공학박사학위논문

Self-attention Based Recurrent Reinforcement Learning for Algorithmic Trading

알고리즘 트레이딩을 위한 셀프 어텐션 기반 순환강화학습

2023 년 8 월

서울대학교 대학원

산업공학과

곽 동 규

Self-attention Based Recurrent Reinforcement Learning for Algorithmic Trading

알고리즘 트레이딩을 위한 셀프 어텐션 기반 순환강화학습

지도교수 장 우 진

이 논문을 공학박사 학위논문으로 제출함

2023 년 8 월

서울대학교 대학원

산업공학과

곽 동 규

곽동규의 공학박사 학위논문을 인준함

2023 년 8 월

Abstract

Self-attention Based Recurrent Reinforcement Learning for Algorithmic Trading

Dongkyu Kwak
Department of Industrial Engineering
The Graduate School
Seoul National University

The advancement of deep neural networks and the capability to effectively process complex data has resulted in a significant increase in academic interest in the application of algorithm trading modeled on neural network structures. The utilization of machine learning in algorithmic trading is driven by two primary objectives. The first objective is to identify meaningful characteristics that can shed light on the fluctuations observed in the financial market. The second objective is to detect underlying causal relationships within multivariate financial time series data. The task of extracting valuable features from financial time series to make predictions or explain market movements is challenging due to the inherent volatility and high levels of noise present in such data. Most algorithmic trading methodologies to date have primarily focused on the process of feature engineering, aimed at directly extracting meaningful features or factors from financial time series data. The majority of algorithmic trading models currently in use determine the optimal position through supervised learning models, such as predicting direction or price, rather than learning from a profit-maximizing objective function. This approach not only fails to fully incorporate the direct utility of a given position, but also leads to double error resulting from indirect decision making. In light of these limitations, reinforcement learning-based algorithmic trading models have emerged as a viable alternative. These models learn the optimal behavior by maximizing the expected reward from observations within a given market environment. This approach overcomes the limitations of supervised learning-based algorithmic trading models by directly incorporating the direct utility of a given position.

The present study proposes a novel convergence model that integrates recurrent reinforcement learning (RRL) and the self-attention mechanism. The efficacy of the proposed model is rigorously tested using various financial time series data sourced from the stock market. The model structure is first defined through the identification of its key components, including the environment, reward, state-space, and action space. The proposed model is built upon RRL, a policy-based model that leverages time dependency among generated trading signals across different time-stamps. RRL recurrently generates trading signals based on previous signals, utilizes market observations as inputs to the policy function, and seeks to maximize expected rewards through the optimization of the utility function with regards to returns. To enhance the performance of the RRL, the proposed model combines auxiliary neural networks, including a supervised learning sub-network for prediction power, a feature extraction sub-network for reconstructing the original input sequence, and a self-attention mechanism for reallocating temporal weights within latent state variables. The proposed model is described in two ways in this paper, including a finite time horizon case and an infinite horizon case. Furthermore, it is demonstrated that the martingale assumption for short-term fluctuations in the infinite time horizon case enables the model to learn from the same advantage setting as in the finite time horizon case.

In this study, two empirical applications of the proposed model were carried out and the results were analyzed in a systematic manner. The first application involved the use of the finite time horizon Deep RRL algorithmic trading model to perform intra-day trading on stocks listed on the KOSPI200 index. The experiment was conducted using 40 days of minute-level price and volume (OHCLV) data for KOSPI200 listed stocks, spanning from March to June 2019. Stocks that experienced external market interventions, such as volatility interruptions and limited price movements, were excluded from the experiment. To enhance the diversity of the training set, data augmentation was employed to reduce the momentum effect and correlation effect among stocks. The second application involved the use of the infinite time horizon Deep RRL algorithmic trading model to perform daily trading on stocks listed on the SP 500 index. The experiment was conducted using daily OHCLV data for SP500 listed stocks, comprising the largest market cap stocks in individual sectors, ranging from January 2000 to January 2020. All data used in the experiments were divided into a training set, validation set, and test set, and a single model was trained for all stocks in the market using the training set.

In the empirical applications of the proposed Deep RRL model, a comprehensive comparative analysis was performed to evaluate the performance of the model against various other commonly utilized models in algorithmic trading. The models considered for comparison include Long-short term memory (LSTM) and Random

Forest, which are typical supervised machine learning models used for predicting direction or price. Another model considered was the A3C policy-based reinforcement learning model. Additionally, the ARIMA time series model was also included in the comparison. To further verify the efficacy of the proposed model, an ablation study was conducted in the infinite time horizon experiment. The study aimed to assess the impact of each of the additional sub-networks on the model's performance. The results showed that the proposed models outperformed the other models in terms of nominal return and return on risk, suggesting that the proposed model has better performance in terms of returns. The results of the ablation study also indicated that the additional structures of the proposed models contribute to improving the performance of the recurrent reinforcement learning model.

Keywords: Recurrent Reinforcement Learning, Self-attention Mechanism, Sequential Auto-encoder, Gated Recurrent Unit, Algorithmic Trading, Intra-day Trading
Student Number: 2016-21095

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Research Motivation and Purpose

Recently, the deep neural network has been most actively used among all machine learning methodologies Deep learning models use a vast set of nodes and layers, where parameters or latent variables within the model are parallelly trainable. Before the prosperity of deep learning in the last decade, most applications of machine learning models highly depended on elaborate feature engineering methodologies. As deep learning models can accommodate high model capacity, the burden on feature engineering has been dramatically reduced. These characteristics of deep learning allow for dealing with a large set of high-dimensional data, even unstructured ones such as images, sounds, and natural languages.

As the automation and individualization of financial service have been gradually highlighted recently, deep learning application for the financial industry of the fintech industry is also actively researched, such as credit ratings or fraud detection driven by deep pattern recognition[54, 19]. Though deep learning methodologies can be adapted to handle various types of data in hand, it is still challenging to extract valuable features from financial time series with deep learning due to its high level of noise. Thus most research on prediction or forecasting from financial time series still has highly relied on feature engineering methodologies such as filtering[17] or technical indicators, even though deep learning models are adopted. In a practical sense, point estimation in financial forecasting is nearly infeasible, where the directional prediction model has the accuracy hit ratio of around 55 and 60%[59].

In order to overcome this limitation, reinforcement learning for algorithmic trading has been widely studied, which deals with the market environment in a reactive way rather than forecasting. Reinforcement learning aims to find the optimal policy that maximizes the expected rewards from interaction with the agent and environment[63]. Reinforcement learning guides itself via trial and error via interacting with the environment, which differs from that supervised learning models specify direct learning targets to hit the correct answers. For this reason, reinforcement learning is advantageous in robotic mechanics or gaming bots where continuous interactions occur between agents and the environment [50]. Recent reinforcement learning models use deep neural networks to construct value or policy functions. The deep neural network makes way for reinforcement learning to deal with the high dimensional features such as visual information in an end-to-end manner. These alleviate the effort for elaborate manual feature extraction in classical reinforcement learning settings. For instance, deep learning plays a critical role in reinforcement learning after the introduction of the Deep Q-Network (DQN) [44], which utilizes deep learning structures such as experienced replay and convolution neural network (CNN).

Despite the growing use of Reinforcement Learning (RL) in algorithmic trading, several challenges impede its successful implementation. The traditional RL algorithms struggle to capture the non-stationary characteristics of financial time series such as structural change, volatility clustering, and long-term memory property[12], that are typically observed in financial data[55, 32]. While some recurrent neural network models, such as Gated Recurrent Units (GRUs)[11] and Long-Short Term Memory (LSTM)[22], may partially address the challenge of long-term dependencies in financial time series, there remains significant room for improvement in the application of RL to financial data. Moreover, the instability of convergence in deep reinforcement learning algorithms when applied to noisy environments also needs to be addressed. Model-free RL schemes often suffer from large variance, disrupting their convergence. For instance, actor-critic based RL algorithms tend to converge slowly and despite several modifications, such as Trust Region Policy Optimization (TRPO)[56], Proximal Policy Optimization (PPO)[57], and Soft Actor-Critic (SAC)[20], no universally robust algorithm exists, as the optimal algorithm depends on the specific use case. Therefore, there is a pressing need for research into RL algorithms that are specifically tailored for algorithmic trading, to address these challenges and make RL a more practical and effective tool for financial applications.

In this dissertation, a novel and innovative model referred to as the 'Self-attention based deep direct recurrent reinforcement learning with hybrid loss (SA-DDR-HL)' has been proposed to address the challenges faced by conventional reinforcement learning methods in the application of algorithmic trading. The SA-DDR-HL, which is an advanced form of recurrent reinforcement learning, possesses two unique characteristics that differentiate it from traditional recurrent reinforcement learning models. First, the SA-DDR-HL features a hybrid learning structure, where the network loss encompasses both the forecasting model loss of supervised learning and the generative model loss of unsupervised learning. While most reinforcement learning models obtain their update targets through interaction with the environment, SA-DDR-HL utilizes historical data to compute the reward target in a price-taker setting. This results in the generation of a prediction loss from the determined target return, allowing the latent input variable to exhibit predictive power. Furthermore, SA-DDR-HL utilizes unsupervised loss from a gated recurrent unit auto-encoder, a sequential generative network, to endow the latent input variable with feature extraction properties. The SA-DDR-HL combines both the proximal policy optimization objective[57] and the utility objective to achieve maximum expected rewards with reduced gradient variance and stable convergence. The self-attention mechanism, as proposed in [67], is applied to the latent input sequences for reinforcement learning in SA-DDR-HL. This mechanism takes into consideration the contextual similarity between elements in input and output sequences and provides temporal feature importance in the learning process. Unlike traditional attention mechanisms, which learn the similarity importance between the input and target sequences, the self-attention mechanism in SA-DDR-HL operates without a target sequence, allowing it to generate trading signals based on both practical observations and known signal patterns in a long-term manner.

In this dissertation, we present a novel reinforcement learning model that overcomes the limitations of traditional Markov Decision Process-based models. Our proposed model incorporates a unique feature that the agent generates actions based on an awareness of the causal importance between the agent's actions. By extending the traditional RRL framework, which only considers the signal generated at a previous point in decision making, our proposed model takes into consideration the time dependency between different signal points or observations in a single iteration. To evaluate the comparative advantage of the proposed model, we conducted experiments in two scenarios. The first scenario focused on intra-day trading with a finite time horizon, while the second scenario focused on daily trading with an infinite time horizon. Before conducting the simulations, we hypothesized that the RL objective function of the two different environment schemes could be approximated to a single utility function. The results of our experiments showed that the proposed model demonstrated a comparative advantage over other forecasting models based on machine learning or classical time series models. Our model demonstrated improved performance in terms of returns such as nominal return and return on risk. The self-attention mechanism and hybrid loss structure in our proposed model, combined with the ability to generate trading signals based on both hands-on observations and known signal patterns in a long-term manner, contribute to its improved performance compared to other models.

1.2 Organization of the Thesis

The structure of this dissertation is organized as follows. In Chapter 2, we undertake an extensive review of prior works in the field of algorithmic trading utilizing the reinforcement learning framework, which has significantly informed and influenced our own research. Additionally, this chapter offers an in-depth analysis of the various machine learning methodologies that have been utilized in the development of algorithmic trading models. In Chapter 3, we present a comprehensive examination of the theoretical components that form the basis of the proposed machine learning methodology. This includes the discussion of relevant and related concepts and methodologies that enable a comparative evaluation of our proposed model. In Chapter 4 and 5, we conduct a comprehensive set of experiments to comparatively evaluate the performance of our proposed model with other existing models. Specifically, Chapter 4 presents the results of experiments conducted in the finite horizon case, whereas Chapter 5 focuses on the results of experiments performed in the infinite horizon case. Finally, in Chapter **??**, we present our conclusion, highlighting the major findings of this research, and suggesting potential avenues for future research.

Chapter 2

Literature Review

2.1 Sequential Model for Time Series

he Autoregressive Integrated Moving Average (ARIMA) model is a well-established and widely utilized method in time-series analysis, serving as a cornerstone in the field of econometrics. The model was introduced by Box and Jenkins, and since then it has been widely adopted as a benchmark model for sequential data analysis [7]. ARIMA models the dependency between each point in a time series as a linear relationship, making it a simple yet powerful tool for capturing the underlying dynamics of the data. One of the key advantages of ARIMA models is their ability to provide a statistical estimate of parameters and perform hypothesis tests, thus allowing for a comprehensive evaluation of the model's goodness-of-fit. Additionally, the model has been extended over time to reflect various phenomena that are commonly observed in time-series data, such as seasonality, volatility clustering, and heteroskedasticity [65, 14]. These advancements have allowed ARIMA to maintain its status as a versatile and robust method for time-series analysis, and have ensured its continued relevance and importance in the field.

Despite the benefits of simplicity and comprehensibility that parametric models such as ARIMA and GARCH offer in explaining time series dynamics, they also suffer from several limitations. Firstly, these models are highly prone to sensitivity with respect to the choice of model specification. The selection of lags and handling of stationarity can greatly alter the model's interpretation, even though modern machine learning models also rely on the selection of hyperparameters. Secondly, these models have a lower level of complexity and higher dependence compared to advanced time series models based on machine learning techniques. This becomes particularly evident in the presence of nonlinearities and high-dimensional time series. Classical time series models, such as the assumption of normality of innovations, which can be applied to simple models, become difficult to apply to non-linear or high-dimensional models, highlighting the limitations of parametric models.

The utilization of Recurrent Neural Networks (RNNs) [23] represents a major advancement in the field of time series modeling by surpassing the limitations posed by model-dependent approaches. RNNs are a type of artificial neural network that employ recurrent connections in their structure. The utilization of multiple layers of neurons in the artificial neural network enables the approximation of complex N-to-N functions, which grants RNNs a remarkable versatility in handling a diverse range of machine learning tasks. Instead of modeling the output sequence directly through the input sequence, RNNs introduce a series of intermediate variables that serve to estimate the complex dynamics between time series. Each hidden neuron in the network is affected by both previous hidden neurons and input neurons, giving rise to the designation of the structure as a 'recurrent' network. RNNs may be viewed as analogous to the Kalman filter [28] in the sense that both approaches estimate the dynamics of a sequence by incorporating latent variables. However, the key difference between the two lies in the fact that while the Kalman filter is limited to modeling linear dynamics, RNNs possess the added advantage of being capable of approximating nonlinear dynamics through the utilization of a nonlinear activation function.

The limitations of RNN as a universal baseline model in time series modeling using artificial neural networks stem from the vanishing gradient problem [34]. This issue results in the incorrect transfer of the gradient of the sequence's terminal node to preceding nodes as the sequence length increases, which hinders the modeling of time series with long-term dependencies. The unidirectional iterative structure of RNN from the long-term sequence is the root cause of the vanishing gradient problem. To address this issue, it is necessary to not only propagate the nonlinear combination of the input sequence and the hidden sequence to subsequent points but also to introduce a structure that controls the weight for the propagation, thereby alleviating the vanishing gradient problem.

he Long-short term memory (LSTM) model [22] represents a significant advancement in mitigating the limitations of recurrent neural networks (RNNs) in modeling time series with long-term dependencies. In contrast to traditional RNNs, LSTM incorporates a gate structure and a cell state layer that dynamically control the contribution of current inputs and previous hidden neurons in determining the output of each neuron. This mechanism effectively reduces the vanishing gradient problem and enables the model to effectively capture long-term dependencies in time series. Gated Recurrent Units (GRUs) [11] represent a further refinement in the field, providing a simplified gated structure for capturing long-term dependencies compared to LSTMs. GRUs simplify the gated structure of LSTMs, replacing multiple gates with a single unit gate that decides whether previous information should be forgotten or propagated. This design offers a structurally sound method for capturing long-term dependencies, while also increasing the efficiency of model training in comparison to LSTMs.

The utilization of GRU and LSTM as artificial neural network models has proven to be highly effective in addressing various sequential problems, including time series analysis, natural language processing, and unstructured data analysis. However, when it comes to financial time series prediction, there are several challenges that arise. One of the most prominent challenges is the tendency of RNN-based models to converge towards the nearest point, as demonstrated by Chen [9]. This can lead to sub-optimal predictions in financial time series analysis, where the target sequence is usually represented by a return series. Financial time series prediction is further complicated by the presence of stationarity and high levels of noise in return series generated from stock prices. Finding an appropriate representation from a complex input sequence is a challenging task in such scenarios. As an alternative, some models consider generating trading decisions through other methods, such as classification or clustering, instead of price prediction. However, such models typically only achieve hit ratios of around 55 to 60%, as stated in the introductory chapter. Additionally, these models often lack a direct measure of the risk associated with a trading decision, even if the predicted probability is relatively high or low.

In conclusion, the utilization of neural network models for the generation of trading signals offers the potential to effectively address long-term dependencies and to process vast amounts of data, including unstructured data, in a manner that considers high and complex dimensions. Despite these advantages, the application of supervised sequential neural network models in this field is subject to various challenges, including difficulties in achieving convergence in the prediction of returns, the appropriate selection of data, the intricate process of feature engineering, and accurately reflecting returns in relation to risk.

2.2 Attention Models

The attention mechanism has its roots in the Sequence-to-Sequence (Seq2Seq) model [62], which is particularly well-suited to solving the task of natural language translation. Seq2Seq consists of two primary components, namely an encoder Recurrent Neural Network (RNN) and a decoder RNN. The encoder RNN generates a context vector based on the input sequence, while the decoder RNN generates the target sequence using this context vector. Although the Seq2Seq model has the advantage of being capable of effectively handling sequences of varying lengths, it is subject to the same vanishing gradient problem that plagues other RNN models. Furthermore, compressing the entire input sequence into a single context vector can result in a significant loss of information, particularly for long sequences. To address these limitations, an alternative approach was introduced by utilizing a scoring function that considers the semantic relevance of each embedding in the input sequence to generate each output of the target sequence. This scoring function serves as the foundation for the attention mechanism, which allows for a more nuanced and information-preserving representation of the input sequence in the target output.

The origin of the attention mechanism was rooted in the need to facilitate the generation of a context vector from the input sequence. This was accomplished by incorporating the context vector with the previous hidden state of the decoder network.[4] The attention layer plays a crucial role in the Seq2Seq model, where it integrates all the elements of the input sequence and the last hidden state of the generated target sequence. The aim of the attention layer is to determine the most relevant or similar values between the most recently predicted output value in the decoder layer and the elements of the input sequence. The weight value generated by the attention layer is utilized to calculate the weighted average of the hidden states of all input sequences. This results in the creation of a context vector, which serves as the genesis state for the target sequence. The seq2seq model operates in a sequential manner, where it iteratively passes through the input sequence to generate one context vector, which is then fed into the decoder. As the attention layer calculates a new context vector in the process of generating each target sequence, the information from the original sequence and elements with high semantic relevance are given relatively more emphasis in the generated target sequence.

The original attention mechanism was developed to mitigate the loss of information from the input sequence during the creation of a context vector in the Recurrent Neural Network (RNN) based sequence-to-sequence (Seq2seq) structure. However, the Transformer model proposed a novel attention mechanism that learned the hidden representation of the sequences of the encoders and decoders using only the attention layer between the sequences.[67] The main innovation introduced in the Transformer was the implementation of the self-attention structure. Unlike the previous attention mechanism, where the scoring was performed directly between the latest element of the decoder and each neuron of the encoder, the self-attention structure enables the allocation of temporal weights not only from the encoder-decoder connection but also from the input sequence of the encoder or decoder itself. The original attention mechanism was limited in its ability to learn the relevant importance of each feature, as it utilized a relatively simple network and relied on the traditional RNN structure. As a result, it struggled to learn long-term sequences. However, the self-attention structure transformed each input or output sequence into independent multi-dimensional features, making it possible to stack multiple attention layers for processing. This versatility provided an advantage in that the self-attention mechanism was relatively less constrained by the length of the sequence. As a result of its versatility, the self-attention mechanism is now widely used in natural language processing and has been applied to various problems in the field of time series.[39]

2.3 Financial Trading Agents using Reinforcement Learning

The Recurrent Reinforcement Learning (RRL) model, as documented in the seminal work by Moody et al. [47], constitutes a significant advancement in the realm of policy-based reinforcement learning. It proposes the imposition of recurrence on the action sequence, thereby surpassing the limitations imposed by the Markov Decision Process (MDP) assumption in traditional reinforcement learning. The RRL approach incorporates the real-time recurrent learning methodology put forth by Williams et al. [72] to alleviate the cascading property of observations, inherent in the MDP assumption. The RRL framework proposes the utilization of the Sharpe ratio utility function as the objective function for online reinforcement learning. The model shows that the gradient of the parameters can be updated in an online manner using the recurrent gradient of the differential Sharpe ratio utility function. Furthermore, RRL takes into consideration the sequential dependency between action values and the market observations, thereby allowing for a more comprehensive evaluation of the dynamics of the system. Empirical evidence, presented in the RRL paper, supports the superiority of the performance of the trading agent in the RRL framework compared to the Q-learning framework. This evidence further highlights the practical applicability of RRL in the realm of algorithmic trading, incorporating crucial variables such as transaction costs, direct returns on position, and the risk-free rate. In conclusion, the RRL model represents a crucial contribution to the field of reinforcement learning and algorithmic trading, offering a more comprehensive and effective approach to decision-making.

The introduction of the Deep Q-network (DQN) algorithm by Mnih et al. [44] marked a significant milestone in the field of artificial intelligence. DQN proposed the extraction of latent features from images using convolutional neural networks, thereby demonstrating the crucial role of feature extraction through artificial neural networks in the state space. While the DQN algorithm has been applied to financial trading agents [24], analogous trials have also been conducted in the Recurrent Reinforcement Learning (RRL) framework. The Deep Direct Reinforcement Learning (DDR) approach, as proposed by Deng et al. [12], represents a significant advancement in policy-based frameworks. DDR introduced the concept of a separate feature extraction network and demonstrated the superiority of policy-based frameworks over value-based ones in terms of optimization flexibility and continuous descriptions of market conditions. The DDR framework suggested incorporating the training of deep auto-encoder feature extraction networks and policy networks using task-aware Back-propagation Through Time (BPTT) from the utility-maximizing objective function. The BPTT algorithm in DDR is used not only to train the direct policy function from the sequence of rewards and policies, but also to backpropagate the BPTT gradients to the feature extraction networks in a sequential manner. The feature extraction process employs fuzzy clustering to generate denoised features from input observations and cluster them based on trends. The deep auto-encoder model is used as an auxiliary tool to impose reconstructive power on the latent input features, and its robust denoised features have been exploited by DDR to extract hidden representations [68]. In a manner similar to DQN, which effectively combined deep feature extraction for images and value-based reinforcement learning using Convolutional Neural Networks (CNNs), DDR also combined automated feature extraction and reinforcement learning methodologies, reducing the burden of elaborate feature engineering. DDR represents a crucial contribution to the field of reinforcement learning and feature extraction, offering a more effective and efficient approach to decision-making in complex systems.

The study by Li et al.[35] proposed a unique approach of combining the Deep Q-network with a Recurrent Neural Network (RNN) to indirectly learn the parameters of the value function in a recurrent manner. This approach expands upon previous studies that utilized neural networks for feature extraction. The author proposed the utilization of a hybrid loss function, which is more comprehensive compared to previous studies. In this study, each time point's observation is utilized as an input sequence to the RNN, and the hidden sequence of the RNN is not just employed as the state variables of the value function but also is connected to an additive subnetwork that predicts the subsequent observation and rewards. This results in predictive power for each hidden state of the RNN. This hybrid loss approach contributes to the performance improvement of the reinforcement learning (RL) agent, as emphasized by the authors. The integration of the Deep Q-network and RNN in this manner presents a promising direction for further research in the field of reinforcement learning. It highlights the potential of combining deep neural networks with recurrent architectures to enhance the performance of RL agents in various domains, including financial trading.

The authors of Time-driven Feature-aware Jointly Deep Reinforcement Learning (TFJ-DRL)[33] present a novel approach to policy-based reinforcement learning that leverages the strengths of gated neural networks and attention structures. This approach is designed to learn the relative importance of features in the learning process and to evaluate the similarity between features over time. The authors introduced a gated neural network to assess feature importance, and an attention module to evaluate the similarity between the current time point and previous ones. The TFJ-DRL framework builds upon the RRL framework, generating a trade action sequence through a recurrent policy network and using the Vanilla Policy Gradient (VPG) optimization, as opposed to utility functions. Furthermore, the authors introduce a hybrid loss function that leverages supervised learning loss. The attention structure helps to extract salient points from noisy and unsegmented sequences, as demonstrated in previous studies on temporal attention-gated models[51]. This novel approach to policy-based reinforcement learning considers the temporal importance of features throughout an episode, while considering the similarity between current and previous features. The use of a gated neural network, attention structure, and hybrid loss function makes TFJ-DRL a promising approach for algorithmic trading.

In this study, we propose a novel hybrid model that leverages the strengths of previous works while overcoming their limitations. The Recurrent Reinforcement Learning (RRL) framework, while imposing recurrence on the action sequence to consider the previous positions of the agent, only refers to the last previous position with a trainable constant weight. This structural limitation makes the model less adaptive to changes in market conditions. To address this issue, our proposed model incorporates the self-attention mechanism [67] to allocate the sequence of hidden features in an episode based on their temporal importance, providing greater flexibility in dealing with temporal dependencies on the feature side. Furthermore, our model jointly learns from the utility objective function commonly used in the RRL framework and a general policy-based reinforcement learning optimization such as Proximal Policy Optimization (PPO), balancing effective convergence of learning with profit maximization. Our model also employs a hybrid loss function, comprising a reconstruction loss and a forecasting loss, to ensure that the hidden feature reflects the various latent characteristics of the original input sequence and to mitigate the gradient vanishing problem that arises from dilution of the gradient of the last loss along the deep structure. The innovation of our research is demonstrated in Figure 2.1. Our proposed model is expected to make a significant contribution to the field of financial trading agents and to provide a promising solution to the challenges faced in this domain.

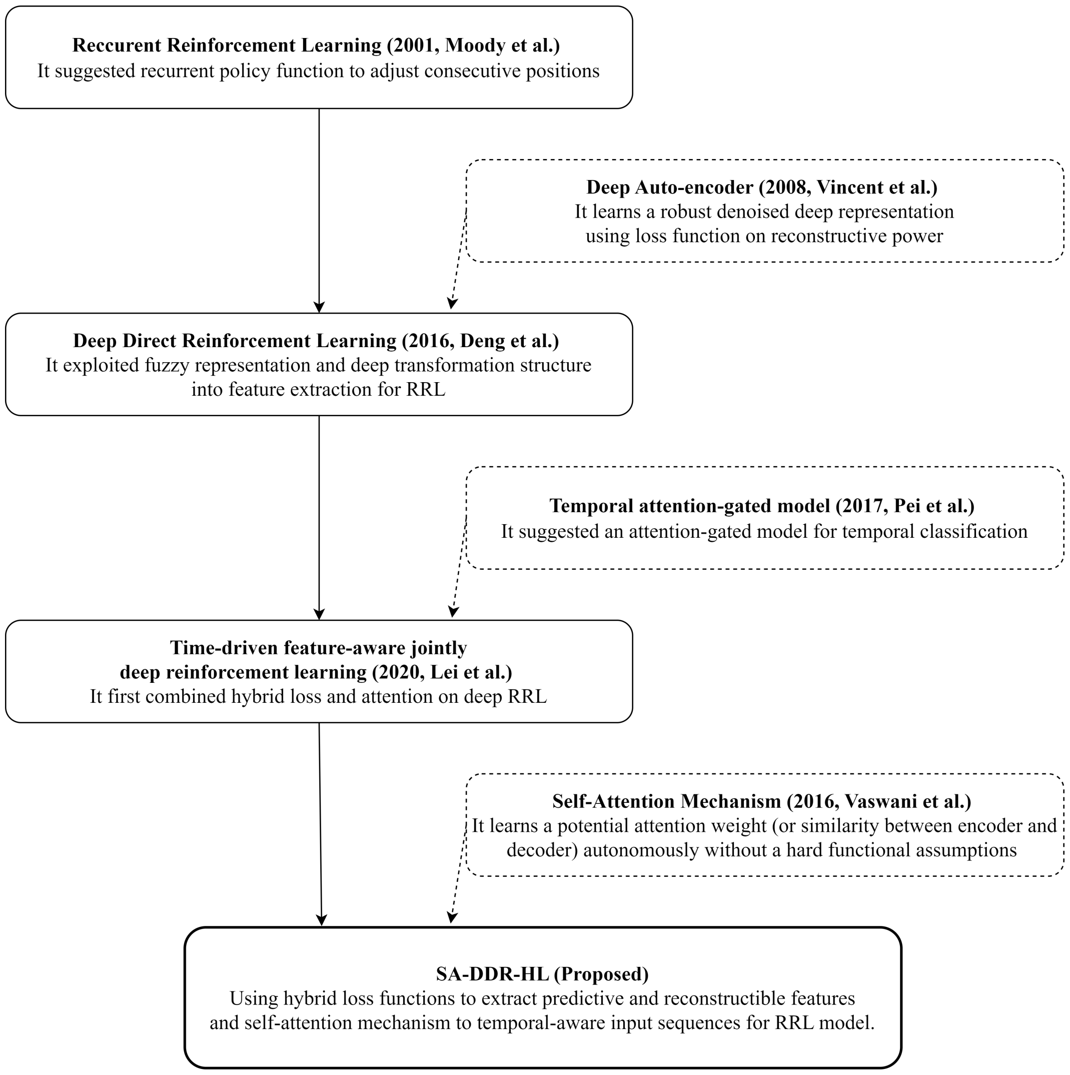

Reccurent Reinforcement Learning (2001, Moody et al.)
It suggested recurrent policy function to adjust consecutive positions
Deep Auto-encoder (2008, Vincent et al.)
It learns a robust denoised deep representation using loss function on reconstructive power
Deep Direct Reinforcement Learning (2016, Deng et al.)
It exploited fuzzy representation and deep transformation structure into feature extraction for RRL
Temporal attention-gated model (2017, Pei et al.)
It suggested an attention-gated model for temporal classification
Time-driven feature-aware jointly deep reinforcement learning (2020, Lei et al.)
It first combined hybrid loss and attention on deep RRL
Self-Attention Mechanism (2016, Vaswani et al.)
It learns a potential attention weight (or similarity between encoder and decoder) autonomously without a hard functional assumptions
SA-DDR-HL (Proposed)
Using hybrid loss functions to extract predictive and reconstructible features and self-attention mechanism to temporal-aware input sequences for RRL model.

Figure 2.1: Diagram of innovations of the research

Chapter 3

Self-attention based Deep Direct Reinforcement Learning with Hybrid Loss

3.1 Reinforcement Learning

3.1.1 Value-Based Reinforcement Learning

The primary setting of reinforcement learning is to construct an environmental setting, which is expressed as the following triplet: $(\mathcal{S},\ \mathcal{A},\ \mathcal{E})$. $\mathcal{S}$, $\mathcal{A}$, are called state space and action space respectively. The state space is the set of all observations that can be observed by the reinforcement learning agent. On the other side, the action space is the set of all actions that the agent can do based on the current space. $\mathcal{E}$ is called the environment. The environment provides a corresponding reward to an agent for the action of the agent. In that setting, let the set of all rewards acquirable be defined as $\mathcal{R}$

In a general model-free reinforcement learning problem, we should find a policy, or target policy that maximizes the expected value of the sum of all future rewards through a given state. Here, a policy function and a value function can be defined. Policy function, or target policy function $\pi : \mathcal{S} \mapsto \mathcal{A}$ is guided to derive maximal expected rewards. Value function $V : \mathcal{S} \mapsto \mathbb{R}$ is defined as expected rewards, such as the following:

$$V(s) := \mathbb{E}[\sum_{i=t}^{\infty} \gamma^{i-t} R_i | s_t = s] \tag{3.1}$$

where γ is a discount factor with a value between 0 and 1 and R_i is a reward at the time point t. Also, The action-value function, which is the value of the expected value function for a given action, can also be defined as follows:

$$Q(s,a) := \mathbb{E}[\sum_{i=t}^{\infty} \gamma^{i-t} R_i | s_t = s, a_t = a] \tag{3.2}$$

The value function V_π and the action-value function Q_π have the following relationship when an action follows a policy pi :

$$V_\pi(s) = \mathbb{E}_{a\sim\pi}[Q(s,a)] = \sum_{a\in\mathcal{A}} Q(s,a)\pi(s) \tag{3.3}$$

When an agent learns policy from environment, The policy used when creating an action from a given state, and the target policy may be the same or different. If the two policies are the same, it is referred to as an on-policy manner, and if they are different, it is referred to as an off-policy manner. However, general learning scheme such as ϵ-greedy, which follows a random policy with a probability ϵ for exploration and takes an action that maximized the current action-value function with a probability $1 - \epsilon$, two manners are nearly indistinguishable due to the greedy target policy is distributively analogous to ϵ-greedy policy.

As in the definition, the estimation of the value function basically collects all reward sequences within an episode and uses a discounted sum of rewards for them.

We call it a Monte-Carlo estimation, as averaged iterated rewards are used as an estimator of the value function. However, this estimation has the disadvantage that the estimator's variance increases excessively as the episode lengthens. There are two progressive learning algorithms representing on-policy manners and off-policy manners, instead of using Monte-Carlo estimation. One is SARSA and the other is Q-learning, respectively. In the on-policy SARSA setting, a policy π is given, the value function is updated a so-called TD(0) error. TD(0) error estimates value function or its parameters from the immediate reward and the next value function, instead of using the discounted sequence of subsequent rewards. By replacing the projected further rewards with the current value function, a slight bias exists, but the variance of the value function estimator can be significantly reduced. That is, in TD(0) error, the value function is updated as the following manner:

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha(r + \gamma V_\pi(s') - V_\pi(s)) \tag{3.4}$$

where s' is a next state or observation and α is a learning rate. In the SARSA, similar to above, the Q value is updated along a policy π-generally ϵ-greedy policy-, which yields the immediate reward and subsequent state and action. Using those, on-policy SARSA is updated as the following:

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha(r + \gamma Q_\pi(s',a') - Q_\pi(s,a)) \tag{3.5}$$

On the other hand, off-policy Q-learning setting, the value function is updated using a difference of previous value function and a new estimated value function such as TD(0) error. However, which is obtained from the reward, the next state and action along the manner of Q-value maximization, as follows:

$$Q_(s,a) \leftarrow Q_(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{3.6}$$

where a' is all possible action. In the on-policy SARSA manner, the trajectory of agent-the previous and next pair of state and action is used to update the value function. But in the off-policy Q-learning manner, only the current action from ϵ-greedy policy is used only, and the following action used in updating parameters is assumed to maximize the subsequent value function instead of sampling an additive action. Although it cannot be strictly said that one of both is better than the other, it is known that Q-learning learns more exploratively than SARSA, accepting highly negative rewards. Also, some theorems, such as the contraction theorem, guarantee the convergence of the Q-function.[63]

3.1.2 Policy-Based Reinforcement Learning

A framework that performs reinforcement learning only by value function estimation is called a value-based method. In value-based methods, the policy is usually assumed to depend on a particular sampling method or given such as ϵ-greedy. In contrast, a framework that directly learns a policy function instead of relying the policy on the value function for the policy is called a policy-based method. In policy-based method, log-likelihood maximization based methods are basically used, which weighed to the reward of each time point of an episode. As with value-based methods, the most basic way to update a policy function, called REINFORCE, is to directly use a sequence of rewards to update the parameters of the policy function as follows:

$$\theta_\pi \leftarrow \theta_\pi + \alpha \sum_t [\gamma^t R_t \nabla_{\theta_\pi} \log \pi(a_t|s_t, \theta_\pi)] \tag{3.7}$$

where θ_π is parameters of policy function π.

Since this method shared the same problems or limitations as the value function's Monte Carlo estimation such as high variance problem, several methods were developed that uses value function estimation instead of using an immediate reward directly. One of the basic form of those methodologies is Actor-critic method.[29] Actor-critic method use values from the value function as a substitute for reward values. In the actor-critic method, the value function is called a critic, and the policy function is called an actor. When the state vector is shared, the phase of the actor-critic method is split into two steps: one is to update the parameters of the policy function, and the other is to update the parameters of the value function.

The first phase is to use a policy function through policy gradient, which is almost similar to REINFORCE, and there are several variations depending on what reward is replaced with. For example, in the Q actor-critic method, the immediate reward in Eq. 3.7 is replaced to action-value function as follows:

$$\theta_\pi \leftarrow \theta_\pi + \alpha \sum_t [\gamma^t Q_{\pi_\theta}(s, a) \nabla_{\theta_\pi} \log \pi(a_t|s_t, \theta_\pi)] \tag{3.8}$$

Note that the value function along the target policy is approximated to the function with the policy function; that is, the actor-critic method is in an on-policy manner.

For another instance, advantageous actor-critic (A2C) method exploits the advantage instead of Q value. Advantage is defined as the difference between V(s), the expected sum of the discounted reward for the current state, and Q(s,a), the value function according to the action. Considering the advantage, when updating the policy function or estimating the policy function, not only the value function is simply weighted, but also the behavioral benefit of the action can be considered simultaneously. In A2C method, the update of policy function become as follows:

$$\theta_\pi \leftarrow \theta_\pi + \alpha \sum_t [\gamma^t A_{\pi_\theta}(s,a) \nabla_{\theta_\pi} \log \pi(a_t|s_t, \theta_\pi)] \tag{3.9}$$

where $A_{\pi_\theta}(s,a) = Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s)$ If we use an approximation $A(s,a) \approx R + \gamma V(s')$ to the action-value function, the advantage is approximated as $A_{\pi_\theta}(s,a) = R + \gamma V_{\pi_\theta}(s') - V_{\pi_\theta}(s)$.

The second phase is to update value function. Though actions could be sampled from a policy function like a value-based method or use a separate policy like ϵ-greedy, in policy-based methods, the value function is used only as an assistant element to help the policy function to be update, so there are often cases in which the mean-squared-error loss is simply used as shown below:

$$\theta_V \leftarrow \theta_V + \nabla_{\theta_V} \beta [R + \gamma V(s') - V(s)]^2 \tag{3.10}$$

where β is another learning rate and θ_V is parameters for value function.

Although the actor-critic method could be used to learn the optimal policy itself less dependent on value estimation, the actor-critic is also more unstable in convergence or less explorative as the trajectory in an episode relies on past actions[63]. Therefore, many optimization methods are being studied to improve the safety of learning, and one of these optimization methods was borrowed in our study.

3.1.3 Recurrent Reinforcement Learning

Recurrent reinforcement learning (RRL)[47] is a policy-based reinforcement learning method specialized for trading. RRL adopts policy gradient recurrently for the policy function along with time horizon. In the RRL framework, a utility function on rewards is exploited instead of value function estimation, such as the Sharpe ratio. In detail, F_t, the position (or action) of time t, is a recurrent function such as $F_t = F_\theta(F_{t-1}, I_t) \in \{-1, 0, 1\}$ the utility function S_T is defined for the reward sequence $\{R_t\}$ as:

$$S_T = \frac{\bar{R_t}}{(\bar{R_t^2} - (\bar{R_t})^2)^{1/2}} \tag{3.11}$$

where I_t is information (or observation, feature) of time point t, $\bar{R_t}$, $\bar{R_t^2}$ are the average of a reward and a squared reward, respectively, and T is the window size. Here, position values of -1, 0, and 1 mean short, neutral, and long, respectively. In the RRL model, the reward is determined as a direct function of the position at a specific time and the position at a previous point in time, just like the recurrence of a position:

$$R_t = F_t * r_t - \delta|F_t - F_{t-1}| \tag{3.12}$$

where δ is an unit transaction cost and r_t is a return of the asset. As mentioned above, since the reward is determined by the unknown distribution of return and the composition of the policy function, the gradient for the policy function is obtained directly from the predetermined utility function of the reward instead of explicitly setting an estimate for a separate value function.

Since it is difficult to calculate the difference in utility function between adjacent time points in an on-line manner, the differential Sharpe ratio D_t, which corresponds to a first-order approximation of the difference in Sharpe ratio between adjacent time points, was proposed as an alternative to the gradient $\frac{dU_t}{dR_t}$ as following:

$$D_t = \frac{B_{t-1}\Delta A_t - \frac{1}{2}A_{t-1}\Delta B_t}{(B_{t-1} - A_{t-1}^2)^{3/2}} \tag{3.13}$$

where A_t and B_t are estimates of the first and the second order moment with exponential moving average:

$$A_t = A_{t-1} + \eta(R_t - A_{t-1})$$
$$B_t = B_{t-1} + \eta(R_t^2 - B_{t-1})$$

where η is a decay rate. From the above differential Sharpe ratio, the entire policy gradient in the on-line manner from the utility function is approximately determined as follows:

$$\frac{dU_t}{d\theta} = D_t \Big\{ \frac{dR_t}{dF_t}\frac{dF_t}{d\theta} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta} \Big\} \tag{3.14}$$

Note that as the policy gradient $\frac{F_t}{\theta}$ is exploited in a recurrent manner such as a recurrent neural network, the policy function could be approximated to recurrent neural network form.

3.2 Model formulation

3.2.1 Setting of The Agent and The Environment

In the suggested trading agent model in this thesis, we exploit RRL framework for the trading agent. For the model formulation, we set the state space of observation, $\mathcal{S} := \mathbb{R}^{d\times w}$, action space, $\mathcal{A} := \{-1, 1\}$, and time horizon, $\mathcal{T} := \{1, 2, \cdots, \tau\}$. The state space is a normalized time series consisting of open, high, close, low prices and trading volume within window size w. The state of action space is either -1 or 1, corresponding to short and long positions, respectively. We denote $s_t \in \mathcal{S}$ as the sequence observed in the time period from $t-w+1$ to t. The sequence s_t consists of price return rates $\frac{p_{t-j}}{p^c_{t-w}} - 1$ $(j = 0, 1, \cdots, w-1)$ where $p_\cdot$ is a price (open, high, close or low) and $p^c_\cdot$ is a close price. The normalization of volume is the standardized logarithm of volume within a time window. The length of time horizon τ is decided from the batch size. We define the normalized close price at $t+j$ against the close price at $t-w$, the $p^t_{(j)} = \frac{p^c_{t+j}}{p^c_{t-w}}$ $(j = -(w-1), \cdots, -1, 0, 1)$.

The temporal feature encoding function is $\phi^E : \mathcal{S} \mapsto \mathcal{H}$ which maps the state space to the temporal feature space. The hidden representation is $\phi^A : \mathbb{R}^H \mapsto \mathbb{R}^{H\prime}$ which maps temporal feature space to hidden representation space. Recurrent policy function, $\pi_t \in [0, 1]$, for observation $s_t \in \mathcal{S}$ at time t using the input for reinforcement learning $z_t = (\phi^A \circ \phi^E)(s_t) = \phi^A\Big((\phi^E)(s_t)\Big)$ are defined as follows.

$$\pi_t := \pi(z_t,\ \pi(z_{t-1})) = \sigma(w_f \cdot z_t + w_\pi \pi(z_{t-1}) + w_b) \tag{3.15}$$

where $\cdot$ is the inner product, w_f is weight coefficient vector, and w_π and w_b are weight coefficients. Note that σ is a sigmoid function. Actual action value, ρ :

$[0,1] \mapsto \mathcal{A}$, is generated from the policy function π_t defined as follows.

$$\rho_t := \rho(\pi_t) = \begin{cases} 1 & \text{when } \pi_t \geq 1/2 \\ -1 & \text{when } \pi_t < 1/2 \end{cases} \tag{3.16}$$

The reward at time t, R_t, is defined as $R_t := \rho_t(p^t_{(1)} - p^t_{(0)}) - c|\rho_t - \rho_{t-1}|$ where c is a transaction cost.

3.2.2 Policy Optimization

The objective of reinforcement learning network consists of two parts: one is utility maximization objective and the other is surrogate objective. Utility maximization objective $\mathcal{L}^{RL_U}$ is the period yield per volatility from the reward series as follows.

$$\mathcal{L}^{RL_U}(\theta) := \frac{\sum_{t=1}^{\tau} R_t}{\sqrt{\sum_{t=1}^{\tau} R_t^2}} \tag{3.17}$$

As the daily rate of return, r_t, could be calculated from each daily profit that is scaled from the previous close price, the utility maximization objective equation (3.17) can approximate $\frac{\sum_t r_t}{\sqrt{\sum_t r_t^2}}$, which is derived from original Sharpe ratio during the batch period. This part of policy optimization is used to learn a policy that maximizes return over risk in the environment. Thus in this optimization, all the rewards are used in one single loss function value.

On the other hand, the surrogate objective is the objective function to derive a policy gradient from the policy function directly, which is oriented from the proximal policy optimization [57]. The surrogate objective function is used to policy function to be converged stably and is a summed loss of each time point of loss function value

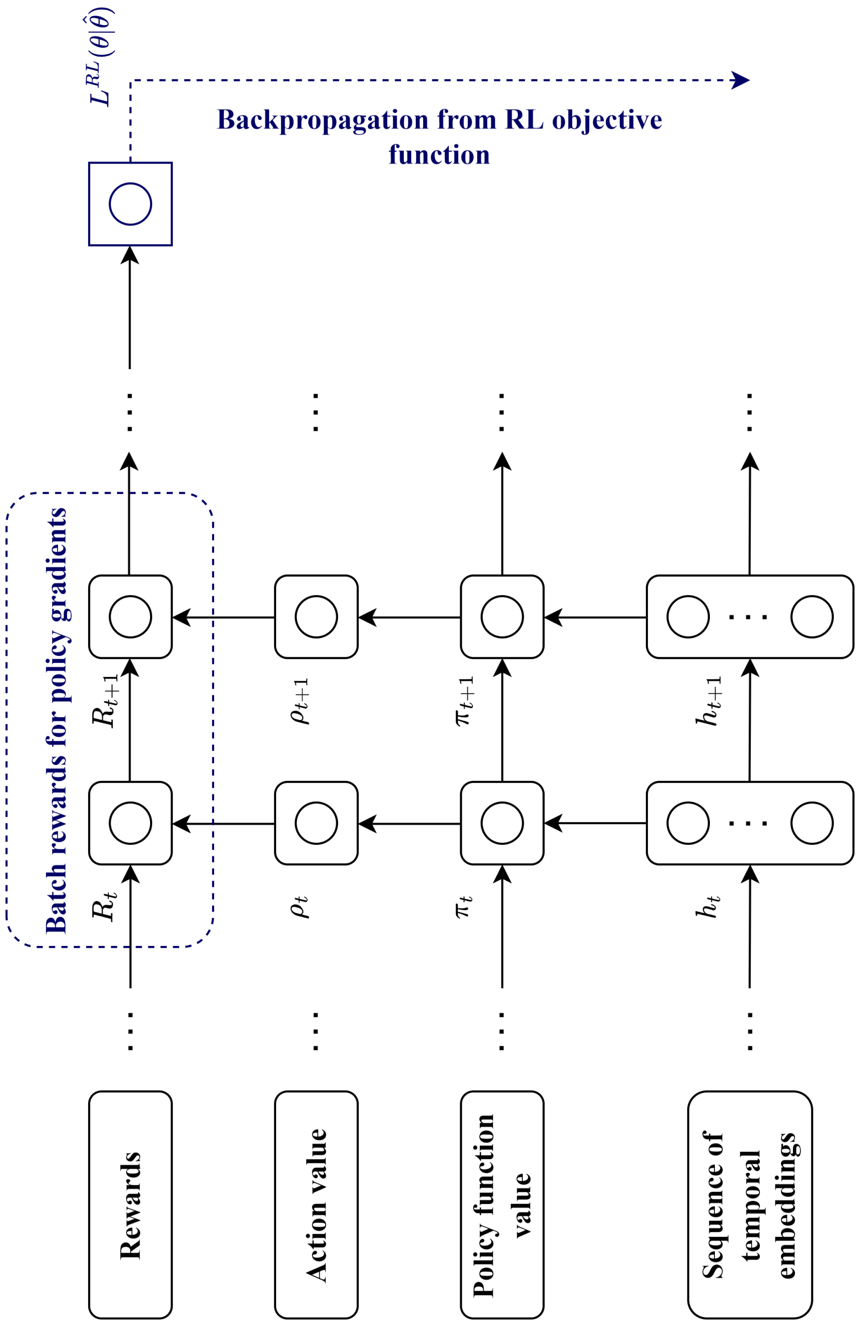

Rewards
Action value
Policy function value
Sequence of temporal embeddings
R_t
R_{t+1}
ρ_t
ρ_{t+1}
π_t
π_{t+1}
h_t
h_{t+1}
Batch rewards for policy gradients
$L^{RL}(\theta|\hat{\theta})$
Backpropagation from RL objective function

Figure 3.1: Recurrent reinforcement learning

from the immediate reward. A surrogate objective $\mathcal{L}^{RL_S}$ is updated from reward by regularizing a new policy value with the previous one as follows.

$$\mathcal{L}^{RL_S}(\theta|\hat{\theta}) := \frac{1}{\tau}\sum_{t=1}^{\tau}\min\left[\frac{\pi(z_t,\pi(z_{t-1}))}{\pi_{\hat{\theta}}(z_t,\pi_{\hat{\theta}}(z_{t-1}))}R_t,\ \mathrm{clip}\Big(\frac{\pi(\mathrm{z_t},\pi(\mathrm{z_{t-1}}))}{\pi_{\hat{\theta}}(\mathrm{z_t},\pi_{\hat{\theta}}(\mathrm{z_{t-1}}))},\ 1-\epsilon,\ 1+\epsilon\Big)\mathrm{R_t}\right] \tag{3.18}$$

where $\hat{\theta}$ is the estimators of the recently updated parameters and $\mathrm{clip(x, a, b)} := \mathrm{x1(a \le x \le b) + a1(x < a) + b1(x > b)}$ assuming $a < b$. Note that $1(\cdot)$ is a binary indicator function.

Based on the assumption that action-value function $Q(z_t, \rho_t) \propto R_t$ and the value function, $V(z_t) \approx 0$, we only use the immediate reward, R_t, to update parameters without estimating the value function separately for advantage function. Note that $Q(z_t, \rho_t)$ and $V(z_t)$ are future rewards expectations.

The policy gradient update method can prevent the parameter estimation variance from diverging more effectively than vanilla policy gradient meThe surrogatetesutton1999policy, schulman2017proximal. The overall policy gradient update is performed jointly by minimizing $\mathcal{L}^{RL}(\theta|\hat{\theta}) := -\lambda_{\mathcal{L}^{RL_U}}\mathcal{L}^{RL_U}(\theta) - \lambda_{\mathcal{L}^{RL_S}}\mathcal{L}^{RL_S the}(\theta|\hat{\theta})$ where $\lambda_{\mathcal{L}^{RL_U}}$ and $\lambda_{\mathcal{L}^{RL_S}}$ are coefficients of positive value. The temporal feature embedding is constructed using gated recurrent unit(GRU) [11], which is suitable for reflecting long-term dependency.

3.2.3 Gated-Recurrent Unit

Gated Recurrent Unit(GRU) [11] is a form of recurrent neural network (RNN) designed to reflect the long-term dependence of a time series. It is characterized by adding a gate structure to the recurrent neural network. Gate unit in recurrent neural network structure was proposed in long short term memory (LSTM) [22] first, and GRU simplified gated structure to enable faster learning [74]. The gate structure dynamically adjusts the weight of information between previous hidden neurons and the new input neuron to account for the new hidden output. We set $x^t_{(j)} \in \mathbb{R}^d$ $(j = 1, 2, \cdots w)$ as the daily feature elements in s_t and denote $h^t_{(j)} \in \mathbb{R}^H$ as the hidden neuron of GRU corresponding to $x^t_{(j)}$. Reset gate $r^t_{(j)}$, update gate $u^t_{(j)}$ and candidate neuron $\tilde{h}^t_{(j)}$ is defined as follows.

$$r^t_{(j)} := \sigma(h^t_{(j)} W_r + x^t_{(j)} U_r + b_r) \tag{3.19}$$

$$u^t_{(j)} := \sigma(h^t_{(j)} W_u + x^t_{(j)} U_u + b_u) \tag{3.20}$$

$$\tilde{h}^t_{(j)} := \tanh\left((r^t_{(j)} \odot h^t_{(j)}) W_h + x^t_{(j)} U_h + b_h\right) \tag{3.21}$$

where $\sigma(\cdot)$ is an element-wise sigmoid function and $\odot$ is the Hadamard product. Note that W_r, W_u, W_h, U_r, U_u, and U_h are weighted matrices and b_r, b_u, and b_h are bias constant matrices. Final hidden unit is recurrently determined as follows.

$$h^t_{(j)} := u^t_{(j)} \odot \tilde{h}^t_{(j)} + (1 - u^t_{(j)}) \odot h^t_{(j-1)} \tag{3.22}$$

We use the final output hidden neuron of GRU, $h^t_{(w)}$, as a temporal embedding of state s_t. Thus, $\phi^E(s_t) = h^t_{(w)}$ holds for the encoding map. From this point on, we set $h_t = h^t_{(w)}$ for the simplicity of notation. Initial hidden neuron is separately set as $h^t_{(1)} := \phi^{E_0}(x^t_{(1)})$ through a fully connected neural network (FCNN) $\phi^{E_0} : \mathbb{R}^D \mapsto \mathbb{R}^H$ which can be trained via backpropagation from temporal embedding.

3.2.4 Hybird Loss from Prediction Network

While learning temporal embedding in section 3.2.3, both the supervised learning to predict the next normalized close price and the unsupervised learning to reconstruct the input state are performed using the same embedding $\phi^E(s_t) = h_t \in \mathbb{R}^H$ in addition to reinforcement learning. This allows temporal embedding to have the reconstruction capability and the predictive power.

Supervised learning is constructed to predict the next close price p^c_{t+1} from temporal embedding $\phi^E(s_t)$ with the mean squared error loss as follows.

$$\mathcal{L}^{H_F}(\theta) := \frac{1}{\tau}\sum_{t=1}^{\tau}[p^t_{(1)} - (\phi^F \circ \phi^E)(s_t)]^2 \tag{3.23}$$

Note that forecasting map $\phi^F : \mathbb{R}^H \mapsto \mathbb{R}$ consists of fully connected network. and the normalized $(t+1)$th close price $p^t_{(1)}$ is used as target value.

Unsupervised learning reflects the temporal feature of subsequence using sequential autoencoder [61]. Decoder network for reconstruction is the GRU network shown in section **??**. The input of decoder network is $[h_t, \cdots, h_t] \in \mathbb{R}^{H\times w}$ where temporal embedding element, $h_t = \phi^E(s_t)$, is a column vector of length H. The target of decoder network is reversed subsequence of s_t, $\overleftarrow{s}_t := [x^t_{(w)}, \cdots, x^t_{(1)}] \in \mathbb{R}^{d\times w}$. The time series decoding map $\phi^D : \mathbb{R}^H \mapsto \mathbb{R}^{d\times w}$ has the objective of mean squared loss minimization for reconstruction as follows.

$$\mathcal{L}^{H_R}(\theta) := \frac{1}{\tau \times \sqrt{D \times w}}\sum_{t=1}^{\tau}\|\overleftarrow{s}_t - (\phi^D \circ \phi^E)(s_t)\|_F^2 \tag{3.24}$$

where $\|\cdot\|_F$ is Frobenius norm defined as $\|A\|_F := \sqrt{\sum_{i,j} A_{ij}^2}$ for arbitrary $A \in \mathbb{R}^{m\times n}$, and $\sqrt{D \times w}$ is regularizing term for stabilizing the variance of estimator. The hybrid loss, linearly combined minizing loss $\mathcal{L}^{H}(\theta) := \lambda_{H_F}\mathcal{L}^{H_F}(\theta) + \lambda_{H_R}\mathcal{L}^{H_R}(\theta)$, is used for updating the weights of temporal encoding network, allowing temporal input features of reinforcement learning to have predictive and reconstructive power. Note that λ_{H_F} and λ_{H_R} are the coefficients of positive value.

3.2.5 Self-attention Mechanism

We apply self-attention mechanism to reallocate temporal weight in the sequence of temporal embedding. The overall self-attention structure is shown in figure 3.3. When basic dot-product attention is used, similarity measure such as cosine similarity among hidden representations is used to learn the temporal importance [4]. However, the model learns temporal dependency among features using a neural network based on the self-attention mechanism[67]. The self-attention mechanism, which is originated from machine translation motivation, exploits query, key, and value to reallocate sequence's temporal importance. As we use encoder structure only here, those three sequences are derived from the identical embedded sequence. The self-attention mechanism consists of the following elements or structure: positional encoding, scaled-dot product, multi-head attention, and residual connection. Embedding sequence, an episode for the recurrent reinforcement learning, is defined as follows:

$$H_t := [h_{t-\tau+1}, \cdots, h_t]^T, \; H_t \in \mathbb{R}^{\tau \times H} \tag{3.25}$$

where $h_j \in \mathbb{R}^H$ $j = t-\tau+1, \ldots, t$ is a column vector. Note that $[\,]^T$ is the transpose of the matrix. Here, we use positional encoding E defined as follows to encode the

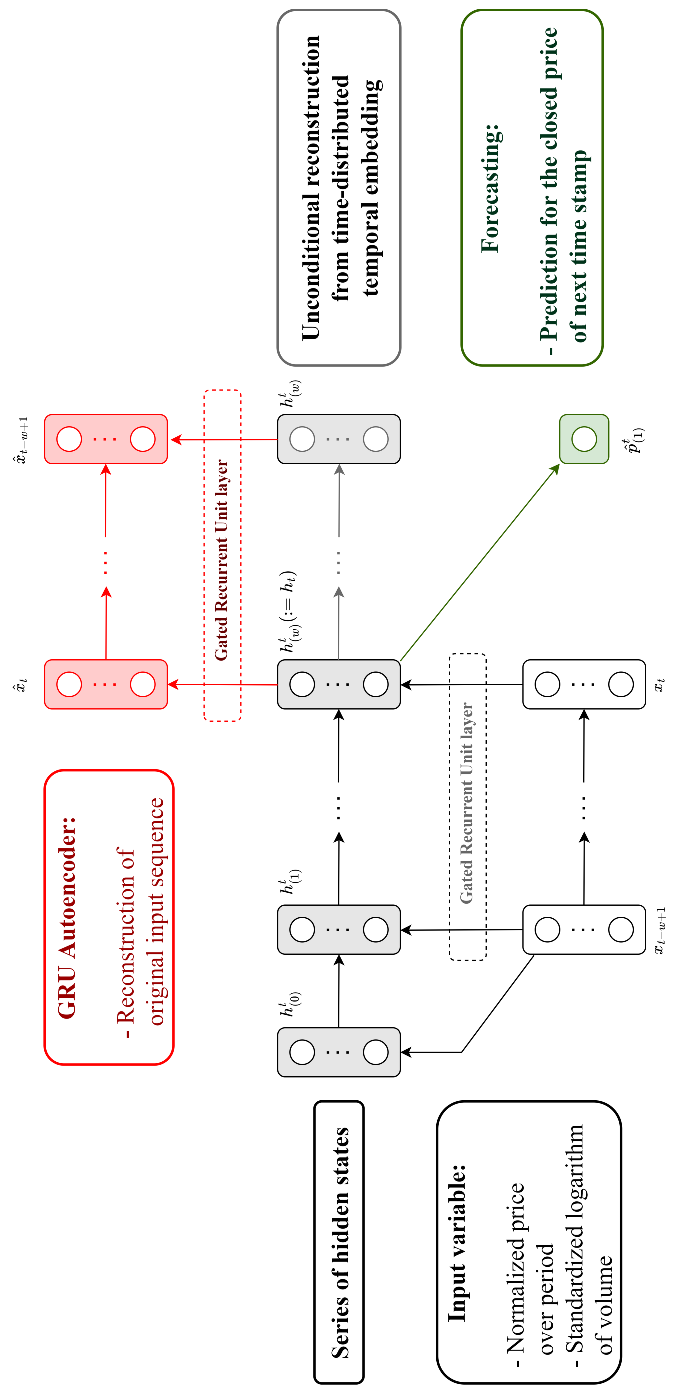

Series of hidden states
Input variable:
- Normalized price over period
- Standardized logarithm of volume
GRU Autoencoder:
- Reconstruction of original input sequence
$h^t_{(0)}$
$h^t_{(1)}$
$h^t_{(w)} (:= h_t)$
$h^t_{(w)}$
x_{t-w+1}
x_t
$\hat{x}_t$
$\hat{x}_{t-w+1}$
$\hat{p}^t_{(1)}$
Gated Recurrent Unit layer
Gated Recurrent Unit layer
Unconditional reconstruction from time-distributed temporal embedding
Forecasting:
- Prediction for the closed price of next time stamp

Figure 3.2: GRU autoencoder with hybrid loss for learning temporal encoding of subsequence

hidden sequence H_t temporally:

$$E := [e_{t,i}]^T \quad \text{where} \quad e_{t,i} = \begin{cases} \sin(t \times 10000^{\frac{-2i}{H}}) & \text{when } i \text{ is even} \\ \cos(t \times 10000^{\frac{-2i}{H}}) & \text{when } i \text{ is odd} \end{cases} \tag{3.26}$$

$$(i = 1, \cdots, H,\ t = 1, \cdots, \tau)$$

Note that $E \in \mathbb{R}^{\tau \times H}$. Using the positional encoding, we get augmented input $\bar{H}_t$ as follows:

$$\bar{H}_t := [H_t, E],\ \bar{H}_t \in \mathbb{R}^{\tau \times 2H} \tag{3.27}$$

In the self-attention mechanism, we use multi-head attention layers that are concatenated. Let ℓ be the number of heads and D be the query, key, and value dimension of each attention layer. Note that ℓ is set to be H/D in the model setting. In other words, H is ℓ multiple of D. The three weight factors: query Q^j, key K^j, and value V^j of each attention layer j are determined from the augmented embedding sequence $\bar{H}_t$ as follows:

$$Q^j := \sigma(\bar{H}_t W_Q^j + b_Q^j) \tag{3.28}$$

$$K^j := \sigma(\bar{H}_t W_K^j + b_K^j) \tag{3.29}$$

$$V^j := \sigma(\bar{H}_t W_V^j + b_V^j) \tag{3.30}$$

where $\sigma : \mathbb{R}^{\tau \times 2H} \mapsto \mathbb{R}^{\tau \times D}$ is an element-wise sigmoid function. From the weight factors, each attention $\tilde{H}_t^j \in \mathbb{R}^{\tau \times D}$ is determined using scaled-dot product as follows:

$$\tilde{H}_t^j := \text{softmax}\Big[\frac{1}{\sqrt{D}}(Q^j(K^j)^T + M)\Big]V^j \tag{3.31}$$

$$\text{where} \quad M := [m_{i,j}] \in \mathbb{R}^{\tau \times \tau},\ m_{i,j} := \begin{cases} 0 & (i \geq j) \\ -\infty & (i < j) \end{cases}$$

We obtain the multi-head attention, $\tilde{H}_t := [\tilde{H}_t{}^1, \ldots, \tilde{H}_t{}^\ell] \in \mathbb{R}^{\tau \times H}$, By passing $\tilde{H}_t$ through layer normalization, $\text{norm}(\tilde{H}_t)$ is computed. Then the residual connection with H_t, $H_t + \text{norm}(\tilde{H}_t)$, is put through fully connected neural network (FCNN), resulting in the temporally reallocated embedding sequence H'_t which is an episode sequence used for recurrent reinforcement learning.

3.3 Self-attention based deep direct reinforcement Learning with hybrid loss

3.3.1 Model Architecture

Self-attention based deep direct reinforcement Learning with hybrid loss (SA-DDR-HL) is an unified model for generating trade signals, which jointly combines RRL objective, hybrid loss of forecasting and reconstruction loss, and multi-head self-attention mechanism for sequence of temporal embedding. SA-DDR-HL consists of main network for generating temporal embedding and subnetworks branched from the main GRU decoder which is a fully connected neural network for prediction and deep recurrent reinforcement learning network. The main network using GRU, as

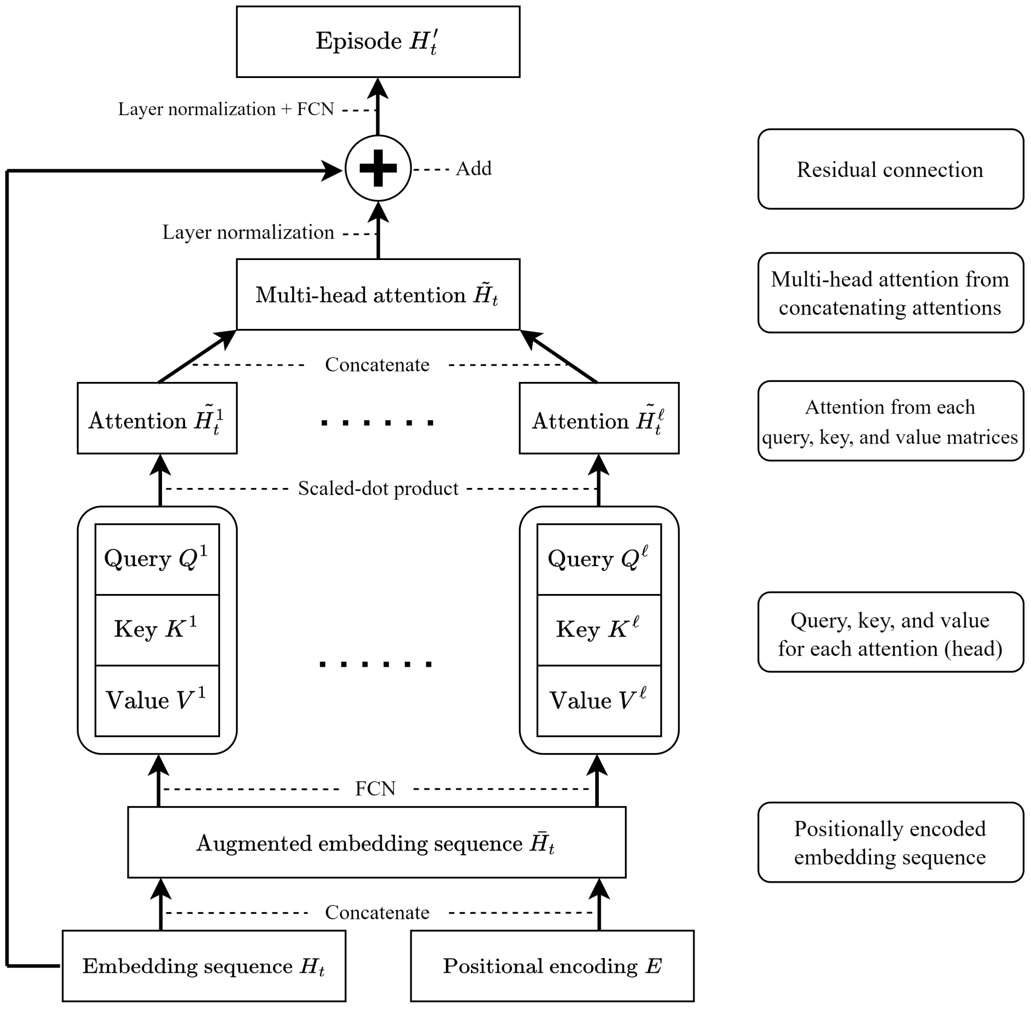

Episode H_t'
Layer normalization + FCN
Add
Residual connection
Layer normalization
Multi-head attention $\tilde{H}_t$
Multi-head attention from concatenating attentions
Concatenate
Attention $\tilde{H}_t^1$
Attention $\tilde{H}_t^\ell$
Attention from each query, key, and value matrices
Scaled-dot product
Query Q^1
Key K^1
Value V^1
Query Q^ℓ
Key K^ℓ
Value V^ℓ
Query, key, and value for each attention (head)
FCN
Augmented embedding sequence $\bar{H}_t$
Positionally encoded embedding sequence
Concatenate
Embedding sequence H_t
Positional encoding E

Figure 3.3: Self-attention structure for the sequence of temporal embedding

described in section 3.2.3, generates tth temporal embedding, h_t using raw historical data from $(t-w)$th to tth time points, $x^t_{(j)} \in \mathbb{R}^d$ $(j = 1, 2, \cdots w)$, to reflect long term dependency. Subnetworks for sequential reconstruction and forecasting in section **??** take the above temporal embedding as input variable. Sequential reconstruction of subnetworks backprogates the gradient of mean squared error (MSE) loss for reconstruction where the input sequence for temporal embedding is used as target sequence. Adjusted MSE loss is applied to adjust the variance increment from dimension. Forecasting subnetwork has the normalized close price of $(t+1)$th close price against $(t-w)$th close price, $p^t_{(1)}$ as output target and backpropagates the gradient of MSE loss for prediction. Reinforcement learning subnetwork has the batch sequence of temporal embedding as input, and the batch sequence passes through multi-head self-attention layer as shown in section 3.2.5. Self-attention layer allows for the generated trade signal to be learned from discriminated time points with more significant information among all past time points. Reinforcement learning subnetwork backpropagates the composite objective consisting of PPO surrogate objective and Sharpe ratio utility to multi-head self-attention layers. All these subnetworks jointly backpropagates each gradient to GRU encoder, the main network, according to previously determined weights.

3.3.2 Algorithm and Computational Complexity

Due to some practical reasons, it is challenging to estimate the accurate computational cost especially in deep learning [27]. From the big-O notation, which is based on sequential fixed-float computation, however, the computational complexity of sub-structures in our model could be analyzed partially by referring to

Vaswani [67]. Self-attention layer in section 3.2.5 is $O(w^2D)$ and RNN-structure layer have $O(w^2K)$ and $O(wH^2)$ complexity respectively. Note that w is the window size for generating temporal embedding. The complexity of RRL in section 3.2.1 is $O(\tau H^2)$. The RNN for GRU encoder and decoder in section **??** has the computational complexity of $O(wH^2)$. FCN for forecasting map in section **??** is $O(HH_F)$ complexity, where H_F is a dimension of intermediate layer. The complexity of each head of self-attention layer in section 3.2.5 is $O(\tau^2 D)$. We use RNN-structure as a GRU encoder, decoder, and recurrent reinforcement learning, which yields $O(wH^2)$, $O(wH^2)$, and $O(\tau H^2)$ computational complexity. Also, each head of self-attention layer has $O(\tau^2 D)$ complexity. Note that GRU feature extraction and reconstruction, FCN forecasting, and self-attention need to be duplicated layer structures due to its length of episode τ and the number of heads ℓ. The detailed training algorithm is described is in Algorithm 1.

3.3.3 Implicit Learning Scheme Along the Cases of Time Horizon

In this section, the proposed RRL model is reinterpreted in detail according to the classical value estimation point of view. As mentioned in 3.2.2, our model approximates action-value function $Q(z_t, \rho_t)$ to the immediate reward R_t and value function $V(z_t)$ to the zero. In the finite horizon case, we assume that the size of the time horizon is to be fixed to the size of an episode. For example, in the case of intraday trading, in which opening and closing times occur at fixed times, all episodes may have a fixed length. In that case, the discount factor γ is nearly 1 as the episode's length is precisely known to the agent. This may lead to the advantage $A(s_t, a_t)$ is equivalent to the difference between the action-value function and value function,

Algorithm 1 SA-DDR-HL training algorithm

```
**Input** : Times series batch set $\mathcal{B}$
**Hyperparameters** : learning rate $\epsilon$, weights for joint objective $\lambda_R,\ \lambda_F,\ \lambda_{\mathcal{L}^{RL_U}},\ \lambda_{\mathcal{L}^{RL_S}}$, max epoch $E$
(Default values are set to 0.0001, 0.1, 1, 1, 10 & 200 respectively)
Initialize parameters of temporal feature encoding map $\phi^E$, time series decoding map $\phi^D$, forecasting map $\phi^F$, hidden representation map $\phi^A$, recurrent policy function $\pi$ ($\theta_{enc},\ \theta_{dec},\ \theta_{forec},\ \theta_h,\ \theta_{\pi_U}$ & $\theta_{\pi_S}$, respectively)
1:  while current epoch $\leq E$ do
2:      for Episode $\mathbf{s} = \{s_1, \cdots, s_\tau\}$ in batch set $\mathcal{B}$ do
3:          Map episode $\mathbf{s}$ through map $\phi^E$ and get temporal embedding series $H := \phi^E(\mathbf{s}) = \{h_1, \cdots, h_\tau\}$
4:          Map temporal embedding series $H$ through time series decoding map $\phi^D$, get reconstructed series
              $\mathbf{s}' := \phi^D(H) = \{s'_1, \cdots, s'_\tau\}$, and calculate the gradient of reconstruction loss, $\Delta\mathcal{L}^{H_R}(\theta_{enc}, \theta_{dec})$
5:          Map temporal embedding series $H$ through forecasting map $\phi^F$, get the series of estimator for next close price
              $\hat{p}^C := \phi^F(H) = \{\hat{p}^1_{(1)}, \cdots, \hat{p}^\tau_{(1)}\}$, and calculate the gradient of forecasting loss, $\Delta\mathcal{L}^{H_F}(\theta_{enc}, \theta_{forec})$
6:          Map temporal embedding series $H$ through hidden representation map $\phi^A$ and get attention based hidden
              representation series of temporal embedding $H' := \phi^A(H)$
7:          Map hidden representation series $H'$ through recurrent policy function $\pi$ and get policy value series $\pi^{\mathbf{s}}$
8:          Get action series $\rho^{\mathbf{s}}$ from $\pi^{\mathbf{s}}$, get reward series $R$, and calculate the joint policy gradient of reinforcement learning
              objective, $\Delta\mathcal{L}^{RL}(\theta_{enc}, \theta_h, \theta_{\pi_U}, \theta_{\pi_S}|\hat{\theta}_{\pi_S}) := -\lambda_{\mathcal{L}^{RL_U}}\Delta\mathcal{L}^{RL_U}(\theta_{enc}, \theta_h, \theta_{\pi_U}) - \lambda_{\mathcal{L}^{RL_S}}\Delta\mathcal{L}^{RL_S}(\theta_{enc}, \theta_h, \theta_{\pi_S}|\hat{\theta}_{\pi_S})$ from $R$
9:          Update parameters of time series decoding map $\phi^D$ through $\theta_{dec} \leftarrow \theta_{dec} - \epsilon\lambda_R\Delta_{\theta_{dec}}\mathcal{L}^{H_R}(\theta_{enc},\ \theta_{dec})$
10:         Update parameters of time series forecasting map $\phi^F$ through $\theta_{forec} \leftarrow \theta_{dec} - \epsilon\lambda_F\Delta_{\theta_{forec}}\mathcal{L}^{H_F}(\theta_{enc},\ \theta_{forec})$
11:         Update parameters of reinforcement learning sub-network composed of $\phi^A$, $\pi$ through
              $(\theta_h,\ \theta_{\pi_U},\ \theta_{\pi_S}) \leftarrow (\theta_h,\ \theta_{\pi_U},\ \theta_{\pi_S}) - \epsilon\Delta_{(\theta_h,\ \theta_{\pi_U},\ \theta_{\pi_S})}\mathcal{L}^{RL}(\theta_{\pi_U},\ \theta_{\pi_S}|\hat{\theta}_{\pi_S})$
12:         Update parameters of temporal feature encoding map $\phi^E$ through
              $\theta_{enc} \leftarrow \theta_{enc} - \epsilon[\lambda_R\Delta_{\theta_{enc}}\mathcal{L}^{H_R}(\theta_{enc}, \theta_{dec}) + \lambda_F\Delta_{\theta_{enc}}\mathcal{L}^{H_F}(\theta_{enc}, \theta_{forec}) + \Delta_{\theta_{enc}}\mathcal{L}^{RL}(\theta_{enc},\ \theta_h,\ \theta_{\pi_U},\ \theta_{\pi_S}|\hat{\theta}_{\pi_S})]$
13:         current epoch $\leftarrow$ current epoch $+1$
14:     end for
15: end while
```

$Q(s_t, a_t) - V(s_t)$. Without a discount, the action-value function $Q(s_t, a_t)$ is equal to simply the sum between the immediate reward and the following value function as $R(s_t, a_t) + \mathbb{E}_\pi[\sum_{j=t+1}^{\tau} R_j | s_{t+1}] = R(s_t, a_t) + V(s_{t+1})$. Therefore the target advantage is derived as $R(s_t, a_t) + V(s_{t+1}) - V(s_t) = R(s_t, a_t) - \mathbb{E}_\pi[R(s_t, a_t)]$. Here, an expectation of immediate reward could be approximated to 0, where the distribution of policy function is centered on neutral action. It leads the target advantage $A(s_t, a_t)$ to be approximated to the immediate reward $R(s_t, a_t)$. Finally, in the finite horizon cases, the immediate reward can be used as the target advantage value by approximating the expectation of the immediate reward to 0.

On the other hand, In the infinite horizon case, a discount factor γ less than 1 is required to guarantee convergence of value function estimates. That leads the action-value function $Q(s_t, a_t)$ to be $R(s_t, a_t) + \mathbb{E}_\pi[\sum_{j=t+1}^{\tau} \gamma^{j-(t+1)} R_j | s_{t+1}] = R(s_t, a_t) + \gamma V(s_{t+1})$. Here, we impose the efficient market hypothesis that all arbitrage chances are instantly shut down as soon as they arise. In this market, there is no chance of additional guaranteed profit over the market. In particular, when the interval is short, such as daily trading, the value function can be approximated to 0 at all points in time on average. In this approximation, the target advantage becomes the immediate reward again as $V(s_t, a_t) \approx 0$ for all time point t.

As a result, the exact implicit value function estimation is possible for both cases of the finite horizon and infinite horizon for trading decisions in sufficiently short time intervals of agents who are not interested in long-term decisions. That is, under the above assumptions, the same reinforcement learning objective can be used for different trading scenarios.

3.4 Summary

In this section, we reviewed or suggested the following three. First, we reviewed reinforcement learning, which is a key framework for the proposed algorithmic trading. Value-based or policy-based reinforcement learning, a conventional reinforcement learning framework, was presented first, and recurrent reinforcement learning, a policy-based reinforcement learning optimized for trading, was reviewed. In this subsection, we introduced the recurrent reinforcement learning exploited in our model, including the value function and policy function, which are the two key elements of reinforcement learning that are the basis for reinforcement learning, and how the agent derives the optimal policy from the environment through them.

Second, we proposed a self-attention based deep direct reinforcement learning model, an algorithmic trading model based on recurrent reinforcement learning. The three main frameworks constituting each model, gated recurrent units, self-attention mechanism, and recurrent reinforcement learning model, were presented, and how these elements were exploited in our model was introduced and formulated. To impose predictive power and reconstruction power on embedded features in the feature extraction process, we proposed to use the hybrid loss of the fully-connected network structure's forecasting loss and the seq2seq structure's sequential auto-encoder loss. In addition, a self-attention mechanism was introduced prior to the input layer of recurrent reinforcement learning to rearrange the temporal importance between different time points for the episode of embedded observation. In the following experimental section, we quantitatively evaluate the impact of these additional structures on trading performance.

Finally, We present how each of the learning elements proposed above convergently generates trading signals in our model and can be learned in reverse. The proposed model was presented in algorithm form, presented in a form that can implement the proposed model step by step, and the computational complexity, which is the cost of implementation, was analyzed. Also, the proposed recurrent reinforcement learning part was analyzed in terms of the conventional reinforcement learning scheme, and how it can be interpreted has been suggested.

Chapter 4

Empirical Trading Applications on the Real-world Markets

4.1 Intra-day Trading Application on KOSPI200 Market

4.1.1 Environmental Setting

In the present study, a comprehensive intra-day trading experiment was designed and executed on the KOSPI200 market to evaluate the efficacy of the proposed hybrid model (SA-DDR-HL). The experiment considered the individual stocks listed on the KOSPI200 as samples within a single market, with all trading decisions being made in minutes. It must be noted that the experiment was designed under the premise that the opening and closing times were fixed, thus ensuring that the length of each episode was finite. This experimental design represents a practical application of the SA-DDR-HL model to a finite time horizon scenario. The results of this experiment provide valuable insights into the basic performance of the proposed model and its efficacy in real-world intra-day trading scenarios.

The KOSPI 200 market opens at 9:00 am and closes at 3:30 pm, and trading is conducted at the synchronized bidding for 10 minutes before the market closes. Thus the length of an episode can be considered to be 379, excluding the 10-minute market closing. We consider the initial 20 minutes as the burn-in period in which

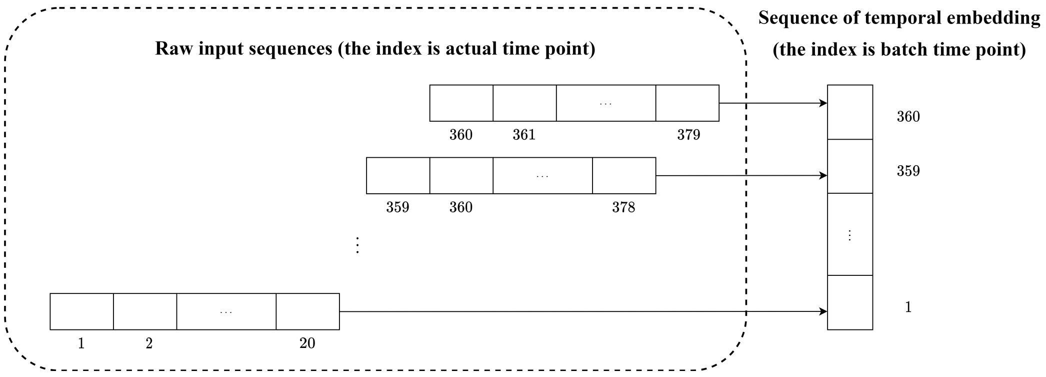

Raw input sequences (the index is actual time point)
Sequence of temporal embedding (the index is batch time point)
360
361
379
359
360
378
1
2
20
360
359
1

Figure 4.1: Schematic diagram for the batch sequence - Intra-day trading case

the observation embedding is formed and the agent actually use data from a total of 360 time points. Since the length τ of an episode is fixed to be 360, intra-day trading applications do not use a rolling window. Learning for the model proceeds for all time points, but in the case of prediction, signals are generated sequentially by partially using the obtained parameters of the self-attention layer from the start point to the corresponding point in time.

To simplify the model, we impose several circumstances or assumptions as followings. First, the price when creating the action is always viable to bid or ask. In other words, it is assumed that there is no slippage caused by the trading at the corresponding price is not made. At this time, all trading volumes are fixed at 1. Also, commission rates are charged only when a position changes and are not related to stock prices themselves. And the position is determined among long or short positions without a neutral position.

4.1.2 Data Description

The dataset used in this experiment is as follows. First, as of March 2020, minute-by-minute data of stocks incorporated into the KOSPI 200 is used. The data used here uses the 5-step asking price and volume and bidding price and volume based on the execution price and execution price at that time. Therefore, the original data has a total of 41 dimensions. the period of data is from March 25, 2020 to June 4, 2020. A list of items used in this experiment is included in appendix.

As mentioned in 4.1.1, it is assumed that a single model corresponding to all stocks in the KOSPI 200 is created. Items where trading is suspended due to excessive rise or fall, such as a circuit breaker or items where the upper or lower limit is reached, are excluded from learning and prediction. As a result, a total of 7694 samples are available for training and prediction. Of these, 20% of the data is used for testing, and 80% of the data is used for training and validation. Significant correlations between data on the same day may exist, but first of all, in this experiment, random sampling was used for all data to enhance learning.

Since the period used for the data was in a rapidly rising market, several data augmentation methodologies were used to compensate for this. First, the data in the reverse order, in which the time order and positions of bid and ask were reversed from the training data, were used for learning as synthetic data. In addition, from the original time series and the reversed time series, four-fold virtual data was created with noise added following a normal distribution with a little variance. Through this, we were able to increase the sample size of the training set to 10 times the original sample size. In addition, we tried to create a model that can respond to both rising and falling markets by removing the trend bias in learning.

In this experiment, instead of using comparisons with several models, two agents were created that followed SA-DDR-HL. The first is a normal SA-DDR-HL agent, and the second is an agent applying a dueling network that learns by separating the behavioral network and the target network. Unlike the finite horizon case, since data on tens of thousands of different samples can be used in an episode here, the other model is trained to adjust the variance of model learning, and the effect is compared to the agent without a dueling network.

The batch episode is composed of 360 minutes without a moving window, and the window size for generating the feature is 20. The price is normalized using the opening price and the volume is normalized using the volume excluding the shares of the top 3 major shareholders from the total volume and scaled from a convex function. The schematic diagram for this setting is shown in 4.1.

4.1.3 Experimental Result

The experimental results are largely divided into three categories. The first is a comparison between the general SA-DDR-HL model and B&H. The second is a comparison between the dueling agent model and B&H. Finally, we examine whether there is a significant difference in return between the two agents.

Figure 4.2 is a scatter plot comparing the return from the buy and hold strategy and the return from the SA-DDR-HL strategy. Since it is a comparison of the same item, it is possible to compare at a glance how much performance gain exists through the corresponding plot. When training the model, a transaction cost of 0.03%p was considered, so the same number of times was applied when calculating the return from the model. The points above the equal performance line of Figure 4.2 are the number of cases with model gain. As a result of the experiment, it was found that this ratio was 77.76%, indicating that there was a significant model gain. The average of this performance gap was 1.46%p. Let the performance gain between -2% and 2% be 'moderate', and the other performance gains are 'best' and 'worst', respectively. The number of items in the best group accounted for 36.93% of the total, while the number of items in the worst group only accounted for 3.06% of the total. The distribution of the performance gap can be seen in the figure 4.3.

Figure 4.4 is a scatter plot comparing the return from the buy and hold strategy and the return from the dueling SA-DDR-HL strategy. Conditions such as transaction costs are the same as the previous single SA-DDR-HL. The points above the equal performance line of Figure 4.4 are the number of cases with model gain. As a result of the experiment, it was found that this ratio was 81.34%, indicating that there was a significant model gain either. Note that there is an additional gain of 3.58%p compared to the previous figure of 77.76% for the single SA-DDR-HL model. The average of this performance gap was 1.66%p and the gap is also pretty higher than the single model case, which is about 0.2%p. The number of items in the best group accounted for 41.29% of the total, while the number of items in the worst group only accounted for 2.80% of the total. Since there is a performance gap compared to the single model, it can be confirmed that the ratio of the best group increases and the ratio of the worst group decreases. The distribution of the performance gap can be seen in the figure 4.4.

Figure 4.6 is a scatter plot comparing the performance difference between the single SA-DDR-HL model and the dueling SA-DDR-HL model. It was observed that the dueling model had a performance gain compared to the single model for about 54.61% of the total test set samples. The z-score for the average return difference between groups, which is the difference in average performance, was 6.8739, and the paired z-score for the performance gap of each sample was 4.7558. In both cases, since the p-value is less than 0.01, it can be determined that a statistically significant performance gap exists under 99%p confidence. The distribution of the performance gap can be seen in the figure 4.7.

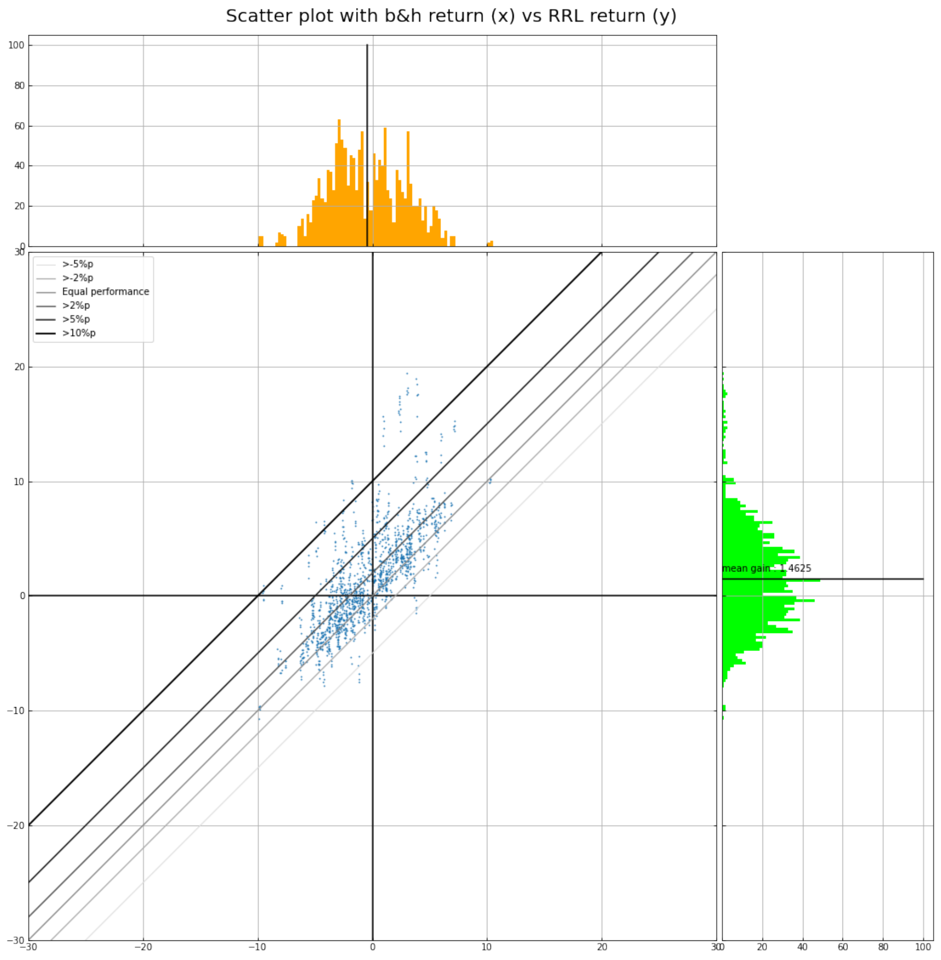


Figure 4.2: Scatter plot of log return: B&H vs SA-DDR-HL

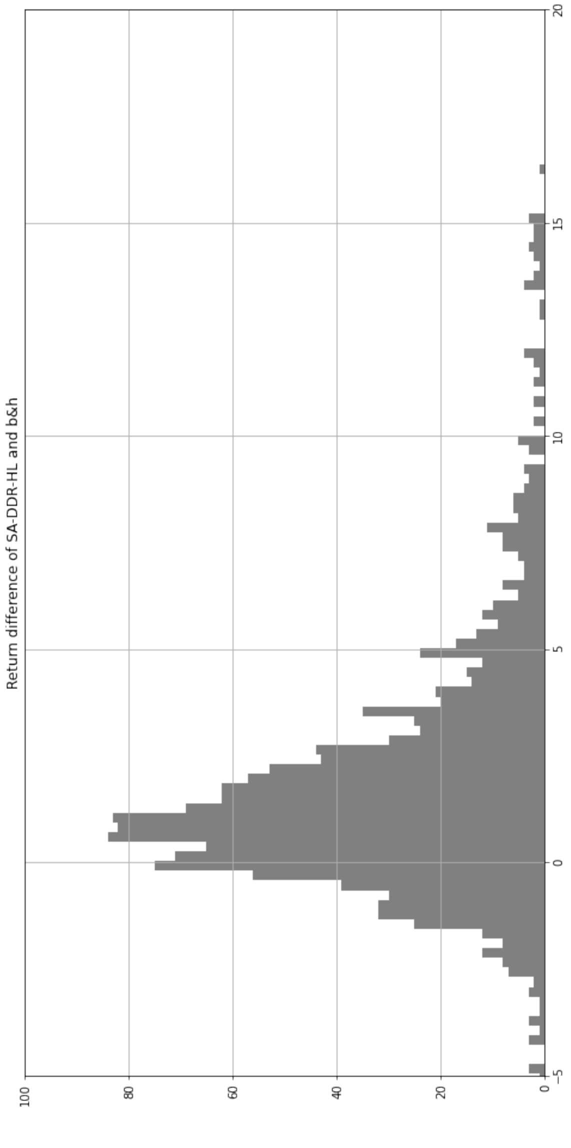


Figure 4.3: Histogram of return difference: SA-DDR-HL - B&H

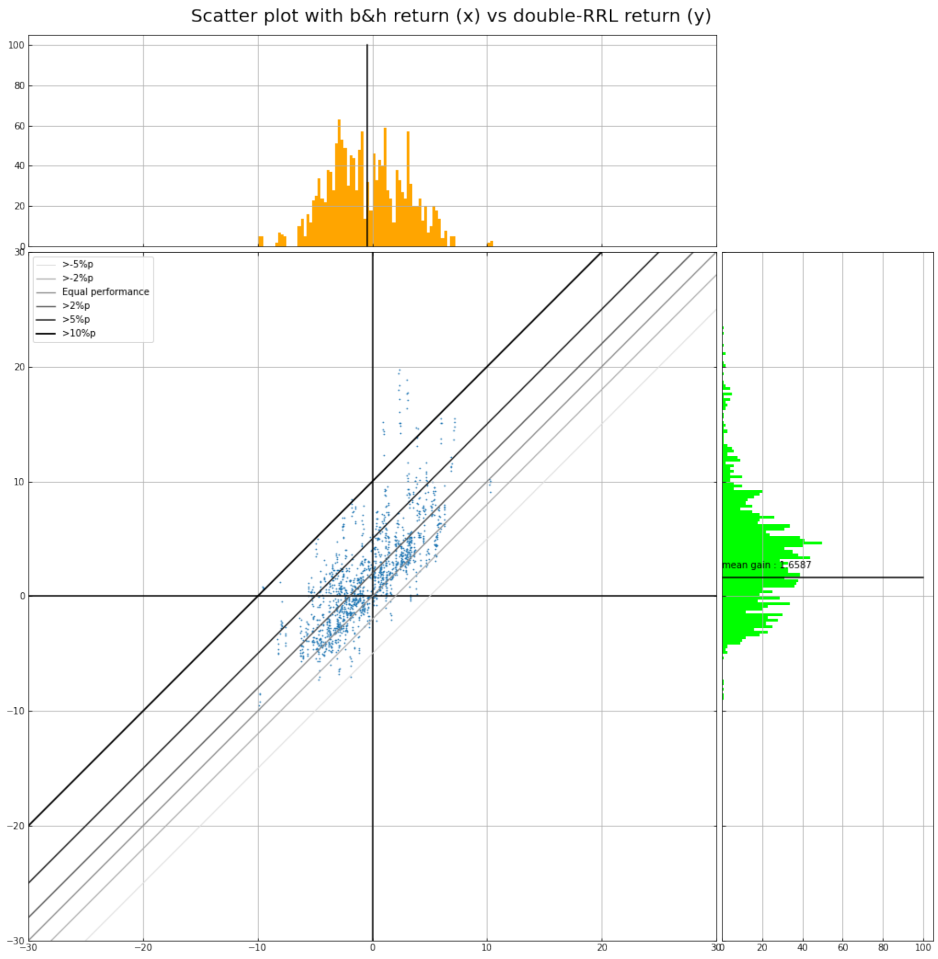


Figure 4.4: Scatter plot of log return: B&H vs dueling SA-DDR-HL

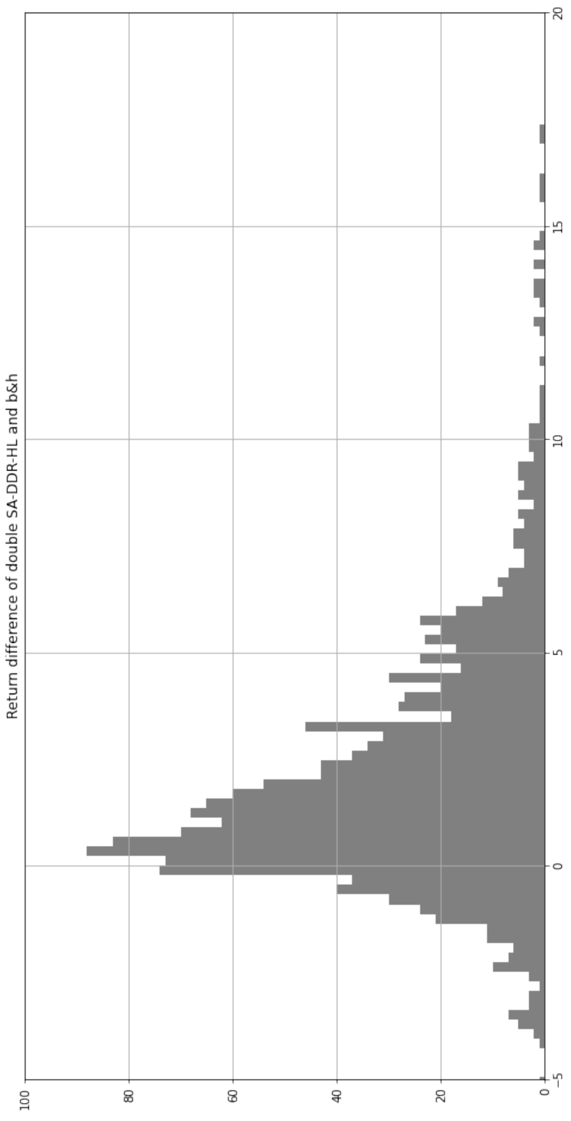


Figure 4.5: Histogram of return difference: dueling SA-DDR-HL - B&H

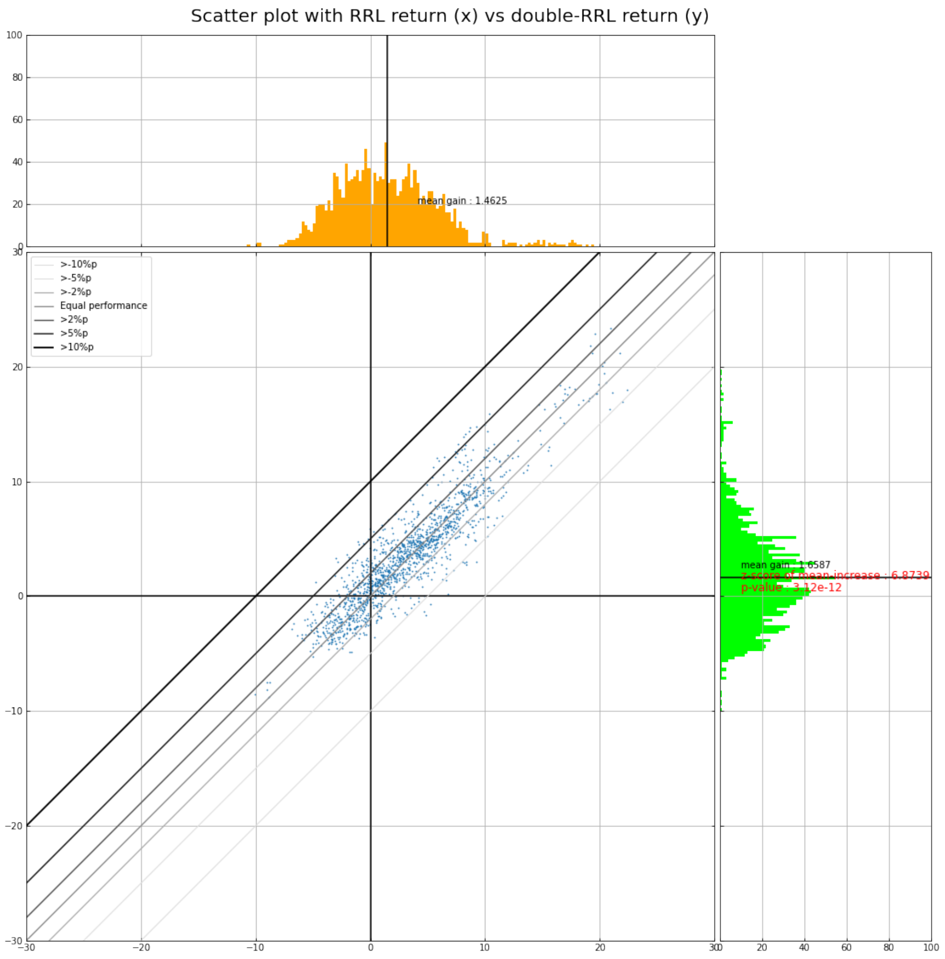


Figure 4.6: Scatter plot of log return: SA-DDR-HL vs dueling SA-DDR-HL

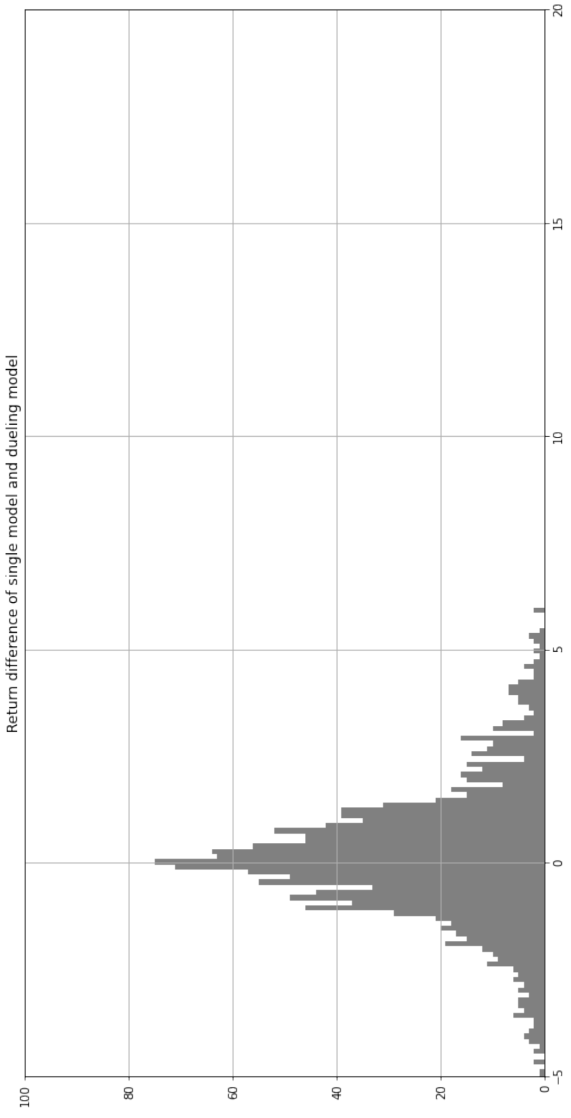


Figure 4.7: Histogram of return difference: SA-DDR-HL vs dueling SA-DDR-HL

4.1.4 Summary

In this section, we conducted an experiment to evaluate the model performance of SA-DDR-HL in the finite time horizon case. First, SA-DDR-HL showed statistically significant additional returns compared to general buy & hold strategies. Although there are aspects where some constraints or assumptions are not realistic, we showed that the proposed model can be an indicator in an actual trading environment. Second, it was shown that the network dueling technique, which separates the behavioral agent and the target agent for learning, can improve performance. It was shown that the SA-DDR-HL model using the dueling network had statistically better performance than the model without it.

4.2 Daily Trading Application on S&P 500 Market

4.2.1 Environmental Setting

In this section, a daily trading application on S&P500 market experiment was conducted to compare the performance of the model for the infinite time horizon case. Unlike the previous finite time horizon case, the infinite time horizon case requires continuous trading decisions. Therefore, the episode's length is limited to a specific interval, configured in a rolling window method, and the decision-making at the last point is set as the target policy. At this point, there is a difference from the intra-day trading model, or finite time horizon case, which uses decision-making itself as a target policy at every time point within an episode.

As with the finite time horizon case, the experiment is given the following circumstances or assumptions. First, all trades are made right before the market closes, so the reward is determined by the difference between the day's closing price and the next day's closing price. In addition, all trading volumes are fixed at 1, and commission rates are charged only for position changes, regardless of stock prices. It is assumed that there is no trade slippage and the trade is made at the closing price. Also, the position is determined among long or short positions without a neutral position.

4.2.2 Data Description

The effectiveness of our model can be verified for both individual stocks and major indices in macro range. The dataset used in this section is described as follows. For each of 11 sectors in S&P500, the stock having the largest market capital share as of 2019 is selected respectively. Major stock indices in the US stock market, S&P500, Dow Jones and NASDAQ are also used. These indices reflect the overall trend of each stock price as they consist of stock returns in proportion to the market share of stock. For both the selected stocks and the stock indices, the period of close price time series is from January 1, 2000 to January 8, 2020.

In this time series, the part prior to February 6, 2016 is used for training and validation sets, the remaining part is for test set. The time series category consists of five dimensions: open price, close price, low price, high price, and trading volume for individual stocks or transaction amount for indices. The technical trading indicators such as moving average, divergence, and strength, are not used for the model input as they may make feature selection process biased. The time series items for our model are shown in table 4.1, and the brief descriptive statistics of the items are shown in table 4.2.

The batch episode is composed of temporal embedding of 20 days. Each temporal embedding consists of raw historical data of financial time series of 20 days. The window size for generating temporal embedding, w, and the length of episode, τ, are set to be 20 respectively. The schematic diagram for this setting is shown in figure 4.10.

Table 4.1: List of items used to study

Symbol	Name	Sector
AAPL	Apple	Information Technology
AMZN	Amazon.com	Consumer Discretionary
DIS	The Walt Disney Company	Consumer Services
JPM	JPMorgan Chase	Financials
JNJ	Johnson & Johnson	Health Care
PG	Procter & Gamble	Consumer Staples
XOM	ExxonMobil	Energy
HON	Honeywell	Industrials
LIN	Linde	Materials
AMT	American Tower	Real Estate
NEE	NextEra Energy	Utilities
ˆGSPC	S&P500	Indices
ˆIXIC	NASDAQ Composite	Indices
ˆDJI	Dow Jones Industrial Average	Indices

4.2.3 Benchmark

Benchmark Algorithms

The performance comparison of SA-DDR-HL with other conventional algorithmic trading benchmark models are provided. We use the following models.

1. LSTM Classifier : Long-short term memory (LSTM) [22] is an RNN-type artificial neural network which is commonly used for sequential features. LSTM introduces recurrent hidden states with cell state and forget gate for capturing long-term dependency, and uses the normalized high dimensional time series of τ days period as an input feature. The final hidden neuron of LSTM network is the hidden representation of the input feature to predict the next close price. The prediction is done with FCN.

2. Actor-critic RL : Actor-critic is a standard framework in reinforcement learning. Actor-critic evaluates two types of network alternately: one is value function (critic) and the other is policy function (actor). The value function esti-

Table 4.2: Descriptive statistics of items used to study

	Training & Validation				Test			
	lowest	highest	average	std	lowest	highest	average	std
AAPL	0.23	33.25	8.50	9.35	22.58	75.79	41.38	11.37
AMZN	5.97	693.97	135.49	140.43	490.48	2039.51	1296.07	465.96
DIS	13.58	121.69	40.57	24.60	88.85	151.64	111.31	14.61
JPM	15.45	70.08	40.04	10.17	50.07	141.09	96.73	20.17
JNJ	34.25	109.07	66.18	16.17	101.70	148.14	128.65	10.18
PG	26.81	93.46	59.28	14.53	70.94	126.09	92.24	13.38
XOM	30.27	104.38	66.59	20.50	65.51	95.12	80.49	6.09
HON	18.30	102.30	49.49	21.35	96.62	128.01	138.01	22.51
LIN	15.84	134.67	70.78	36.01	101.79	212.90	149.25	29.22
AMT	0.71	105.01	43.09	27.65	83.67	241.07	148.13	38.15
NEE	4.60	28.69	13.51	5.86	27.86	60.68	40.04	8.78
ˆGSPC	676.53	2130.82	1334.74	324.65	1829.08	3257.85	2571.12	323.35
ˆIXIC	1114.11	5218.86	2686.92	1005.62	4266.84	9129.24	6714.24	1181.02
ˆDJI	6547.05	18312.39	11884.70	2655.97	15660.18	28868.80	22977.92	3332.10

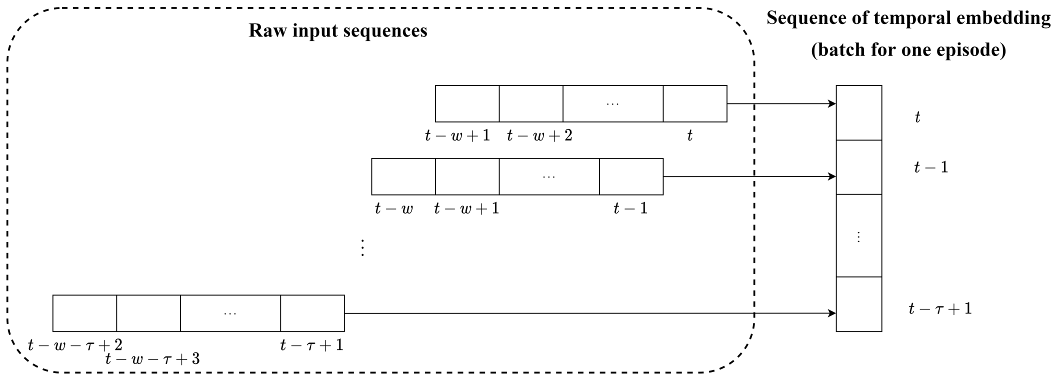


Figure 4.8: Schematic diagram for the batch sequence - Daily trading case

mates the expected sum of rewards and policy function. The policy function estimates the distribution of action value. For trading based on reinforcement learning, policy based models are generally superior to value based models such as Q-learning [37]. We use A3C (asynchronous advantage actor-critic) model with denoising auto encoder for feature extraction as a baseline actor critic model.

3. ARIMA : Autoregressive integrated moving average (ARIMA) model is a traditional nonstationary time series model. In this study, we apply the basic setting of ARIMA, ARIMA(1, 1, 1), to the close price sequence.

4. Random Forest Classifier : Random forest (RF) [21] is a basic classification model. RF is formed from bagging numerous decision tree weak classifiers. RF Model uses the normalized high dimensional time series of τ period and passes through fine tuning of hyper parameters, for example depth and the number of classifiers.

Evaluation Measure

In this section, the model performance is evaluated using total profit, annualized return, Sharpe ratio, Sortino ratio, and Maximum drawdown (MDD) as follows.

1. Total profit (TP) is the asset value of $P_\tau - P_0$. Note that P_τ is the asset value τ days after from the initial day, and P_0 is the initial day value.
2. Annualized percentage return (APR) is defined as $\left(\left(\frac{P_\tau}{P_0}\right)^{\frac{250}{\tau}} - 1\right) \times 100\%$
3. Sharpe ratio is the annualized excess return over annualized volatility, which is defined as $\sqrt{250}\frac{\bar{r}_t - r_f}{\sqrt{\sum_{t=1}^{\tau} r_t^2/\tau - (\bar{r}_t)^2}}$. Note that $r_t = \ln(P_t/P_{t-1})$ is the daily return and $\bar{r}_t = \sum_{t=1}^{\tau} r_t/\tau$ is its τ period average. In our model, we set the risk free rate, r_f, to be 0.
4. Sortino ratio, analogous to Sharpe ratio, uses downward volatility $r_{t-} := r_t 1\{r_t < 0\}$ instead of volatility, which is defined as $\sqrt{250}\frac{\bar{r}_t}{\sqrt{\sum_{t=1}^{\tau} r_{t-}^2/\tau - (\bar{r}_{t-})^2}}$.
5. Maximum drawdown (MDD) is the largest negative return from the local highest price, $\max_t\left(\max_{u \le v \le t} \frac{P_u - P_v}{P_v}\right)$. MDD can distinguish the strategy well adapted to the trend. A strategy with the less MDD is the more robust against noises.

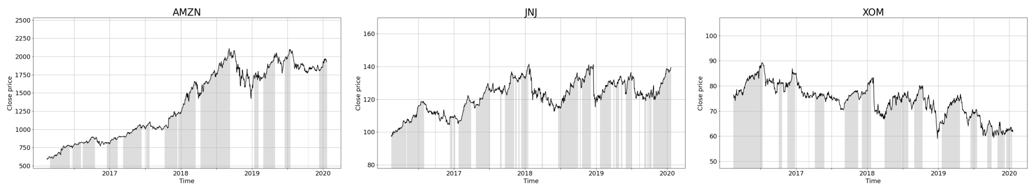


Figure 4.9: Generated trade signals from SA-DDR-HL, upon uptrend (AMZN), flat (JNJ) and downtrend (XOM) respectively

4.2.4 Experimental Result

Ablation Study

SA-DDR-HL model is a derivative model of deep recurrent reinforcement learning with hybrid loss and multi-head self-attention structure. We check whether our additional structures over plain deep recurrent reinforcement learning model improve the trading performance using the performance measures described above. The schematic diagram of the ablation study is suggested in 4.10.

According to table 4.3, all models show positive APR except for self-attention based DDR (SA-DDR) for XOM data, which has the longterm downtrend. In general, self-attention based models perform better than other models. Especially, SA-DDR-HL and SA-DDR show the better performance except for 3 cases, implying that self-attention mechanism improves DDR based algorithmic trading. When stock prices and stock indices increase steadily like AAPL, the trading performance of SA-DDR-HL is indistinct from other baseline models. Meanwhile, when a stock price fluctuates a lot, regardless of whether its trend pattern is up, flat, or down such as AMZN, JNJ, and XOM in figure 4.9, respectively, SA-DDR-HL outperforms other baseline models in trading as SA-DDR-HL properly sells stocks when their price plunge is

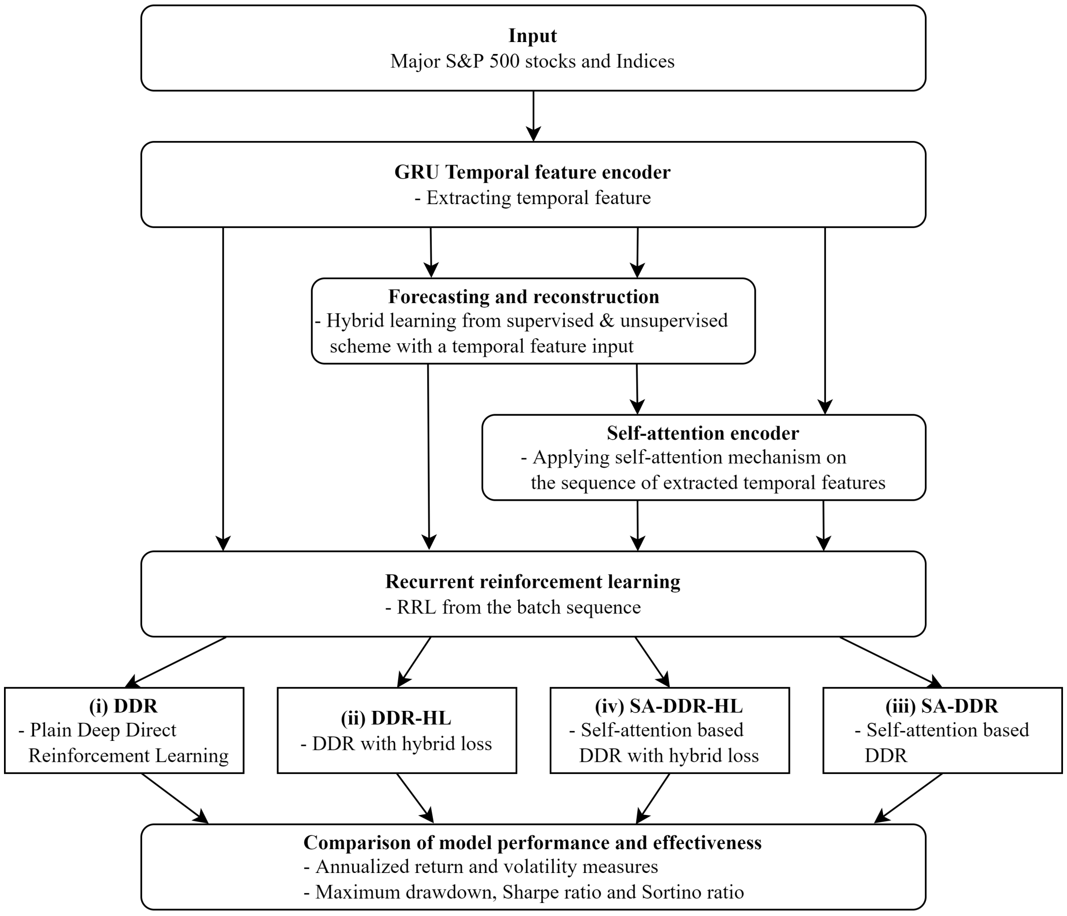

Input
Major S&P 500 stocks and Indices
GRU Temporal feature encoder
- Extracting temporal feature
Forecasting and reconstruction
- Hybrid learning from supervised & unsupervised scheme with a temporal feature input
Self-attention encoder
- Applying self-attention mechanism on the sequence of extracted temporal features
Recurrent reinforcement learning
- RRL from the batch sequence
(i) DDR
- Plain Deep Direct Reinforcement Learning
(ii) DDR-HL
- DDR with hybrid loss
(iv) SA-DDR-HL
- Self-attention based DDR with hybrid loss
(iii) SA-DDR
- Self-attention based DDR
Comparison of model performance and effectiveness
- Annualized return and volatility measures
- Maximum drawdown, Sharpe ratio and Sortino ratio

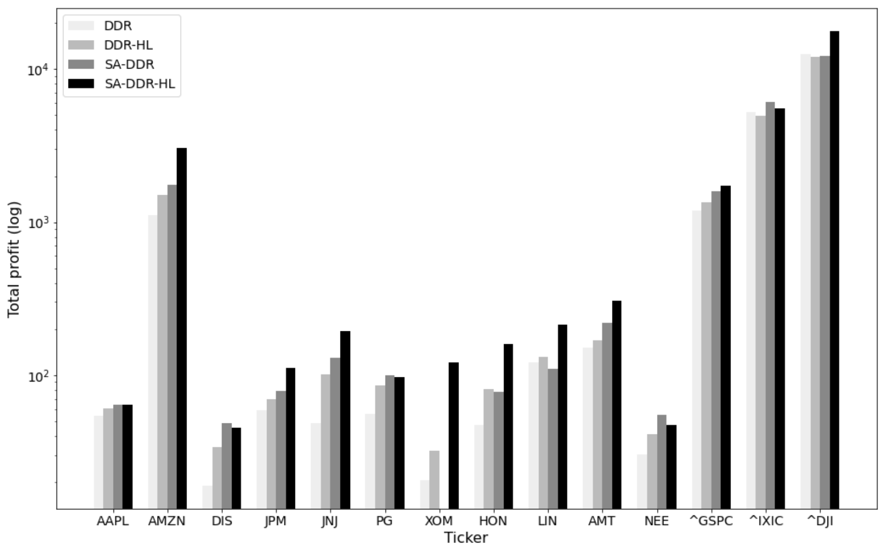
Figure 4.10: Diagram of each model in the ablation study

expected. In our model experiments, short selling has the same transaction cost rate as other general trading so that the model can have profit when market has down trend.

The performance measures in consideration of risk, Sharpe ratio (ShR), Sortino ratio (SoR), and Maximum drawdown (MDD) for SA-DDR-HL and other baseline models are provided in in table 4.4. SA-DDR-HL shows the better trading performance than other models as in table 4.3 except for SA-DDR in SoR for AAPL. In general, self-attention based DDR models perform better even considering the volatility of stock price return. SA-DDR-HL which adopts the hybrid loss structure to self-attention based DDR, shows far better result. For MDD, a downward stability index to which a risk aversion utility is applied, self-attention based models perform better excluding DDR for ˆDJI. SA-DDR-HL shows the best MDD result over half of dataset. For performance measures, APR, ShR, SoR, and MDD, the models with self-attention structure is superior to the model without the structure. In the ablation study, since SA-DDR with a self-attention layer added to DDR structure and DDR-HL with hybrid loss added to DDR structure are on average higher than DDR, it can be shown that additionally considered structure designs enhance the model effectiveness. Also, observing that the combined structure mostly marked the highest trading performance, it is reasonable to deduce that the reallocation of temporal importance and the auxiliary sub-networks to the feature extraction are exclusively trainable. To sum up, we can conclude that a hybrid learning structure for feature embedding could strengthen the performance of the self-attention based DDR model as the structure helps the embedded input sequence for DDR to have the feature extraction and prediction power.

Table 4.3: Nominal performance comparison for ablation study (TP : Total Profit, APR : Annual Percentage Rate)

	(i) DDR		(ii) DDR-HL		(iii) SA-DDR		(iv) SA-DDR-HL	
	TP	APR	TP	APR	TP	APR	TP	APR
AAPL	54.29	30.20%	60.61	32.18%	63.78	33.04%	**63.86**	**33.15**%
AMZN	1115.23	27.37%	1507.70	32.69%	1755.26	35.56%	**3057.54**	**47.56**%
DIS	18.83	4.69%	33.69	7.86%	**48.70**	**10.71**%	45.55	10.13%
JPM	58.80	18.15%	69.62	20.44%	79.21	22.31%	**110.63**	**27.62**%
JNJ	48.67	10.38%	100.43	18.14%	129.76	21.68%	**193.87**	**28.05**%
PG	55.80	14.02%	85.23	19.10%	**100.16**	**21.36**%	96.31	20.86%
XOM	20.67	6.13%	31.95	8.94%	−13.55	−5.00%	**120.92**	**24.29**%
HON	47.25	10.47%	80.56	15.91%	77.88	15.51%	**160.70**	**25.61**%
LIN	121.13	20.75%	130.75	21.86%	109.86	19.39%	**212.81**	**29.74**%
AMT	150.15	25.00%	168.83	26.92%	219.47	31.48%	**304.62**	**37.62**%
NEE	30.45	19.31%	40.89	23.58%	**54.68**	**28.31**%	46.94	25.76%
ˆGSPC	1187.99	12.54%	1353.03	13.88%	1582.88	15.64%	**1732.60**	**16.72**%
ˆIXIC	5224.13	19.76%	4970.22	19.08%	**6087.03**	**21.93**%	5527.24	20.54%
ˆDJI	12611.98	14.89%	12074.89	14.40%	12235.81	14.55%	**17701.05**	**19.08**%



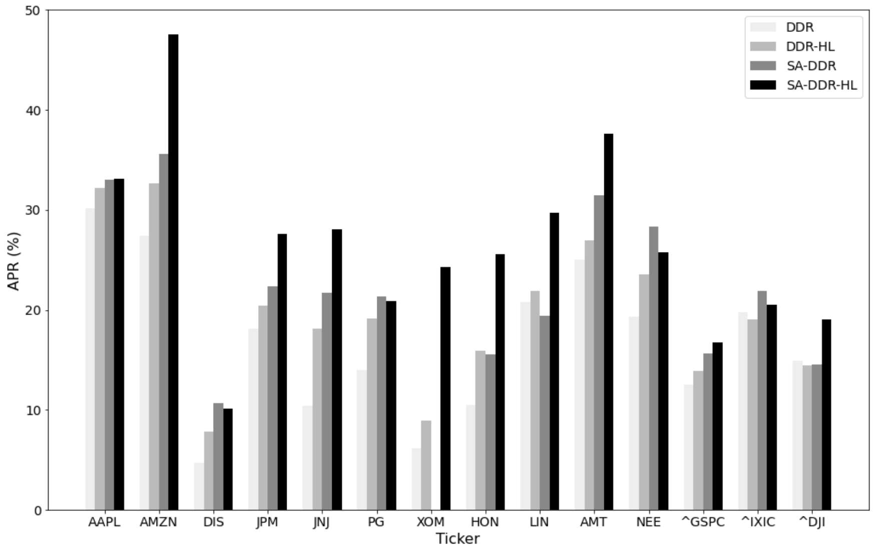
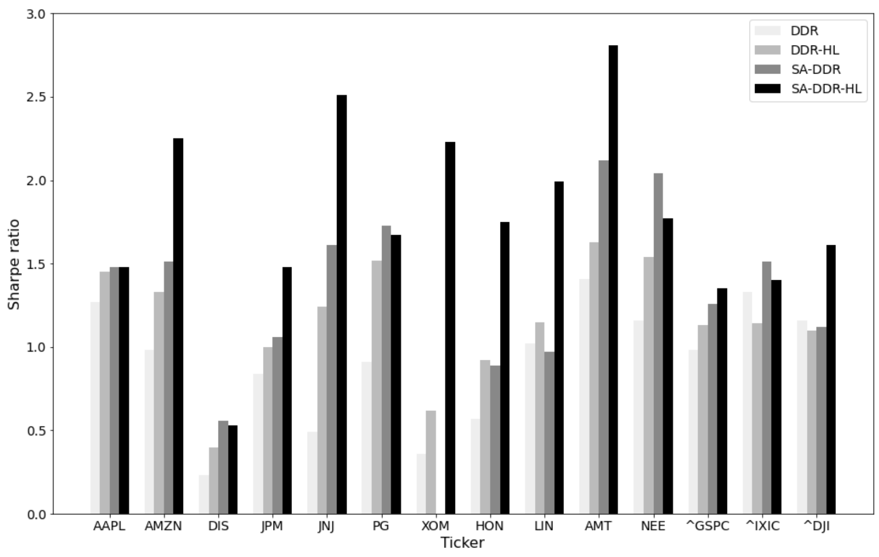


Figure 4.11: TP APR comparison in the ablation study

Table 4.4: Performance over risk comparison for ablation study (ShR : Sharpe Ratio, SoR : Sortino Ratio, MDD : Maximum Drawdown)

	(i) DDR			(ii) DDR-HL			(iii) SA-DDR			(iv) SA-DDR-HL		
	ShR	SoR	MDD	ShR	SoR	MDD	ShR	SoR	MDD	ShR	SoR	MDD
AAPL	1.27	1.64	−37.80	1.45	1.93	−21.36	1.48	**2.12**	−34.19	**1.48**	1.99	**−19.66**
AMZN	0.98	1.30	−36.48	1.33	1.79	−28.90	1.51	2.14	−26.28	**2.25**	**3.37**	**−12.45**
DIS	0.23	0.29	−18.10	0.40	0.58	−18.33	**0.56**	**0.82**	−16.63	0.53	0.79	**−16.38**
JPM	0.84	1.18	−31.15	1.00	1.41	−23.91	1.06	1.49	−25.02	**1.48**	**2.11**	**−12.87**
JNJ	0.49	0.78	−35.85	1.24	1.83	−12.14	1.61	2.62	−16.07	**2.51**	**4.62**	**−6.85**
PG	0.91	1.30	−18.99	1.52	2.61	−8.60	**1.73**	**2.99**	**−6.43**	1.67	2.60	−7.33
XOM	0.36	0.60	−17.89	0.62	1.04	−13.24	−0.26	−0.37	−43.25	**2.23**	**3.57**	**−5.02**
HON	0.57	0.72	−23.88	0.92	1.16	−23.47	0.89	1.13	−23.88	**1.75**	**2.61**	**−13.41**
LIN	1.02	1.56	−16.81	1.15	1.74	−14.55	0.97	1.46	−14.78	**1.99**	**3.42**	**−8.32**
AMT	1.41	1.98	−14.87	1.63	2.36	−18.05	2.12	3.34	−10.76	**2.81**	**5.09**	**−6.79**
NEE	1.16	1.58	−16.20	1.54	2.10	−16.44	**2.04**	**2.74**	**−11.56**	1.77	2.48	−12.03
ˆGSPC	0.98	1.15	−19.88	1.13	1.32	−18.96	1.26	1.53	**−11.32**	**1.35**	**1.61**	−11.36
ˆIXIC	1.33	1.70	−11.94	1.14	1.42	−28.67	**1.51**	**1.94**	**−10.35**	1.40	1.83	−21.24
ˆDJI	1.16	1.36	**−18.86**	1.10	1.30	−19.25	1.12	1.32	−19.13	**1.61**	**1.93**	−19.42



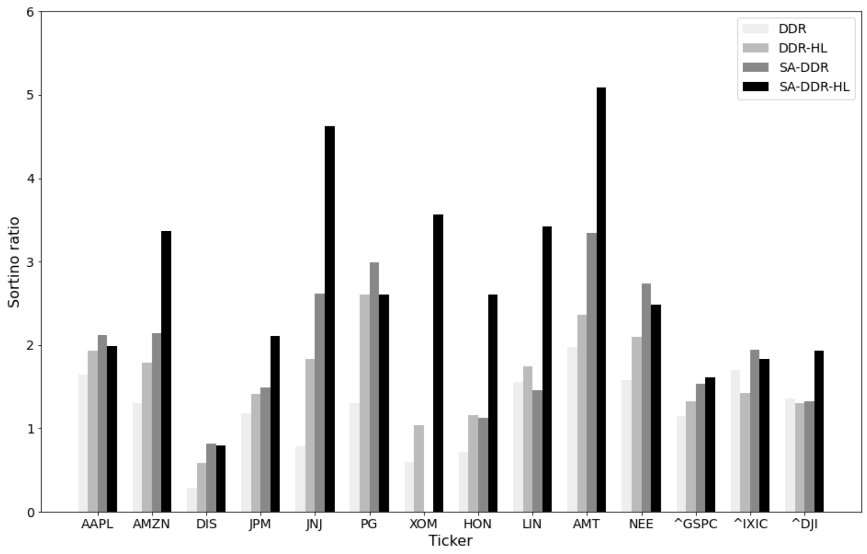


Figure 4.12: ShR SoR comparison in the ablation study

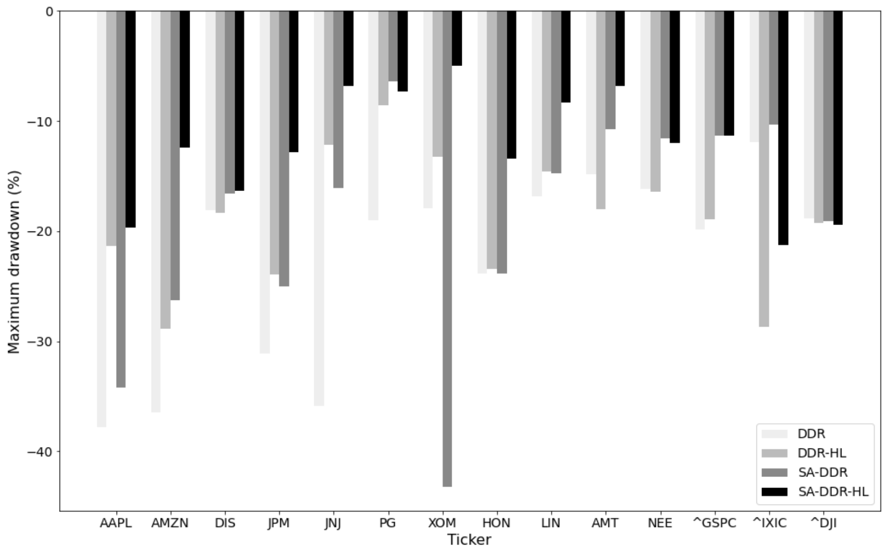


Figure 4.13: MDD comparison in the ablation study

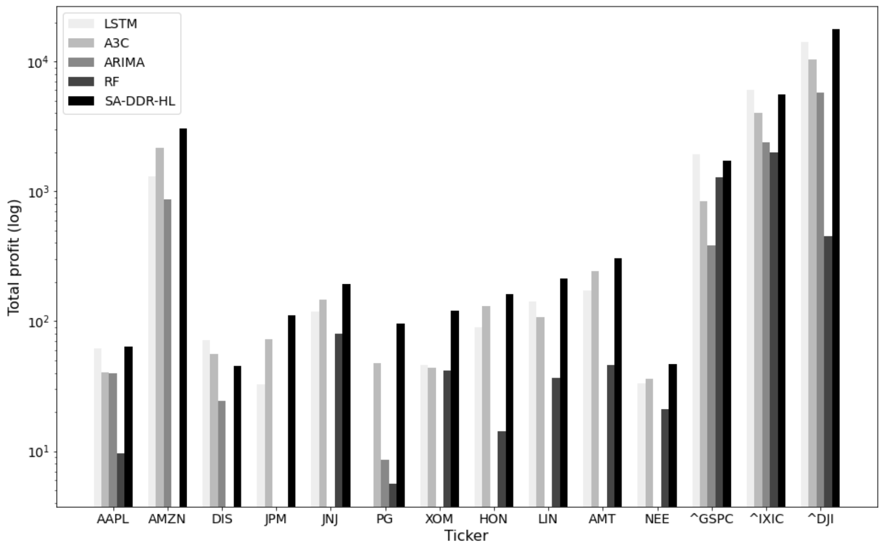
Comparative Study

The performance comparison of SA-DDR-HL with other conventional algorithmic trading benchmark models are provided. Benchmark algorithms are suggested in 4.2.3. Table 4.5 shows that the performance of SA-DDR-HL is almost the best, followed by LSTM classifier model and A3C actor-critic RL model. Classical time series model and tree model without feature engineering is inferior to neural network based models. It is difficult to distinguish the better performed model between LSTM classifier and A3C model. Meanwhile, SA-DDR-HL shows the better performance than LSTM classifier and A3C model due to its hybrid structure based on sequential recurrent neural network and reinforcement learning network.

The performance comparison result considering risk is provided in table 4.6. The models that perform better in APR still outperform in ShR or SoR. For ˆIXIC, SA-DDR-HL is slightly better than other models in ShR. However, LSTM Classifier performs better in APR exceptionally. Although SA-DDR-HL is not the best for ShR, SoR, and MDD in DIS, the performance values such as ShR and SoR in ˆGSPC, SoR and MDD in ˆIXIC, and MDD in ˆDJI, are close to the corresponding top results. For MDD in ˆIXIC and ˆDJI, A3C performs the best. Overall, the trading performance of SA-DDR-HL is better than other benchmark models based on general machine learning or time series. Also the results indicate that temporal feature extraction using both recurrent neural network and reinforcement learning improves the trading performance of the algorithmic trading using deep learning.

Table 4.5: Nominal performance comparison with other methodologies (TP : Total Profit, APR : Annual Percentage Rate)

	(i) SA-DDR-HL		(ii) LSTM Classifier		(iii) A3C		(iv) ARIMA		(v) RF	
	TP	APR	TP	APR	TP	APR	TP	APR	TP	APR
AAPL	**63.86**	**33.15**%	62.05	32.61%	40.17	25.11%	39.57	24.80%	9.61	8.56%
AMZN	**3057.54**	**47.56**%	1314.61	30.21%	2156.13	39.61%	872.86	23.91%	–	–
DIS	45.55	10.13%	**71.66**	**14.53**%	55.76	11.95%	24.39	5.92%	−25.55	−8.11%
JPM	**110.63**	**27.62**%	32.63	11.59%	72.68	21.05%	–	–	−7.61	−3.67%
JNJ	**193.87**	**28.05**%	118.14	20.34%	147.18	23.58%	−8.13	−2.23%	79.71	15.31%
PG	**96.31**	**20.86**%	−46.79	−24.01%	47.91	12.45%	8.55	2.71%	5.59	1.80%
XOM	**120.92**	**24.29**%	46.21	12.12%	43.97	11.64%	−28.18	−11.78%	41.63	11.13%
HON	**160.70**	**25.61**%	89.59	17.21%	131.21	22.45%	−59.51	−26.63%	14.25	3.63%
LIN	**212.81**	**29.74**%	142.14	22.75%	107.59	19.11%	−3.71	−1.00%	36.90	8.09%
AMT	**304.62**	**37.62**%	172.02	27.23%	244.73	33.49%	−43.03	−16.38%	46.01	10.47%
NEE	**46.94**	**25.76**%	33.11	20.46%	36.04	21.68%	−16.51	−24.44%	21.04	14.82%
ˆGSPC	1732.60	16.72%	**1941.93**	**18.16**%	837.72	9.43%	382.32	4.74%	1284.95	13.33%
ˆIXIC	5527.24	20.54%	**6052.77**	**21.95**%	4017.70	16.36%	2400.88	10.96%	2002.76	9.50%
ˆDJI	**17701.05**	**19.08**%	14120.49	16.20%	10282.31	12.71%	5755.11	7.87%	455.35	0.72%



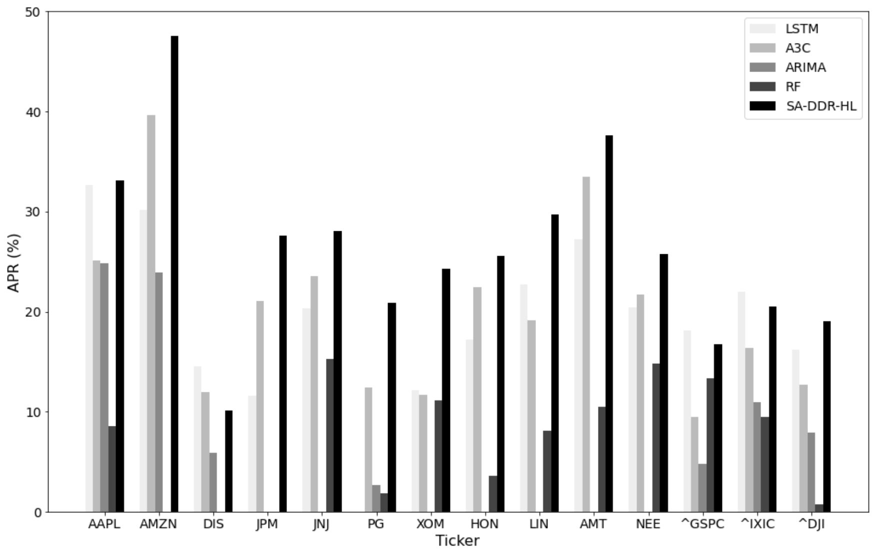
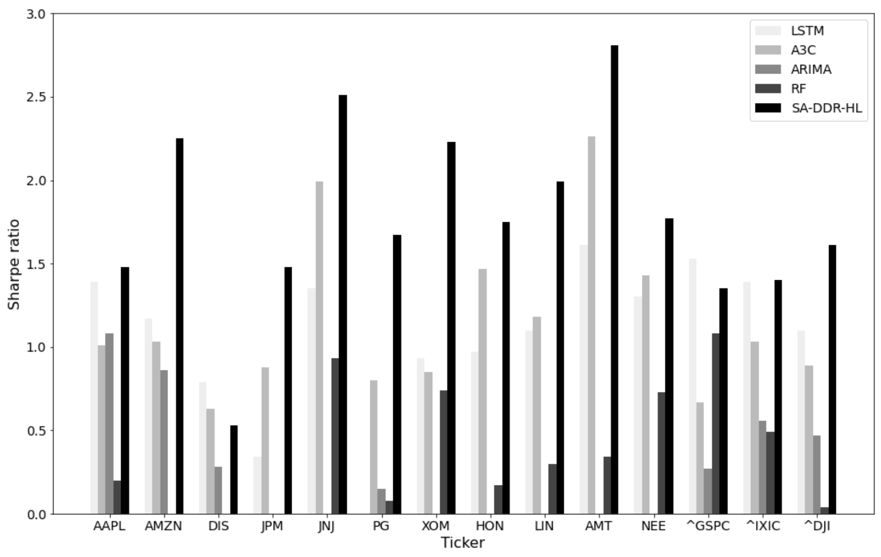


Figure 4.14: TP APR comparison in the comparative study

Table 4.6: Performance over risk comparison with other methodologies (ShR : Sharpe Ratio, SoR : Sortino Ratio, MDD : Maximum Drawdown)

	(i) SA-DDR-HL			(ii) LSTM Classifier			(iii) A3C			(iv) ARIMA			(v) RF		
	ShR	SoR	MDD	ShR	SoR	MDD	ShR	SoR	MDD	ShR	SoR	MDD	ShR	SoR	MDD
AAPL	**1.48**	**1.99**	**−19.66**	1.39	1.88	−28.26	1.01	1.41	−20.57	1.08	1.74	−23.45	0.20	0.29	−58.08
AMZN	**2.25**	**3.37**	**−12.45**	1.17	1.62	−32.99	1.03	1.28	−59.65	0.86	1.36	−35.29	–	–	–
DIS	0.53	0.79	−16.38	**0.79**	**1.16**	**−16.38**	0.63	0.93	−22.40	0.28	0.40	−28.43	−0.25	−0.29	−58.77
JPM	**1.48**	**2.11**	**−12.87**	0.34	0.50	−44.39	0.88	1.42	−19.82	–	–	–	−0.08	−0.11	−71.23
JNJ	**2.51**	**4.62**	**−6.85**	1.35	2.03	−16.06	1.99	3.13	−10.29	−0.10	−0.14	−30.76	0.93	1.41	−11.07
PG	**1.67**	**2.60**	**−7.33**	−0.84	−1.12	−64.84	0.80	1.11	−12.96	0.15	0.23	−17.85	0.08	0.11	−38.19
XOM	**2.23**	**3.57**	**−5.02**	0.93	1.51	−23.09	0.85	1.19	−19.17	−0.40	−0.52	−67.88	0.74	1.13	−21.26
HON	**1.75**	**2.61**	**−13.41**	0.97	1.34	−18.70	1.47	2.47	−13.51	−0.62	−0.74	−73.69	0.17	0.23	−24.88
LIN	**1.99**	**3.42**	**−8.32**	1.10	1.67	−19.32	1.18	1.92	−19.83	−0.03	−0.04	−48.01	0.30	0.48	−35.62
AMT	**2.81**	**5.09**	**−6.79**	1.61	2.18	−13.81	2.26	3.65	−12.39	−0.31	−0.41	−66.88	0.34	0.47	−36.46
NEE	**1.77**	**2.48**	**−12.03**	1.30	1.80	−16.60	1.43	1.95	−17.53	−0.58	−0.80	−73.10	0.73	1.05	−18.77
ˆGSPC	1.35	1.61	**−11.36**	**1.53**	**1.98**	−11.81	0.67	0.92	−17.74	0.27	0.35	−32.95	1.08	1.33	−14.22
ˆIXIC	**1.40**	1.83	−21.24	1.39	**1.92**	−19.60	1.03	1.30	**−15.17**	0.56	0.78	−23.85	0.49	0.63	−24.60
ˆDJI	**1.61**	**1.93**	−19.42	1.10	1.75	−19.69	0.89	1.05	**−13.90**	0.47	0.74	−26.02	0.04	0.05	−34.47



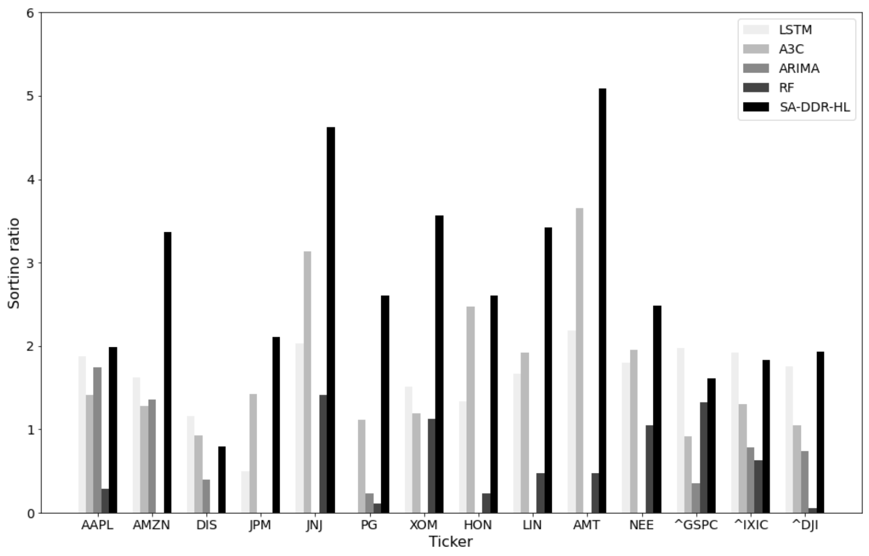


Figure 4.15: ShR SoR comparison in the comparative study

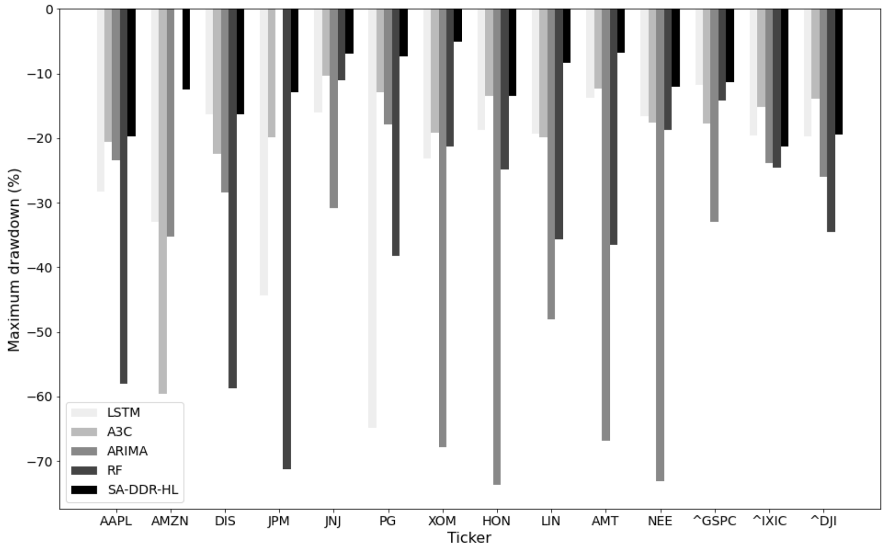


Figure 4.16: MDD comparison in the comparative study

Sensitivity Analysis on the Cost Factor

In order to find out the difference in performance according to the main factors of the model, in this section, we will scrutinize the changes in trading frequency and return according to the cost factor that determines the reward. In this section, we use SA-DDR-HL models for items with three different trends, AMZN, JNJ, and XOM, as illustrated in Figure 4.9. The cost factor was changed by 0.01%p centered on the default value of 0.03%, and 5 values from 0.01% to 0.05% were used.

The table 4.7 provided herein presents trading frequency observed during the designated test period, alongside the nominal return, TP, and APR. One crucial facet under consideration is the effect of high transaction costs on trading activity. As anticipated, elevated transaction costs are found to discourage frequent transactions, and, thus, an expectation was posited to observe a reduction in the transaction frequency. However, upon meticulous examination of the table, it is evident that the actual model performance does not manifest a discernible trend in response to varying transaction frequencies. The table 4.7 reveals that the number of transactions remains relatively constant, contrary to initial expectations.

The examination of the nominal return, in light of transaction costs, also reveals an intriguing absence of a specific trend. That is, the performance of transactions does not exhibit substantial disparities either such as frequencies. When the generated trading signals demonstrate similarity or exhibit a subset relationship, their propensity towards nominal return is noticeably influenced by the cost factor. This observation suggests that the very essence of the generated trading signal undergoes significant modifications due to the interplay with the cost factor. The conclusion drawn is that the cost factor within the reward setting engenders considerable vari-

Table 4.7: Sensitibity analysis on cost factor c

	c	0.0001	0.0002	0.0003	0.0004	0.0005
Trading Freq.	AMZN	27	29	26	30	27
	JNJ	32	42	26	24	38
	XOM	33	31	37	39	31
TP	AMZN	3677.16	3523.51	3057.54	2652.51	2879.82
	JNJ	167.02	175.06	193.87	190.5	161.91
	XOM	126.18	119.3	120.92	121.2	110.73
APR	AMZN	55.90%	54.31%	47.56%	42.98%	47.14%
	JNJ	21.20%	23.19%	28.05%	26.83%	21.05%
	XOM	29.73%	28.10%	28.49%	28.56%	26.07%
ShR	AMZN	2.8	2.57	2.25	2.18	1.99
	JNJ	1.83	2.09	2.51	2.44	1.81
	XOM	3.05	2.66	2.23	2.59	2.12
SoR	AMZN	4.11	3.87	3.37	3.4	3.1
	JNJ	3.23	3.64	4.62	4.51	2.99
	XOM	3.71	3.18	3.57	3.3	2.57
MDD	AMZN	-12.74	-11.81	-12.45	-18.21	-23.1
	JNJ	-9.24	-6.47	-6.85	-5.55	-12.82
	XOM	-6.49	-6.81	-5.02	-9.88	-7.92

ation in the trading signals, each striving to achieve optimal utility. Consequently, it becomes evident that there exists a seldom correlation among the distinct trading signals, thereby highlighting the complexity and multifaceted nature of their responses to the cost factor.

4.2.5 Summary and Discussion

In this section, we conducted two experiments to evaluate the model performance of SA-DDR-HL in the infinite time horizon case. First, as an ablation study, in this experiment, we observed the difference in model performance between when each model component was excluded and when it was not. n this experiment, the proposed model SA-DDR-HL showed the best performance, and the self-attention-based DDR model with hybrid loss removed: SA-DDR, showed the second-best performance. This experiment showed that introducing the self-attention mechanism to the RRL structure significantly contributes to the improvement of the trader agent's model performance. Second, as a comparative study, in this experiment, we compared the performance of the proposed model with a commonly used prediction algorithm or actor-critic reinforcement learning, and observed whether there was a significant performance difference. In this experiment, the proposed model either showed relatively the best performance, and the LSTM-based prediction model showed the second-best performance. A3C, an actor-critic based reinforcement learning model, did not show clearly good performance. Unlike the LSTM classifier, which structurally reflects and predicts the time dependency, the time dependency inside the feature could not be reflected in the feature engineering stage in the A3C agent. This seems to cause the size of the feature dimension to be too large compared to the sample size.

In this experiment, an experiment was performed to compare the performance of the proposed SA-DDR-HL model through evaluation with various models, but unlike the ablation study, a more advanced trading agent or forecasting model may exist as a comparison group in the comparative study. Although it is not possible to evaluate all possible control groups, this experiment is meaningful in that it suggests a trader agent with superior performance compared to algorithms commonly used in daily trading.

Chapter 5

Conclusion

5.1 Contributions and Limitations

Financial time series is one of the highly noisy data. Thus among many application fields of data science, the problem of predicting the financial market is one of the most challenging problems. In financial time series, it is difficult to distinguish between signal and noise, and it is difficult to find suitable features because they have long-term dependencies. In the conventional general sense, algorithmic trading is just automated trading execution from given rules, and it was essential to develop such rules from traders' insight. However, many challenges have been to automating such a trading scheme or rule itself. The most widely used models are regression or classification models in the automated trading decision process. Still, financial time series predictions based on classification models do not reflect the risk or volatility of the time series, are not accurate enough, and are trend dependent, making it difficult to create meaningful models. In addition, in the decision making, it was challenging to concrete the environment closer to that of actual traders, such as fees, positions, and assets, other than direct forecasting.

This dissertation proposed a trader agent that introduced reinforcement learning to overcome the limitations of such prediction models. Generally, reinforcement learning based on the Markov Decision Process is unsuitable for financial time series problems with long-term dependencies. Still, several attempts have been made to solve this by considering the agent's state or structural recurrence. In order to overcome this difficulty, we proposed a model based on recurrent reinforcement learning that introduces recurrence into action. From the basic recurrent reinforcement learning model, additional submodel elements were introduced by exploiting the high model complexity characteristics of deep learning. These additional elements have been mainly used in existing financial time series prediction or reinforcement learning models. In this model, we tried to overcome the inherent disadvantages of the model by effectively combining additional elements. For example, a self-attention mechanism is employed to allocate relative importance between transaction signals to an agent, and generative and predictive models were partially utilized in the feature extraction process so that the agent's embedded observation compactly fully contains information about the sequence.

In addition, two experiments were conducted to confirm that the proposed elements actually operate effectively. First is the finite horizon case, where there is a certain degree of performance gain on average compared to the basic buy hold for the intra-day trading scenario, and the results were explained by directly analyzing the time series according to the relative results of the model. Second, in the infinite horizon case, an ablation study for a daily trading scenario and a comparative analysis for other models were conducted. Through two experiments, we argued that our model has a practical impact. In particular, it was shown that the augmented elements compared to the basic model, actually contributed to improving the model's performance. Not only was the proposed model experimentally verified, but an algorithm was also developed for the practical implementation of the model. The model was also interpreted and presented from a general reinforcement learning point of view.

On the other hand, this dissertation's model and experimental environment also have the following limitations. First, there are several limitations to model simplification. First of all, the no slippage cost assumption, in which trade is always possible at the observed price, is somewhat unrealistic, and this part needs to be further considered, especially in the case of intra-day trading. Second, the fact that decision-making is too simple compared to trading in the real world can also be considered a disadvantage. When holding a product, the decision-making for that product is not limited to simply long or short, but factors such as quantity or neutral position should be considered. In particular, considering the portfolio's composition, it is inevitable that decision-making regarding trading will be somewhat regulated by currently available assets. However, the proposed model did not reflect such a variety of positions for simplification of the model.

Also, there are some limitations in terms of the actual usability of the model in computation. For example, since the feature extraction and reinforcement learning parts depend on the recurrence structure, the overall computational complexity during learning is significant, as presented in this paper. For this reason, it can be pointed out that the computational cost can be high compared to relatively simple models, which could be a trade-off.

5.2 Future Works

One of the possible studies for future work is to reconstruct the trader's environment more realistically. Many of the limitations of the proposed model in the previous chapter can be solved to some extent by properly modifying the environment. For example, the no slippage assumption can be solved by giving a negative reward when an order is not placed at the observed price. Also, the environment can be modified for asset allocation or portfolio selection problems. The agent's assets or cash can be included in the observation vector, and it can be formulated as a problem of distributing portfolio weight by considering multiple items for observation simultaneously. In the proposed model, the agent's state does not reflect the distribution according to the action, but better performance can be expected if appropriate interaction with the environment is added as above.

Also, as an RRL-based model, there is room for improving the structure more efficiently. As described above, the proposed model has a disadvantage: the learning time is long because it is learned from multiple recurrence layers during learning. Therefore, a model that can replace or simplify the GRU structure used for feature extraction can be considered.

Bibliography

[1] A. A. Adebiyi, A. O. Adewumi, and C. K. Ayo, *Comparison of arima and artificial neural networks models for stock price prediction*, Journal of Applied Mathematics, 2014 (2014).

[2] B. Alhnaity and M. Abbod, *A new hybrid financial time series prediction model*, Engineering Applications of Artificial Intelligence, 95 (2020), p. 103873.

[3] G. Appel, *Technical analysis: power tools for active investors*, FT Press, 2005.

[4] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, arXiv preprint arXiv:1409.0473, (2014).

[5] W. Bao, J. Yue, and Y. Rao, *A deep learning framework for financial time series using stacked autoencoders and long-short term memory*, PloS one, 12 (2017), p. e0180944.

[6] A. Borovykh, S. Bohte, and C. W. Oosterlee, *Conditional time series forecasting with convolutional neural networks*, arXiv preprint arXiv:1703.04691, (2017).

[7] G. E. Box, W. H. Hunter, S. Hunter, et al., *Statistics for experimenters*, vol. 664, John Wiley and sons New York, 1978.

[8] E. Chan, *Quantitative trading: how to build your own algorithmic trading business*, vol. 430, John Wiley & Sons, 2009.

[9] K. Chen, Y. Zhou, and F. Dai, *A lstm-based method for stock returns prediction: A case study of china stock market*, in 2015 IEEE international conference on big data (big data), IEEE, 2015, pp. 2823–2824.

[10] E. Chong, C. Han, and F. C. Park, *Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies*, Expert Systems with Applications, 83 (2017), pp. 187–205.

[11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, arXiv preprint arXiv:1412.3555, (2014).

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

[13] Y. Du, *Application and analysis of forecasting stock price index based on combination of arima model and bp neural network*, in 2018 Chinese Control And Decision Conference (CCDC), IEEE, 2018, pp. 2854–2857.

[14] R. Engle, *Garch 101: The use of arch/garch models in applied econometrics*, Journal of economic perspectives, 15 (2001), pp. 157–168.

[15] D. Enke and N. Mehdiyev, *Stock market prediction using a combination of stepwise regression analysis, differential evolution-based fuzzy clustering, and a*

fuzzy inference neural network, Intelligent Automation & Soft Computing, 19 (2013), pp. 636–648.

[16] C. Francq and J.-M. Zakoian, *GARCH models: structure, statistical inference and financial applications*, John Wiley & Sons, 2019.

[17] R. Fried, *Robust filtering of time series with trends*, Journal of Nonparametric Statistics, 16 (2004), pp. 313–328.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning. book in preparation for mit press*, URL¡ http://www. deeplearningbook. org, 1 (2016).

[19] V.-S. Ha, D.-N. Lu, G. S. Choi, H.-N. Nguyen, and B. Yoon, *Improving credit risk prediction in online peer-to-peer (p2p) lending using feature selection with deep learning*, in 2019 21st International Conference on Advanced Communication Technology (ICACT), IEEE, 2019, pp. 511–515.

[20] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, in International Conference on Machine Learning, PMLR, 2018, pp. 1861–1870.

[21] T. K. Ho, *Random decision forests*, in Proceedings of 3rd international conference on document analysis and recognition, vol. 1, IEEE, 1995, pp. 278–282.

[22] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural computation, 9 (1997), pp. 1735–1780.

[23] J. J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities.*, Proceedings of the national academy of sciences, 79 (1982), pp. 2554–2558.

[24] C. Y. Huang, *Financial trading as a game: A deep reinforcement learning approach*, arXiv preprint arXiv:1807.02787, (2018).

[25] B. Hurst, Y. H. Ooi, and L. H. Pedersen, *A century of evidence on trend-following investing*, The Journal of Portfolio Management, 44 (2017), pp. 15–29.

[26] A. J. Hussain, A. Knowles, P. J. Lisboa, and W. El-Deredy, *Financial time series prediction using polynomial pipelined neural networks*, Expert Systems with Applications, 35 (2008), pp. 1186–1199.

[27] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, *Predicting the computational cost of deep learning models*, in 2018 IEEE international conference on big data (Big Data), IEEE, 2018, pp. 3873–3882.

[28] R. E. Kalman, *A new approach to linear filtering and prediction problems*, (1960).

[29] V. Konda and J. Tsitsiklis, *Actor-critic algorithms*, Advances in neural information processing systems, 12 (1999).

[30] M. Kumar and M. Thenmozhi, *Forecasting stock index returns using arima-svm, arima-ann, and arima-random forest hybrid models*, International Journal of Banking, Accounting and Finance, 5 (2014), pp. 284–308.

[31] D. Kwak, S. Choi, and W. Chang, *Self-attention based deep direct recurrent reinforcement learning with hybrid loss for trading signal generation*, Information Sciences, 623 (2023), pp. 592–606.

[32] E. Lecarpentier and E. Rachelson, *Non-stationary markov decision processes, a worst-case approach using model-based reinforcement learning, extended version*, arXiv preprint arXiv:1904.10090, (2019).

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

[34] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, *Independently recurrent neural network (indrnn): Building a longer and deeper rnn*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 5457–5466.

[35] X. Li, L. Li, J. Gao, X. He, J. Chen, L. Deng, and J. He, *Recurrent reinforcement learning: a hybrid approach*, arXiv preprint arXiv:1509.03044, (2015).

[36] Y. Li, *Deep reinforcement learning: An overview*, arXiv preprint arXiv:1701.07274, (2017).

[37] Y. Li, W. Zheng, and Z. Zheng, *Deep robust reinforcement learning for practical algorithmic trading*, IEEE Access, 7 (2019), pp. 108014–108022.

[38] Z. Li, G. Liu, and C. Jiang, *Deep representation learning with full center loss for credit card fraud detection*, IEEE Transactions on Computational Social Systems, 7 (2020), pp. 569–579.

[39] B. Lim, S. Ö. Arik, N. Loeff, and T. Pfister, *Temporal fusion transformers for interpretable multi-horizon time series forecasting*, International Journal of Forecasting, 37 (2021), pp. 1748–1764.

[40] Y. Liu, Q. Liu, H. Zhao, Z. Pan, and C. Liu, *Adaptive quantitative trading: an imitative deep reinforcement learning approach*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 2020, pp. 2128–2135.

[41] W. Long, Z. Lu, and L. Cui, *Deep learning-based feature engineering for stock price movement prediction*, Knowledge-Based Systems, 164 (2019), pp. 163–173.

[42] L. Menkhoff, *The use of technical analysis by fund managers: International evidence*, Journal of Banking & Finance, 34 (2010), pp. 2573–2586.

[43] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, *Asynchronous methods for deep reinforcement learning*, in International conference on machine learning, PMLR, 2016, pp. 1928–1937.

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing atari with deep reinforcement learning*, arXiv preprint arXiv:1312.5602, (2013).

[45] B. Moews, J. M. Herrmann, and G. Ibikunle, *Lagged correlation-based deep learning for directional trend change prediction in financial time series*, Expert Systems with Applications, 120 (2019), pp. 197–206.

[46] P. Mondal, L. Shit, and S. Goswami, *Study of effectiveness of time series modeling (arima) in forecasting stock prices*, International Journal of Computer Science, Engineering and Applications, 4 (2014), p. 13.

[47] J. Moody and M. Saffell, *Reinforcement learning for trading*, Advances in Neural Information Processing Systems, 11 (1998).

[48] ——, *Learning to trade via direct reinforcement*, IEEE transactions on neural Networks, 12 (2001), pp. 875–889.

[49] D. M. Nelson, A. C. Pereira, and R. A. de Oliveira, *Stock market's price movement prediction with lstm neural networks*, in 2017 International joint conference on neural networks (IJCNN), IEEE, 2017, pp. 1419–1426.

[50] Y. P. Pane, S. P. Nageshrao, J. Kober, and R. Babuška, *Reinforcement learning based compensation methods for robot manipulators*, Engineering Applications of Artificial Intelligence, 78 (2019), pp. 236–247.

[51] W. Pei, T. Baltrusaitis, D. M. Tax, and L.-P. Morency, *Temporal attention-gated model for robust sequence classification*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 6730–6739.

[52] J. M. Poterba and L. H. Summers, *Mean reversion in stock prices: Evidence and implications*, Journal of financial economics, 22 (1988), pp. 27–59.

[53] A. Pumsirirat and L. Yan, *Credit card fraud detection using deep learning based on auto-encoder and restricted boltzmann machine*, International Journal of advanced computer science and applications, 9 (2018), pp. 18–25.

[54] N. F. Ryman-Tubb, P. Krause, and W. Garn, *How artificial intelligence and machine learning research impacts payment card fraud detection: A survey and industry benchmark*, Engineering Applications of Artificial Intelligence, 76 (2018), pp. 130–157.

[55] T. A. Schmitt, D. Chetalova, R. Schäfer, and T. Guhr, *Non-stationarity in financial time series: Generic features and tail behavior*, EPL (Europhysics Letters), 103 (2013), p. 58003.

[56] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, *Trust region policy optimization*, in International conference on machine learning, PMLR, 2015, pp. 1889–1897.

[57] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, arXiv preprint arXiv:1707.06347, (2017).

[58] A. F. Serban, *Combining mean reversion and momentum trading strategies in foreign exchange markets*, Journal of Banking & Finance, 34 (2010), pp. 2720–2727.

[59] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, *Financial time series forecasting with deep learning: A systematic literature review: 2005–2019*, Applied Soft Computing, 90 (2020), p. 106181.

[60] O. B. Sezer and A. M. Ozbayoglu, *Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach*, Applied Soft Computing, 70 (2018), pp. 525–538.

[61] N. Srivastava, E. Mansimov, and R. Salakhudinov, *Unsupervised learning of video representations using lstms*, in International conference on machine learning, PMLR, 2015, pp. 843–852.

[62] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, Advances in neural information processing systems, 27 (2014).

[63] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[64] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al., *Policy gradient methods for reinforcement learning with function approximation.*, in NIPs, vol. 99, Citeseer, 1999, pp. 1057–1063.

[65] R. S. Tsay, *Testing and modeling threshold autoregressive processes*, Journal of the American statistical association, 84 (1989), pp. 231–240.

[66] M. R. Vargas, C. E. dos Anjos, G. L. Bichara, and A. G. Evsukoff, *Deep learning for stock market prediction using technical indicators and financial news articles*, in 2018 International Joint Conference on Neural Networks (IJCNN), IEEE, 2018, pp. 1–8.

[67] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, arXiv preprint arXiv:1706.03762, (2017).

[68] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, *Extracting and composing robust features with denoising autoencoders*, in Proceedings of the 25th international conference on Machine learning, 2008, pp. 1096–1103.

[69] J.-H. Wang and J.-Y. Leu, *Stock market trend prediction using arima-based neural networks*, in Proceedings of International Conference on Neural Networks (ICNN'96), vol. 4, IEEE, 1996, pp. 2160–2165.

[70] W. Wang and K. K. Mishra, *A novel stock trading prediction and recommendation system*, Multimedia Tools and Applications, 77 (2018), pp. 4203–4215.

[71] J. W. Wilder, *New concepts in technical trading systems*, Trend Research, 1978.

[72] R. J. Williams and D. Zipser, *Experimental analysis of the real-time recurrent learning algorithm*, Connection science, 1 (1989), pp. 87–111.

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

[74] S. Yang, X. Yu, and Y. Zhou, *Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example*, in 2020 International workshop on electronic communication and artificial intelligence (IWECAI), IEEE, 2020, pp. 98–101.

국문초록

심층신경망(Deep neural network)의 발달로 인해 복잡한 데이터를 다룰 수 있게 됨에 따라, 심층신경망 구조를 활용한 알고리듬 트레이딩은 단순히 기계학습의 응용분야 중 하나에 그칠 뿐 아니라 학문적 차원에서 발달하고 있다. 머신러닝을 이용한 알고리듬 트레이딩의 핵심은 다음과 같이 두 가지이다. 첫째로는 금융시계열로부터 시장 가격의 변동을 설명할 수 있는 특성을 추출하는 것이고, 둘 째로는 다차원의 금융시계열로부터 특성들 간의 시간적 인과성을 찾아내는 것에 있다. 금융시계열은 노이즈가 강한 특성이 있기 때문에, 금융시계열 자체로부터 직접 예측에 필요한 유의미한 특성을 뽑아내는 것이 어렵다. 이 때문에 그동안의 알고리듬 트레이딩은 주로 사전적 지식(domain knowledge)을 이용하여, 금융시계열로부터 특성 변수를 직접 추출 또는 가공하는 방법론(feature engineering)이 주요하게 연구되었다. 또한 알고리듬 트레이딩의 이윤 극대화를 직접적으로 학습의 목적함수에 이용하는 대신, 방향 분류 모델 또는 가격에 대한 회귀 모델 등과 같은 지도학습 모델을 통해 간접적으로 최적의 포지션을 예측하는 경우가 많았다. 그러나 지도학습을 이용한 방법은 포지션에 의한 효용을 학습에 충분히 반영하지 못하고, 예측 결과로부터 간접적인 의사결정을 수행하기 때문에 이윤극대화라는 목적에 대해 이중 오차가 발생한다. 이와 같은 문제를 극복하기 위해, 주어진 환경에서의 관측으로부터 기대보상을 최대화하는 행동을 학습하는 강화학습 기반의 알고리듬 트레이딩 모델이 지도학습 기반의 모델의 단점을 극복하는 좋은 대안이 되고 있다.

본 논문에서는 순환강화학습과 셀프 어텐션 구조를 결합한 융합 모델을 제안하고, 주식 시장에서의 거래 신호를 발생하는 실증적으로 제안된 모델의 유용함을 검증한다. 먼저 제안된 모델의 네트워크 구조 및 학습에 필요한 환경, 보상, 상태집합, 그리고

에이전트의 행동집합을 정의한다. 제안된 모델은 서로 다른 시점에서 생성된 거래 신호들 간의 시간적 의존성(temporal dependency)를 반영한 정책 그라디언트 모델인 심층 순환강화학습 구조를 기반으로 한다. 심층 순환강화학습은 에피소드(episode)내의 각 시점에서의 관측치를 정책함수의 입력으로 갖고 이전의 거래 신호에 대해 순환적(recurrent)으로 결정되는 거래 신호를 생성하고, 생성된 거래 신호로부터 얻는 보상인 수익률들에 대한 효용함수를 극대화하도록 정책함수의 파라미터를 학습하는 정책 기반(policy-based) 강화학습 모형이다. 여기에 지도학습에 해당하는 예측모델 및 비지도학습에 해당하는 생성모델 오차를 전파하는 하이브리드 인공신경망과 은닉 표현 시퀀스의 시간적 중요도를 학습하는 셀프 어텐션 계층을 접목하였다. 또한 제안된 모델의 강화학습 과정을 상세하게 기술하기 위해, 시계(Time horizon)의 범위가 한정이 되어있는 경우와 한정이 없는 경우 두 상태 모두에 대해서 모델을 정의한다. 이 때, 시계의 범위가 한정이 되어있지 않은 경우라도 단기 변동에 대해 위험중립적 선호, 즉 매수와 매도에 대한 정책의 사전적인(prior) 위험 중립 확률이 1대 1에 가깝다고 근사하는 경우에는 시계의 범위가 한정되어있는 모델과 동일한 이익(Advantage)로부터 학습이 가능함을 보인다.

다음으로, 모델 성능을 검증하는 두 가지 응용 실험을 수행하고 그 결과를 분석한다. 첫 번째로 유한 시간 순환강화학습 알고리듬 트레이딩 모델의 응용으로, 코스피200 상장 종목에 대한 데이 트레이딩(Intraday trading)에 적용한 실험을 수행한다. 유한 시간 순환강화학습 알고리듬 트레이딩 모델의 응용 실험에 사용된 데이터는 2019년 3월부터 6월까지의 총 40일 간의 KOSPI200 등재 종목의 분 단위의 시가, 종가, 고가, 저가, 거래량이며, 이 중 거래정지, VI, 제한가 등 외부적인 시장 거래 개입이 발생한 종목을 제외한 종목들에 대해 실험을 수행하였다. 실험은 전체 종목 데이터를 훈련 데이터와 검증 데이터, 그리고 시험 데이터로 나누어 시장 전체 종목에 대한 데이 트레이딩을 수행하는 단일 모델을 생성하였으며, 추세 효과 또는 종목 간 동조 효과를 줄이기 위해

데이터 증강(Data augmentation)을 사용하여 훈련 데이터를 다양화하였다. 두 번째로 무한 시간 순환강화학습 알고리듬 트레이딩 모델의 응용으로, S&P 500 상장 종목에 대한 일일 거래에 적용한 실험을 수행한다. 무한 시간 순환강화학습 알고리듬 트레이딩 모델의 응용 실험에 사용된 데이터는 2000년 1월부터 2019년 11월까지의 총 5000일 간의 개별 S&P500 등재 종목으로, 이를 훈련 데이터와 검증 데이터, 그리고 시험 데이터로 나누어 개별 종목 별로 모델을 생성하였다. 해당 기간에 존속했던 기업 중 11개의 섹터 별로 말일 기준 시가총액이 가장 큰 기업들에 대해 개별적으로 모델을 학습 및 시험을 수행하였다.

두 실험에서 공통적으로 머신러닝 기반의 지도학습 모델인 LSTM, Random Forest와 강화학습 모델인 A3C 모델, 그리고 시계열 모델인 ARIMA와의 모델 성능을 비교한다. 또한 무한 시간 순환강화학습 알고리듬 트레이딩 모델 실험에서는 제안된 모델의 구조적 효과를 검증하기 위해, 모델의 구성요소가 되는 하부 학습 구조를 모델에서 제거하여 제안된 융합 강화학습 모델의 각 요소가 성능 향상에 영향을 끼치는지를 검증한다. 결과적으로, 우리의 모델이 일반적인 머신러닝 모델 또는 기본적인 강화학습 모델에 비해 명목 수익률, 리스크 대비 수익률 등의 수익률 평가 지표 측면에서 더욱 높은 성능을 가짐을 제시한다. 또한 모델 하부 학습 구조들에 의한 모델효과 검증 결과를 통해, 강화학습 입력 시퀀스에 도입한 셀프 어텐션 인코더 구조와, 피 지도학습-비지도학습 융합 학습 모델이 제안된 모델의 성능 향상에 유기적으로 도움을 주는 요소가 됨을 제시한다.

주요어: 순환강화학습, 셀프 어텐션 메카니즘, 순차적 오토인코더, Gated Recurrent Unit, 알고리듬 트레이딩, 데이 트레이딩

학번: 2016-21095

감사의 글

처음 졸업논문을 작성하면서 감사의 글을 논문 작업의 최후의 최후에 쓰려고 마음먹었는데, 감사의 글을 정말로 쓰게 되는 이 순간이 와서 감개무량합니다. 만으로도 7년이 넘는 긴 시간동안 수많은 감사한 분들과 함께했고, 이 글을 쓰는 시간은 그 분들과의 소중했던 순간 하나 하나를 꺼내볼 수 있는 소중한 시간인 것 같습니다.

우선 가장 먼저 부족한 저를 보듬어주시고 졸업까지 이끌어주신 지도교수님인 장우진 교수님께 감사의 말씀을 드립니다. 다소 개인적인 일이지만, 전세사기를 당해서 당장 주거가 막막했던 순간에 교수님께서 직접 손 내밀어 지낼 곳을 마련해주셨던 기억이 떠오릅니다. 당시 주변에 이 얘기를 할 때면, 대학원 괴담(?)만 알고있던 주변 사람들이 하나같이 세상에 그런 교수님이 어딨냐며 놀라워했던 기억이 떠오릅니다. 논문지도 때는 거의 매일같이, 저자인 저보다도 디테일하게 신경써주시면서, 주도적으로 논문을 작성해본 경험이 일천한 저에게 귀중한 도움들을 주셨던 기억 또한 떠오릅니다. 어렸을 때 아버지가 돌아가셔서 아버지의 사랑이란 것이 어떤것인지 느끼지 못하고 살아왔는데, 아마도 교수님께서 저에게 보내주신 헌신이 이와 가장 비슷하지 않을까하는 생각이 듭니다. 또 그만큼 스스로의 부족함에 굉장히 마음속으로 죄송스러웠던 적도 있었습니다. 이 자리를 빌어서 무한한 감사의 말씀을 전합니다. 또한, 2012년부터 지금까지 산업공학과에 다니면서 많은 가르침을 주신 서울대학교 산업공학과 교수님들, 과사무실 선생님들께도 감사드리며, 특별히 부족한 저의 논문에 아낌없는 조언을 주셨던 심사위원장 이덕주 교수님 및 이재욱 교수님, 박우진 교수님께 따로 감사의 말씀을 전해드리고 싶습니다.

다음으로는 긴 시간동안 저의 연구실 생활을 같이 했던 선배님들께 감사의 말씀을 드립니다. 먼저 제가 학자로써 존경하고 경외감을 느끼는, 지금까지 우리 연구실에 많은 도움을 주고계신 송재욱 교수님께 저 또한 굉장히 많은 도움을 받은것 같습니다. 이

자리를 빌어 무한한 감사의 말씀 전합니다. 지금까지도 많은 연락을 하고있는 박지환 선배님, 김선도 선배님, 구승모 선배님, 그리고 이창주 선배님까지 89 선배님들께도 사람 만들어주셔서(?) 감사한다는 말씀 전해드립니다. 곧 2세가 생기는 김선도, 구승모 선배님 2세 건강하게 나와서 세상에서 행복한 가장이 되시길 기원하며, 박지환 선배님도 조만간 좋은 소식 기대하고 있겠습니다. 사업을 위해 열심히 발로 뛰고계신 이창주 선배님, 지금도 잘 되고 있지만 앞으로 더욱 번창하시길 선배님의 충성스러운 심복으로써 진심으로 기원합니다. 조풍진, 이민혁 91 선배님이자 교수님, 제가 연구실에서 한 때 매일같이 봐왔고 논문으로 고민하는 모습까지 옆에서 지켜봤었는데, 가까운 선배님들 중에 교수님이 생겼다는 점이 새삼 신기하였습니다. 두 분 다 존경받는 교수님이 될 수 있을 것이라 믿어 의심치 않습니다. 저와 대학원 동기이지만 저보다 2년 먼저 졸업하신 최성윤 형님, 입학은 동시에 했지만 아무것도 모르는 제가 형님의 뒤를 따라서 후배처럼 이것저것 여러가지 많이 여쭈어보기도 하고 굉장한 의지가 되었습니다. 이 자리를 빌어서 감사의 말씀 드립니다. 혹시 추가로 논문 쓰실 계획이 있으시다면, 서로 또 도울 수 있는 기회가 된다면 좋겠습니다.

그리고 지금 저와 같이 연구실 생활을 하고있는 금융리스크공학연구실 멤버들에게도 격려과 감사의 인사를 전하고 싶습니다. 논문과 사투중인 박사 후보 정윤이, 현주, 그리고 기영이까지, 매일같이 노력하고있는 만큼 좋은 결실 얻을 수 있을것이라고 생각합니다. 다들 제가 아는 모든 사람들 중에서도 손에 꼽는 똑똑하고 유능한 인재들이라 제가 특별히 걱정이 안될 정도로 잘 해내고 있고, 잘 마칠 수 있을거라고 생각합니다. 저와 졸업동기(?)인 지훈이와 제승이 곧 졸업 축하하고, 앞으로 원하는 대로 일 잘 풀리길 바랍니다. 연구실 두 번 입학한 종우와, 영봉이, 가장 최근에 들어온 한서와 재원이까지 저처럼 오래 헤매지 않고 순탄하게 졸업까지 이르기를 진심으로 기원합니다. 저보다 먼저 졸업한 자랑스러운 석사 후배 도현이, 준열이, 우혁이, 루이도 현재 하고있는 학업 또는 일 모두 순탄하게 잘 풀리길 바랍니다.

그리고, 저와 제가 걸어가고 있는 길을 믿어주고 12년의 기나긴 학교생활을 옆에서

묵묵히 지원해준 우리 가족들, 감사하고 사랑합니다.