os=0)\end{cases} \quad (3)$$

Reinforcement learning is a framework used to learn a sequence of decision tasks. In general, the RL agent interacts with the environment, observes the state, takes action according to the policy and the observed state, and gets a reward. In a sequence of decisions made by the RL agent, the agent learns a policy π regarding the actions taken and rewards earned at each episode. Afterward, the agent should optimize its policy to maximize cumulative reward after each episode. For this purpose, we used deep Q-learning, a critic-based reinforcement learning algorithm, which uses action-value function $Q(S,A)$ denoting the expected cumulative reward in state S when action A is taken. More formally, we use a multi-layered perceptron to approximate the optimal action-value function, which is demonstrated in (4).

$$Q^*(s,a)=\max_{\pi}E[r_t+\gamma r_{t+1}+\gamma^2 r_{t+2}+...] \quad (4)$$

Where γ is the discounting factor, r_t is the reward at time-step t, π is the behaviour policy learned, s is the observed state, and a is the agent's action. The optimal action-value function obeys the Bellman equation:

$$Q^*(s,a)=E_{s'}[r+\gamma\max_{a'}Q^*(s',a')] \quad (5)$$

In order to reduce the mean squared error in the Bellman equation, we use two sets of parameters (neural networks). The target values are approximated using the target network weights (from previous iterations) shown with $\overline{\theta_i}$ at iteration i. The policy network, which is being trained in each iteration to adjust its parameters to reduce the mean squared error, is used to approximate the Q function using parameters θ_i at iteration i. Thus we have a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration.

$$L_i(\theta_i)=E[(y-Q(s,a;\theta_i)^2]+E[V_{s'}[y]] \quad (6)$$

In order to further stabilize the deep Q-learning algorithm, we use the Huber loss proposed by [40] instead of the mean squared error, which pays attention to large and small errors equally.

$$Huber(e)=\begin{cases}\frac{1}{2}e^2 & |e|\le 1\\ |e|-\frac{1}{2} & |e|>1\end{cases} \quad (7)$$

Furthermore, our agent stores the last c experiences in the replay memory. The agent's experience vector $e_t=(s_t,a_t,r_t,s_{t+1})$ is saved in the experience replay memory $D_t=\{e_1,e_2,...,e_c\}$ (where c is the length of the replay memory) and used as a batch to optimize the policy network. When the model wants to update, it samples a batch of experiences uniformly at random from D. The steps of the deep Q-learning algorithm used in our work are represented in Algorithm 1.

3.4. Encoder module

So far, we have discussed the input data representation, different parts of the trading agent, and the deep Q-Learning algorithm, used by the agent to optimize its policy to learn devise profitable strategies. However, the essential part of each RL algorithm is the representation of the environment. As mentioned earlier, the environment tells the RL agent in which state it currently is, based on which the agent would take actions and receive rewards from the environment. A proper state representation can significantly improve the performance of the RL agent. Therefore, in this section, we want to concentrate on our model's feature extraction and state representation the encoder module.

Algorithm 1. Deep Q-Learning algorithm used for training the agent.

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 pre-processed sequence $\phi_1=\phi(s_1)$
  For t from 1 to T do
    With probability ϵ select a random action a_t
    Otherwise select $a_t = armax_a Q(\phi(s_t),a;\theta)$
    Execute action a_t and observe reward r_t and state s_{t+1}
    Set $s_{t+1}=s_t$, and pre-process $\phi_{t+1}=\phi(s_{t+1})$
    Store transition $(\phi_t,a_t,r_t,\phi_{t+1})$
    Sample random mini-batch of transition $(\phi_j,a_j,r_j,\phi_{j+1})$ from D
    Set $y_j=\begin{cases}r_j & if\ episode\ terminates\ at\ step\ j+1\\ r_j+\gamma\max_{a'}\hat{Q}(\phi_{j+1},a';\theta) & otherwise\end{cases}$
    Perform a gradient descent step on $\left(y_j-Q(\phi_j,a_j;\theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q}=Q$
  End for
End for

We introduced the φ function, given the input candlesticks, extracts features, and outputs the state space, which is then fed to the decoder module–the DQN model. The φ function can be either an identity function or a deep neural network. We do not have any feature extraction in the first case, and candlesticks are directly fed to the DQN model. The DNNs we use as the feature extractor are Multi-Layered Perceptron (MLP), Gated Recurrent Unit (GRU) proposed by Cho *et al*. for machine translation [22], 1-dimension Convolution in the direction of time (CNN) proposed by LeCun *et al*. as an encoder for machine translation [23], and GRU with 1-dimension convolution in the direction of price (CNN-GRU).

The MLP model can extract features from candlesticks without considering the temporal relationship among candles. In contrast, the other four models not only pay attention to the structure of each candlestick, but they also consider the temporal relationships. In this section, we explain the detailed architecture of each feature extractor. We will compare the performance of these DNNs as the feature extractor for the DQN later.

Before we dive into each model's description, we need to explain different inputs to these models. Our inputs partition into two categories:

1. **Vanilla:**
   The OHLC prices without any change. This kind of input contains only the representation of candle c_t at time-step t.
2. **Windowed:**
   A series of candlesticks with size w are grouped together to form a window of candles $W = \{c_{t-w}, c_{t-w+1}, ..., c_t\}$ at time step t.

All models use the windowed input type but the raw OHLC prices are only for MLP encoder and DQN without any encoder models.

3.4.1. MLP

The MLP model is a NN with only one hidden layer. In order to regularize the outputs of layers, we used Batch normalization after the hidden layer. The dimensions of layers are $InputSize * 128, BatchNormalization(128), 128 * FeatureVectorSize$. The MLP architecture takes both types of inputs. If the raw OHLC is used, then $InputSize = 4$, and in case the input type is windowed, then the $InputSize$ would be equal to the size of the window.

3.4.2. GRU

The Gated Recurrent Unit (GRU) is a recurrent neural network exerted on extracting features from time-series data. This model's input type is the windowed input, which contains a sequence of candles at each time step. The role of GRU here is to extract features from each candlestick, while considering the history in each window. The architecture of the GRU model is represented in figure 3a.

3.4.3. CNN

The Convolutional Neural Networks (CNN) has been widely used in image processing to extract deep features from images, and also it is applied in signal processing for analyzing signals. CNN has a kernel, which can move in one, two or higher directions and extract features from multi-dimensional data. Here, our input is the OHLC prices windowed to form time-series data. Thus we have a 2-dimensional input, the first of which is the price (candles), and the second one is time. The input channel size is the size of each candle vector (i.e. 4 for OHLC) and the kernel size is 3 in the direction of time (i.e. w the window size). This architecture is shown in 3b.

3.4.4. CNN-GRU

The combination of CNN and GRU model is proposed here, where the CNN model's kernel moves in the direction of candles and extracts candlestick features. Then it outputs a sequence of features from the windowed input to the GRU model. The GRU model here is responsible for the input's temporal behavior, where it takes the candlesticks' features in sequence from the CNN, and extracts temporal features. The details of this architecture are represented in figure 3c.

4. Experimental Results

4.1. Dataset

All the models are tested on real-world financial data including stocks and crypto-currencies. Data is chosen to be varied in the behavior like the bullish trend, bearish trend, and side-markets. Furthermore, the data length is chosen to be 20 years with the last five years as test (trading) data, ten years with the last two years as test data, and six years (BTC/USD) with the last two years as test data. The interval of candlesticks in all data is chosen to be daily. All data used in this work is available on Yahoo Finance and Google Finance. The summary of the datasets is represented in Table 1.

Figure 4 shows the condition of each dataset in different periods. The AAL data is bullish on the training-set and bearish on the test-set, market GE is both bearish on the train and test sets, AAPL and GOOGL are both bullish; KSS and HSI are

examples of volatile markets, and BTC/USD is side on the test-set. These datasets are selected to measure the flexibility of different models in different market conditions. A robust model can generalize its performance to provide a proper strategy behaving profitable on the test-set.

4.2. Evaluation Metrics

The trading strategy proposed by each model is evaluated from three perspectives:

1. How profitable is the proposed strategy.
2. What is the risk of the proposed strategy.
3. The effect of hyper-parameters (e.g. window size) in proposing a strategy for each asset.

The metrics are mentioned and described in detail as follows**.**

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

Dataset	Start date	Split point	End date
GOOGL	2010/01/01	2018/01/01	2020/08/25
AAPL	2010/01/01	2018/01/01	2020/08/25
AAL	2010/01/01	2018/01/01	2020/08/25
BTC-USD	2014/09/17	2018/01/01	2020/08/26
KSS	1999/01/01	2018/01/01	2020/08/24
GE	2000/01/01	2015/01/01	2020/08/24
HSI	2000/01/01	2015/01/01	2020/08/24

4.2.1. Profit curve

This is a qualitative metric showing the percentage of profit concerning the initial investment. At each point of time t, if the current wealth W_t and the initial investment is W_0 then the percentage of the profit at each time step is calculated using (8).

$$Rate_t = \frac{w_t - w_0}{w_0} * 100 \tag{8}$$

The profit curve compares the $Rate$ of profit for each model at different time steps.

4.2.2. Arithmetic return

This metric is the sum of the rate of increase or decrease in the current investment due to the decisions made by the model (Buy, Sell, None). The rate of wealth change at the current time-step if the model has already invested the money (not sold before) is as in (9).

$$AR_t = \frac{w_t - w_{t-1}}{w_{t-1}} \tag{9}$$

Using (9), we can calculate the arithmetic return in (10).

$$AR = \sum_{t=1}^{T} AR_t \tag{10}$$

Which shows the cumulative return at each time step.

4.2.3. Time Weighted Return

The amount of return in different periods are not independent of each other. In other words, when the amount of loss is significant at one time, then the capital would be significantly lower to invest afterward. For this purpose, we use Time Weighted Return (TWR) which is calculated in (11).

$$TWR = (\prod_{i=1}^{n} (x_i + 1))^{\frac{1}{n}} - 1 \tag{11}$$

To avoid negative values, we add 1 to all the return values, then we remove 1 from the result.

4.2.4. Daily Return Variance

This metric is the variance of daily arithmetic returns.

$$RV = \frac{\sum_{t=1}^{T} (AR_t - AR)^2}{T - 1} \tag{12}$$

Where AR is the average arithmetic return and AR_t is the arithmetic return at time t.

4.2.5. Total return

It is the percentage of the increase in the capital during trading time. Total Return is calculated in (13) where W_0 and W_t are the initial and final wealth, respectively.

$$TR = \frac{w_t - w_0}{w_0} \tag{13}$$

4.2.6. Value at risk

The value at risk (VaR) is a metric to measure the quality level of financial risk within a portfolio during a specific period of time. VaR typically is measured with a confidence ratio $1-\alpha$ (e.g., with a confidence level of 95% where $\alpha = 5$) and measures the maximum amount of loss in the worst situation with confidence $1-\alpha$ in the corresponding time period. The higher the value of the VaR_α (i.e., the absolute value of VaR_α) with a fixed value of α, the higher the level of the portfolio's financial risk. There exist 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 futures 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.

4.2.7. Daily return volatility

The volatility of the daily returns evaluates the risk level of trading rules by calculating daily returns' standard deviation. This metric is calculated for each strategy using (14), where AR is the average daily arithmetic return,
and AR is the daily arithmetic return.

$$\sigma_p = \sqrt{\frac{\sum_{i=1}^{T}(AR_i - AR)^2}{T-1}} \tag{14}$$

4.2.8. Sharpe ratio

The Sharpe ratio (SR) was proposed first by Sharpe *et al*. [24] 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 (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 $R_f = 0$.

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

4.2.9. Window size heat-map

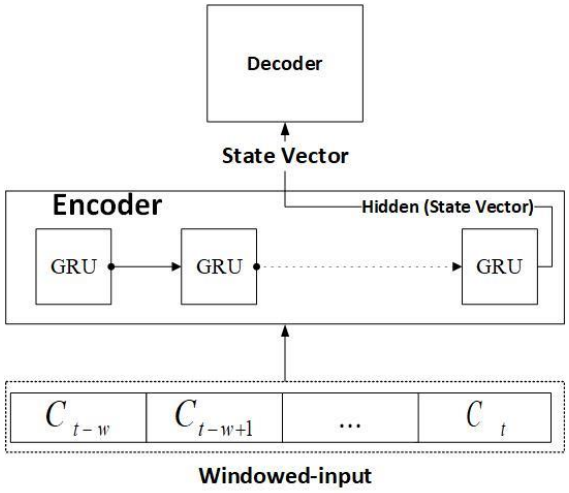
This diagram illustrates the impact of window-size in extracting appropriate patterns from the input candlesticks for each asset, which is reflected as the total profit earned by the agent corresponding to each window size.

4.2.10. Decision curve

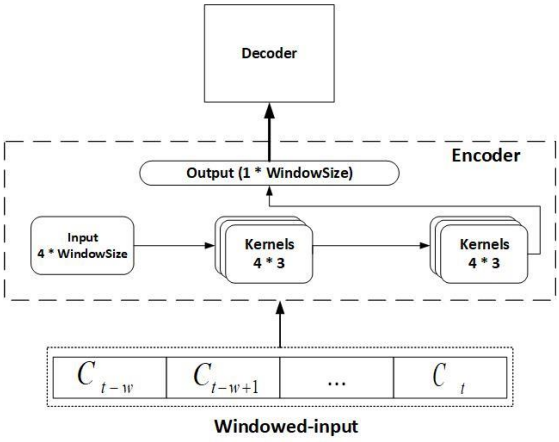
In this curve, the trading signals to trade each asset are demonstrated over that asset's raw price curve. This chart gives insight into the quality of decision-making power of each model on each financial asset.

4.3. Experimental setup

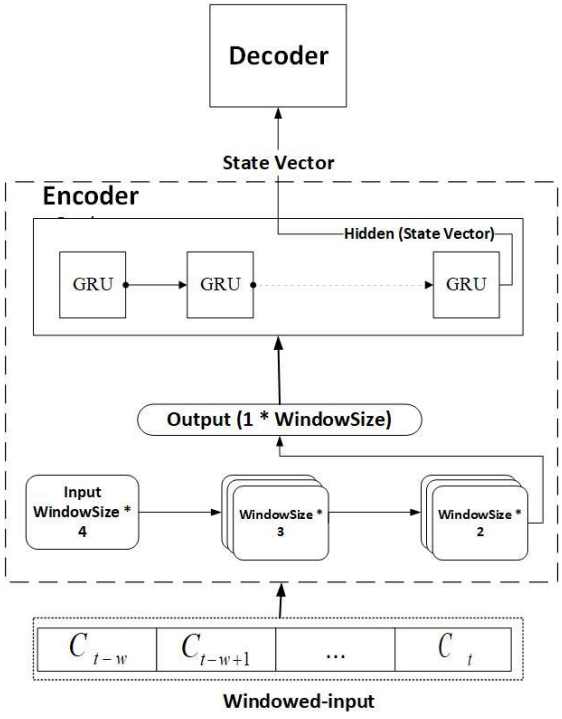
All the models are implemented using *Pytorch* library in Python. In order to optimize the models, we used Adam optimizer. The mini-batch training is also conducted using a batch size of 10, and the replay memory size is set to 20.



(a) GRU model architecture



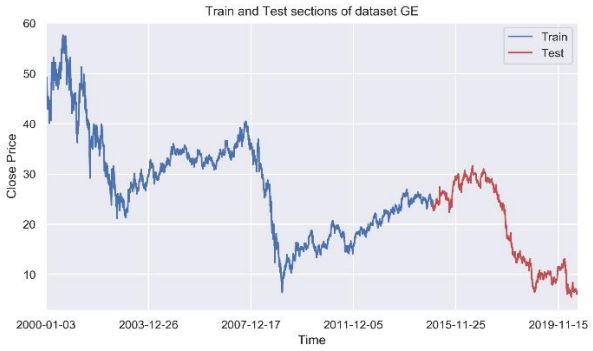
(b) CNN model architecture



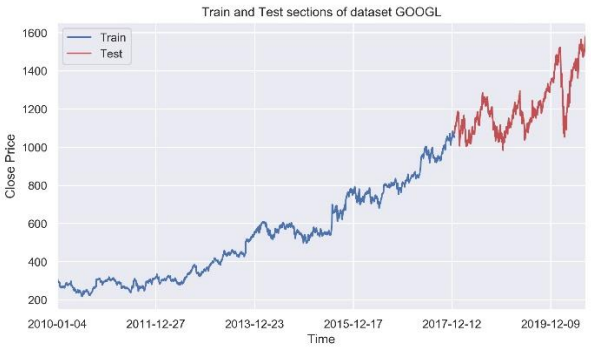
(c) CNN-GRU model architecture

Figure 3. Architecture of different models proposed to use as the encoder.

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

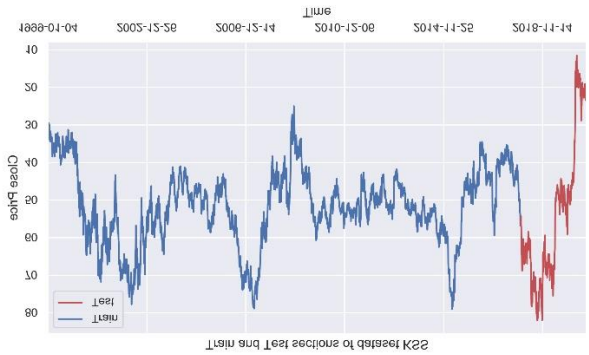


(b) Price history of GE stock used to train and test the model.

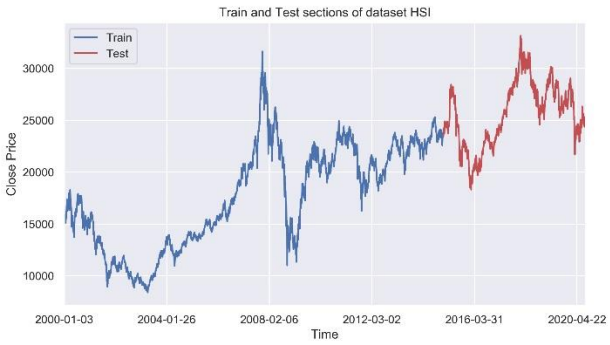


(c) Price history of GOOGL stock used to train and test the model.

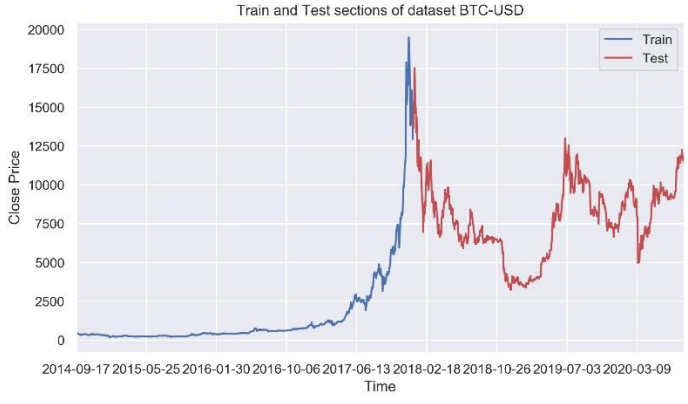
(d) Price history of AAPL stock used to train and test the model.



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



(f) Price history of HSI stock used to train and test the model.



SS

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

The only regularization used in the experiments in the policy and target networks is the *Batch Normalization*. The transaction cost is set to zero during the training process; however, it may be non-zero during the evaluation.

4.4. Performance evaluation of models

In this section, the overall performance of the different models, along with different input types for the MLP and DQN models are compared.

Figure 5 illustrates the profit curves of models on the test set for different datasets. DQN-vanilla is the DQN model without any encoder and with the input of raw OHLC. DQN-windowed and MLP-windowed are the same as DQN-vanilla and DQN windowed, except that MLP contains an encoder part, which is an MLP model. CNN, GRU, and CNN-GRU are models with the encoder part as described in sections 3.4.3, 3.4.2, and 3.4.4, respectively, with input type as a window of candles (time series).

The general conclusions we reached from the experiments are reported in 5:

- Stocks can be categorized into two kinds: the one in which the sequence of candlesticks have effective temporal relationships and those with few meaningful time dependencies.
- The most profitable trading strategies for data with a high level of time dependency in their price history can be generated using windowed-input models. BTC/USD, GOOGL, AAPL, and GE are of this kind. The GRU, and CNN have the best performance on the BTC/USD model; The CNN, DQN-windowed, and MLP-windowed have the best performance on GE; The GRU, CNN-GRU, and MLP-windowed have the best performance on GOOGL; MLP-windowed, GRU, DQN-windowed, and CNN provided the most profitable strategies for AAPL.
- On the other hand, we have data with a low level of dependency in time among candlesticks, which leads to the models with raw OHLC input having a better performance. AAL, HSI, and KSS are among this type of data. By low level of dependency in time, we do not mean that models with time-series inputs have poor performance. Their performance is very good, but they behave a little poorly compared to models with raw OHLC input.

Figure 2 represents the details of experiments with regards to both profit and risk. One crucial point that can be inferred from the results is that as the models' total return increases, the Sharpe ratio increases correspondingly. It means that the models devise strategies in a risk-adjusted way. However, if we want to examine the results in specific, on some data, the models with the highest profitability acted riskily. For example, in AAL, the best model in total return is MLP-vanilla but the Sharpe ratio and VaR (here we consider the absolute value of VaR) of its strategy are, respectively, lower and higher than those strategies proposed by windowed-input models. The same is true about GRU in BTC/USD, where GRU has the highest total return but CNN provided more risk-adjusted strategies with respect to VaR and Sharpe ratio.

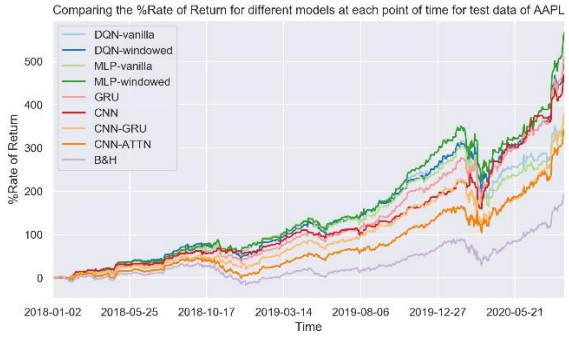
Another important conclusion deducted from comparing data diagrams and the models' performance is that stocks with highly volatile prices such as KSS, AAL, and HSI can best be processed by models with raw OHLC inputs, whereas other data with more stable prices are best analyzed with windowed-input models. Therefore, in order to select among feature extractors, we should pay careful attention to the type of input data. The best feature extractor for stable stocks would be windowed-input models, whereas, for highly volatile stocks, models with raw OHLC can decide and change their behavior more quickly since they only pay attention to the current candlestick, not the history of candles.

4.5. Impact of window size

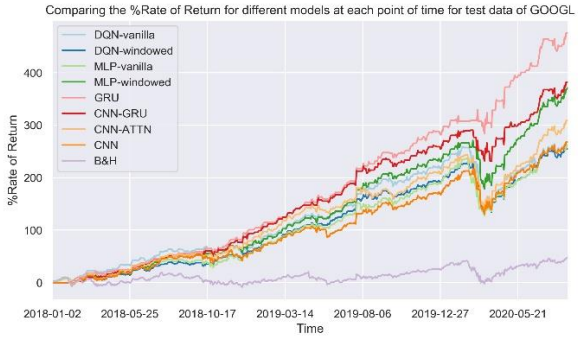
Now that we have examined the performance of different feature extractor models, it is time to dive deeper into the temporal feature extractor concept. Feature extractors with windowed inputs can perform better on data with more stable price movements (rather than highly volatile data). Moreover, considering 2 and 5, each windowed-input model has its best performance varying from data to data. We want to inspect the impact of window size for each model differently using the data in which the model has its best performance.

We test the performance of GRU and CNN-GRU using GOOGL; CNN, MLP-windowed, and DQN-windowed using GE. Figure 6 demonstrates a heat-map showing the relationship between the window size and the normalized total profit earned by the models with windowed input. Window sizes vary from 3 to 75, and the total profit is normalized to bring between 0 and 1.

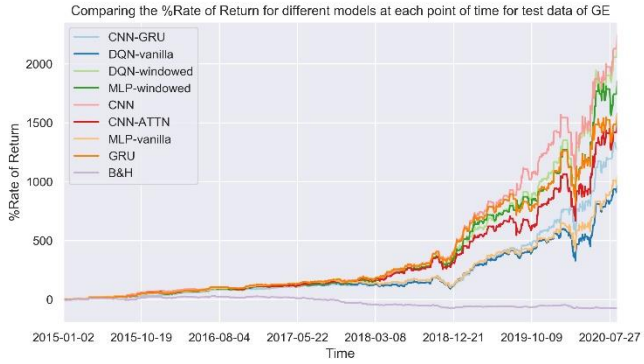
Blocks having lighter colors earned higher profit than those with darker colors. As obvious from the heat-map, the number of lighter colors in the interval of 10 to 20 is more than other window sizes. Therefore, the best feature extraction using a sequence of candlesticks can be done with window size between 10 and 20.



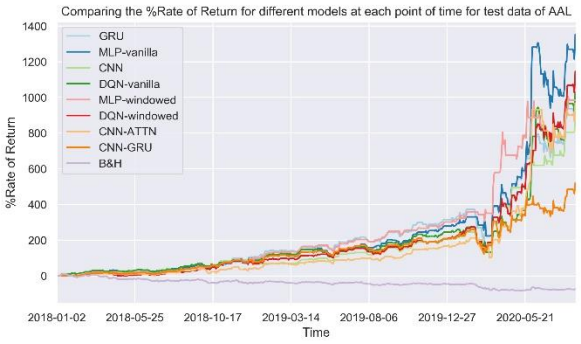
(a) Performance of different models on AAPL



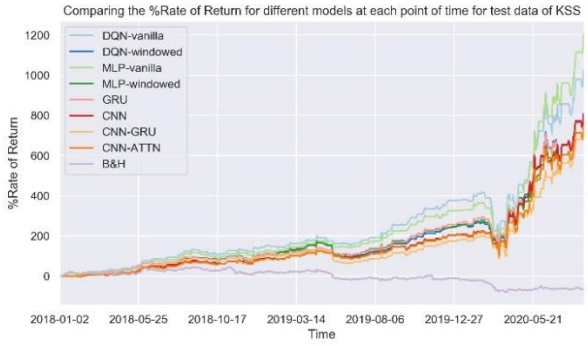
(b) Performance of different models on GOOGL



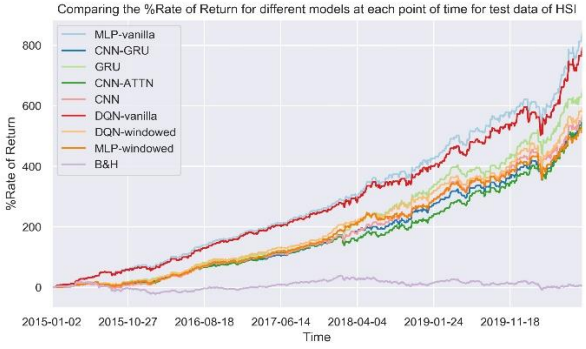
(c) Performance of different models on GE.



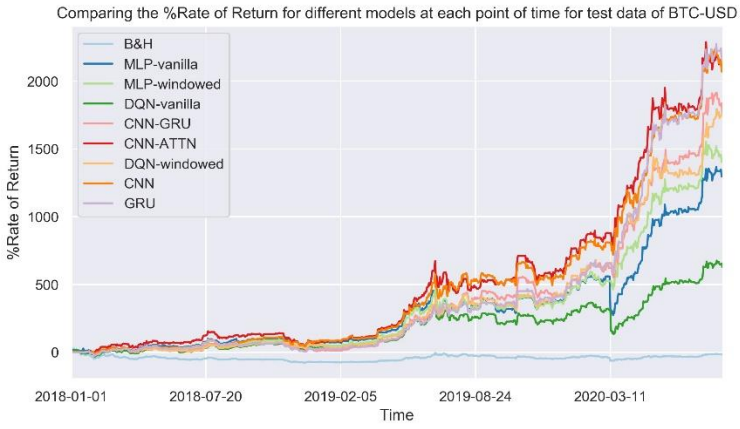
(d) Performance of different models on AAL.



(e) Performance of different models on KSS.



(f) Performance of different models on his.



(g) Performance of different models on BTC/USD.

Figure 4. Profit curve of the models different from the viewpoint of encoder part.

4.6. Sample signaling

For each data, the trading strategy is illustrated in figure 7 based on the decisions made at each time step by the most profitable model. The green, red, and blue points represent the 'buy', 'sell', and 'none' signals, respectively. When the agent generates a signal, it will influence the next day's investment. In other words, when the agent decides to buy a share, this action is actually done the next day. As mentioned earlier, we use a parameter $OwnShare$, which tells us whether the agent already bought the share or not. Thus when the agent bought a share at the time step t, the

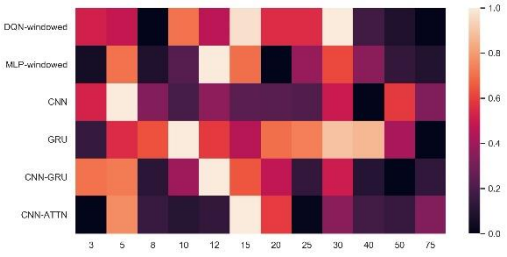


Figure 5. The heat-map generated to show the impact of different window sizes on the feature.

$OwnShare$ parameter would become true, and if the next action is none, the agent's money will continue to be invested. We begin by an initial investment at t_0, and at each time-step, when the agent decides to buy or sell, all the money would be invested or withdrawn. As shown in figure 7, agents could generate signals properly in positions where the trend of the market changes. In order to represent the strategy behavior for each data, we select a period of 100 intervals. The stable markets such as GE, GOOGL, and AAPL contain 'none' signals in their strategy more than volatile markets such as HSI and KSS. That can explain the fact that in stable markets, the market trend changes less rapidly than highly volatile markets; therefore, agents can produce more 'none' action in their strategy.

4.7. Comparing models with similar works

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

1. **Buy and Hold (B&H)**
   B&H is one of the most widely used benchmark strategies to compare the performance of a model. In this strategy, the investor selects an asset and buys it at the first time step of the investment. The purchased asset is held to the end of the period regardless of its price fluctuations.
2. **GDQN**
   Proposed by Wu *et al*. [17], uses the concatenation of the technical indicators and raw OHLC price data of the last nine time steps as the input, a two-layered stacked structure of GRUs as the feature extractor, and the DQN as the decision-making module.
3. **DQT**
   Proposed by Wang *et al*. [25], implements 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 by computing the accumulated wealth over the last n days.
4. **DDPG**
   Proposed by Xiong *et al*. [14] uses Deep Deterministic Policy Gradient (DDPG) as the deep reinforcement leaning approach to obtain an adaptive trading strategy. Then the model's performance is evaluated and compared with the Dow Jones Industrial Average and the traditional min-variance portfolio allocation strategy**.**

Tables 2, 3, and 4 represent our models' performance along with the state-of-the-art models using the profit metrics. According to the results reported in table 2 and table 3, the performance of the model with MLP encoder and raw OHLC input is significantly better than DQT and RRL on stocks HSI and S&P500 proposed by Wang et al. [25]. For HSI, time-series models achieve a performance close to MLP-vanilla, but they behave poorly on S&P500.

Table 5 represents the rate of return (%) for our models with different encoders and models proposed by Wu *et al*. [17]. Wu *et al*.'s best model performance is on AAPL stock with rate of return equal to 77.7 but the GRU model gains the rate of return 438, which is significantly better.

Moreover, wherever the models proposed by Wu *et al*. got a negative return, our model returns a highly positive return. Consider stock GE, where the maximum return value in Wu *et al*. is -6.39% but the best strategy proposed by MLP with vanilla input is 130.4%. When examining the returns gained by different models on IBM, it is obvious that return values for time-series models are better than those with raw OHLC input, and the GRU encoder gains the highest return of 174%. This concept explains the existence of a temporal relationship in IBM stock in that specific period.

Table 5 shows the performance of DDPG, the model presented by Xiong et al. [23].

(a) Trading strategy generated for AAL.

(b) Trading strategy generated for GE.

(c) Trading strategy generated for GOOGL.

(d) Trading strategy generated for AAPL.

(e) Trading strategy generated for KSS.

(f) Trading strategy generated for his.

(g) Trading strategy generated for BTC/USD.

Figure 6. Histogram of strategies generated on each dataset for a period of time.

The final portfolio value of models in our work is better than DDPG, starting with an initial portfolio value of 10000. The CNN-GRU has the best performance with a final portfolio value of 21984, while the DDPG model's final portfolio value is 19791.

As the results indicate, our models perform significantly better than similar models in profitability, ranging from time-series models to raw OHLC inputs. As previously mentioned, these papers' codes were not available, and we had to compare the performance according to common metrics.

5. Conclusion

In this work, we proposed a method based on the encoder-decoder framework, where the encoder model is a DNN, which helps extract essential features from the raw financial data, and the decoder is a DRL agent which makes a decision at each time-step and generates trading signals. The model is trained end-to-end, and the encoder's feature extraction function is optimized toward the policy improvement of the DRL agent.

DRL is based on the deep Q-learning algorithm, and consists of a policy and a target network, both sssof which are multi-layered perceptron networks. For the encoder part, the feature extraction performance of various DNNs is evaluated and compared.

The proposed models for the encoder part are categorized into two types: 1) The raw OHLC input, which receives candle OHLC prices directly. 2) Time-series input, which concatenates a window of consecutive candles and receives the window as input.

Based on the experimental results, the performance of models depends on the market behavior.

Table2. Performance of different models on BTC/USD, GOOGL, AAPL, KSS, and GE.

Agent	Arithmetic Return	Average Daily Return	Daily Return Variance	Time Weighted Return	Total Return (%)	Sharpe Ratio	Value At Risk	Volatility	Final Portfolio Value
BTC/USD									
DQN-vanilla	262	0.27	12.64	0.002	629	0.076	-5.58	110.6	7287
DQN-windowed	334	0.34	8.69	0.003	1757	0.117	-4.51	91.7	18567
MLP-vanilla	324	0.33	12.01	0.003	1296	0.097	-5.37	107.8	13959
MLP-windowed	320	0.33	10.19	0.003	1402	0.104	-4.93	99.3	15021
GRU	359	0.37	9.89	0.003	2158	0.118	-4.81	97.8	22577
CNN	353	0.36	9.42	0.003	2069	0.119	-4.69	95.5	21693
CNN-GRU	338	0.35	9.47	0.003	1770	0.114	-4.72	95.7	18701
GOOGL									
DQN-vanilla	138	0.21	2.59	0.002	263	0.128	-2.45	41.6	3631
DQN-windowed	134	0.20	2.25	0.002	255	0.134	-2.27	38.7	3546
MLP-vanilla	135	0.20	2.60	0.002	252	0.125	-2.45	41.6	3520
MLP-windowed	163	0.24	2.29	0.002	371	0.162	-2.25	39.0	4714
GRU	180	0.27	1.56	0.003	475	0.217	-1.79	32.2	5752
CNN	139	0.21	2.73	0.002	268	0.127	-2.51	42.6	3678
CNN-GRU	163	0.25	1.75	0.002	382	0.185	-1.93	34.1	4819
AAPL									
DQN-vanilla	166	0.25	3.07	0.002	372	0.142	-2.63	45.2	4722
DQN-windowed	190	0.29	3.22	0.003	500	0.159	-2.67	46.3	5997
MLP-vanilla	165	0.25	3.16	0.002	366	0.139	-2.68	45.9	4657
MLP-windowed	200	0.30	3.03	0.003	566	0.172	-2.57	44.9	6658
GRU	191	0.29	2.99	0.003	511	0.166	-2.56	44.6	6112
CNN	181	0.27	2.07	0.003	469	0.189	-2.10	37.1	5688
CNN-GRU	170	0.26	4.03	0.002	379	0.127	-3.05	51.8	4786
KSS									
DQN-vanilla	272	0.41	9.25	0.004	1024	0.134	-4.60	78.5	11236
DQN-windowed	251	0.38	9.38	0.003	809	0.123	-4.67	79.0	9088
MLP-vanilla	287	0.43	9.14	0.004	1205	0.143	-4.55	78.0	13048
MLP-windowed	242	0.36	8.93	0.003	747	0.122	-4.56	77.1	8467
GRU	248	0.37	8.84	0.003	801	0.125	-4.52	76.7	9005
CNN	250	0.37	9.10	0.003	806	0.124	-4.59	77.8	9055
CNN-GRU	242	0.36	9.19	0.003	737	0.120	-4.63	78.3	8369
GE									
DQN-vanilla	260	0.18	3.25	0.002	967	0.101	-2.79	68.0	10673
DQN-windowed	333	0.23	2.87	0.002	2179	0.138	-2.56	63.9	22788
MLP-vanilla	264	0.19	2.87	0.002	1044	0.110	-2.61	63.9	11442
MLP-windowed	317	0.22	2.89	0.002	1848	0.131	-2.58	64.1	19482
GRU	304	0.21	3.12	0.002	1580	0.121	-2.69	66.5	16795
CNN	335	0.24	2.74	0.002	2242	0.142	-2.49	62.3	23416
CNN-GRU	283	0.20	2.91	0.002	1278	0.117	-2.61	64.3	13779

Table 3. Compare profitability performance with Wang et. al. [25] based on Rate of Return (%).

Model name	HSI	S&P
MLP-vanilla	13231.2	5032.3
DQN-vanilla	5016	2524
MLP-windowed	7227	4118
DQN-windowed	7576	4289
GRU	10911	3918
CNN	10575	3859
CNN-GRU	12566	2573
DQT [25]	350	214
RRL [25]	174	141
B&H [25]	154	169

When the market is highly volatile, meaning that the rate of price fluctuation is high, DQN and MLP model with the raw OHLC input had the best performance since they are able to make decisions only based on current input representation, disregarding to the historical changes of the market.

On the other hand, there are more stable markets where models with time-series input can devise more profitable trading strategies because the market behavior enables them to exploit efficient features from financial data history. The window size impact was further studied, and we concluded that window sizes in the interval of 10 to 20 had the best feature extraction performance. Using the trading strategies generated for each data, we understand that agents working on stable stocks will generate none signal more frequently than in strategies devised for highly volatile markets.

Table 4. Compare profitability performance with Wu et. al. [17] based on Rate of Return (%).

Model name	AAPL	GE	AXP	CSCO	IBM
DQN-vanilla	336	129	183	182	144
MLP-vanilla	262	130	260	259	149
DQN-windowed	425	70	252	241	118
MLP-windowed	402	74	280	299	165
GRU	438	129	262	233	174
CNN	290	84	189	251	153
CNN-GRU	411	78	284	227	152
GDQN [17]	77.7	-10.8	20.0	20.6	4.63
GDPG [17]	82.0	-6.39	24.3	13.6	2.55
Turtle [17]	69.5	-17.0	25.6	-1.41	-11.7

Table 5. Final portfolio value of different models starting with the same initial investment.

Model name	Final portfolio value
DQN-vanilla	20275
MLP-vanilla	21580
DQN-windowed	19475
MLP-windowed	20760
GRU	21360
CNN	20287
CNN-GRU	21984
DDPG [23]	19791
Min-Variance [23]	14369
DJIA [23]	15428

The future of the work can be viewed from different perspectives.

- As we have experimented, if we could predict the next state of the environment using the current state, and feed the predicted next state to the DRL model, the performance would significantly increase.
- The actor and actor-critic based DRL methods can be tested and compared with the performance of critic based Deep Q-learning algorithm used here.
- Working on offering a metric used to describe the behavior of the market, based on which, we can specify whether the time-series models can work efficiently in rule extraction or not. Using this metric, we can distinguish where to apply models with raw OHLC input or apply time-series input.

References

[1] E. P. Chan, *Quantitative trading: how to build your own algorithmic trading business*. John Wiley & Sons, 2021.

[2] P. Gomber and M. Haferkorn, "High frequency trading," in *Encyclopedia of Information Science and Technology, Third Edition*, IGI Global, 2015, pp. 1–9.

[3] Z. Zhang, S. Zohren, and S. Roberts, "Deep reinforcement learning for trading," *J. Financ. Data Sci.*, vol. 2, no. 2, pp. 25–40, 2020.

[4] P. Ganesh and P. Rakheja, "VLSTM: Very Long Short-Term Memory Networks for High-Frequency Trading," *arXiv Prepr. arXiv1809.01506*, 2018.

[5] A. Arévalo, J. Niño, G. Hernández, and J. Sandoval, "High-frequency trading strategy based on deep neural networks," in *International conference on intelligent computing*, 2016, pp. 424–436.

[6] M. F. Dixon, N. G. Polson, and V. O. Sokolov, "Deep learning for spatio-temporal modeling: dynamic traffic flows and high frequency trading," *Appl. Stoch. Model. Bus. Ind.*, vol. 35, no. 3, pp. 788–807, 2019.

[7] M. Taghian, A. Asadi, and R. Safabakhsh, "Learning financial asset-specific trading rules via deep

reinforcement learning," *Expert Syst. Appl.*, p. 116523, 2022.

[8] J. Moody, L. Wu, Y. Liao, and M. Saffell, "Performance functions and reinforcement learning for trading systems and portfolios," *J. Forecast.*, vol. 17, no. 5-6, pp. 441–470, 1998.

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

[10] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[11] S. Luo, X. Lin, and Z. Zheng, "A novel CNN-DDPG based AI-trader: Performance and roles in business operations," *Transp. Res. Part E Logist. Transp. Rev.*, vol. 131, pp. 68–79, 2019.

[12] J. Wang, Y. Zhang, K. Tang, J. Wu, and Z. Xiong, "Alphastock: A buying-winners-and-selling-losers investment strategy using interpretable deep reinforcement attention networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1900–1908.

[13] Z. Xiong, X.-Y. Liu, S. Zhong, H. Yang, and A. Walid, "Practical deep reinforcement learning approach for stock trading," *arXiv Prepr. arXiv1811.07522*, 2018.

[14] J. B. Chakole, M. S. Kolhe, G. D. Mahapurush, A. Yadav, and M. P. Kurhekar, "A Q-learning agent for automated trading in equity stock markets," *Expert Syst. Appl.*, vol. 163, p. 113761, 2021.

[15] T. Théate and D. Ernst, "An application of deep reinforcement learning to algorithmic trading," *Expert Syst. Appl.*, vol. 173, p. 114632, 2021.

[16] A. Brim, "Deep reinforcement learning pairs trading with a double deep Q-network," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, 2020, pp. 222–227.

[17] X. Wu, H. Chen, J. Wang, L. Troiano, V. Loia, and H. Fujita, "Adaptive stock trading strategies with deep reinforcement learning methods," *Inf. Sci. (Ny).*, vol. 538, pp. 142–158, 2020.

[18] T. N. Rollinger and S. T. Hoffman, "Sortino: a 'sharper'ratio," *Chicago, Illinois Red Rock Cap.*, 2013.

[19] L. Weng, X. Sun, M. Xia, J. Liu, and Y. Xu, "Portfolio trading system of digital currencies: A deep reinforcement learning with multidimensional attention gating mechanism," *Neurocomputing*, vol. 402, pp. 171–182, 2020.

[20] A. Asadi and R. Safabakhsh, "The encoder-decoder framework and its applications," in *Deep Learning: Concepts and Architectures*, Springer, 2020, pp. 133–167.

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

[22] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv Prepr. arXiv1406.1078*, 2014.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

[24] W. F. Sharpe, "The sharpe ratio," *Streetwise–the Best J. Portf. Manag.*, pp. 169–185, 1998.

[25] Y. Wang, D. Wang, S. Zhang, Y. Feng, S. Li, and Q. Zhou, "Deep Q-trading," *cslt. riit. tsinghua. edu. cn*, 2017.

مدل مبتنی بر چارچوب کاری کدگذار-کدگشا با یادگیری تقویتی عمیق برای یادگیری استراتژی‌های معاملاتی سهام

مهران تقیان، احمد اسدی و رضا صفابخش*

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی امیرکبیر، تهران، ایران.

ارسال ۱۴۰۱/۰۴/۱۵؛ بازنگری ۱۴۰۱/۰۸/۳۰؛ پذیرش ۱۴۰۱/۱۰/۱۸

چکیده:

استراتژی‌های معاملاتی یادگرفته شده توسط مدل‌های مبتنی بر یادگیری تقویتی عمیق دسته از مدل‌ها بخصوص در حوزه معاملات پربسامد، نسبت به مدل‌های دیگر عملکرد بهتری از خود نشان داده‌اند. پژوهش‌ها نشان داده‌اند، کیفیت ویژگی‌های استخراج شده از دنباله‌های زمانی بلندمدت قیمت‌ها توسط مدل‌های یادگیری تقویتی عمیق، بطور مستقیم بر عملکرد استراتژی‌هایی که این مدل‌ها یاد می‌گیرند، تاثیر می‌گذارد. از طرفی در مسائلی که استخراج ویژگی‌های غنی از دنباله‌های زمانی پیچیده در آن‌ها مهم است، مانند مسائل ترجمه ماشینی و تولید شرح بر ویدیوها ،ساختارهای عصبی مبتنی بر چارچوب کاری کدگذار-کدگشا عملکرد خیره‌کننده‌ای از خود به نمایش گذاشته‌اند. ساختارهای عصبی مبتنی بر چارچوب کاری کدگذار-کدگشا قادرند بطور توامان استخراج ویژگی‌های غنی از دنباله‌های زمانی قیمت‌ها و تصمیم‌گیری در خصوص خرید و فروش سهام بر اساس این ویژگی‌ها را یاد بگیرند. در این مقاله، یک مدل انتها به انتها جدید برای استخراج ویژگی‌های غنی از دنباله‌های زمانی بلندمدت قیمت‌ها و تصمیم‌گیری در خصوص خرید و فروش تک‌سهم‌ها در گام‌های زمانی مختلف ارائه شده است. ساختار عصبی ارائه شده در این مقاله، از دو بخش عصبی کدگذار و کدگشا تشکیل شده است که به ترتیب مسئول یادگیری استخراج ویژگی و یادگیری استراتژی‌های معاملاتی هستند. پارامترهای این دو بخش بطور توامان یاد گرفته می‌شوند. بعلاوه در این مقاله، تاثیر معماری‌های عصبی مختلف برای کدگذار و کدگشا روی عملکرد نهایی مدل مورد بررسی قرار گرفته است. نتایج آزمایشات نشان می‌دهد، مدل ارائه شده نسبت به مدل‌های ارائه شده اخیر برای یادگیری استراتژی‌های معاملاتی تک‌سهم در بازارهای مالی عملکرد بهتری دارد.

کلمات کلیدی: یادگیری تقویتی عمیق، مدل Deep Q-Learning، معامله تک‌سهم در بازارهای مالی، چارچوب کاری کدگذار-کدگشا.