IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 17, 2019, accepted July 29, 2019, date of publication August 2, 2019, date of current version August 19, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2932789

Deep Robust Reinforcement Learning for Practical Algorithmic Trading

YANG LI[1,2], WANSHAN ZHENG[1,2], AND ZIBIN ZHENG[1,3]

[1]School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510275, China
[2]Guangdong Key Laboratory for Big Data Analysis and Simulation of Public Opinion, Sun Yat-sen University, Guangzhou, China
[3]National Engineering Research Center of Digital Life, Sun Yat-sen University, Guangzhou, China

Corresponding author: Zibin Zheng (zhzibin@mail.sysu.edu.cn)

This work was supported in part by the National Key Research and Development Program under Grant 2016YFB1000101, in part by the National Natural Science Foundation of China under Grant 61722214 and Grant U1811462, in part by the Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme under Grant 2016, and in part by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant 2016ZT06D211.

ABSTRACT In algorithmic trading, feature extraction and trading strategy design are two prominent challenges to acquire long-term profits. However, the previously proposed methods rely heavily on domain knowledge to extract handcrafted features and lack an effective way to dynamically adjust the trading strategy. With the recent breakthroughs of deep reinforcement learning (DRL), sequential real-world problems can be modeled and solved with a more human-like approach. In this paper, we propose a novel trading agent, based on deep reinforcement learning, to autonomously make trading decisions and gain profits in the dynamic financial markets. We extend the value-based deep Q-network (DQN) and the asynchronous advantage actor-critic (A3C) for better adapting to the trading market. Specifically, in order to automatically extract robust market representations and resolve the financial time series dependence, we utilize the stacked denoising autoencoders (SDAEs) and the long short-term memory (LSTM) as parts of the function approximator, respectively. Furthermore, we design several elaborate mechanisms to make the trading agent more practical to the real trading environment, such as position-controlled action and n-step reward. The experimental results show that our trading agent outperforms the baselines and achieves stable risk-adjusted returns in both the stock and the futures markets.

INDEX TERMS Algorithmic trading, Markov decision process, deep neural network, reinforcement learning.

I. INTRODUCTION

Algorithmic trading is a valuable topic in the financial market and has been widely discussed in modern artificial intelligence. For both institutional investors and individual investors, there is a strong demand in exploring autonomous trading algorithms that are adaptable to the dynamic trading market. However, mainstream methods for learning to trade have longstanding challenges as follows: (1) the difficulty in extracting effective market representations, and (2) the difference between classification (up or down prediction) and directly learning trading strategies (direct trading). According to the different approaches for market modeling, previous studies can be roughly categorized into three types: *traditional financial analysis*, *machine learning (ML) approaches*, and *deep learning (DL) approaches*. In *traditional financial analysis*, mathematics is wildly adopted to recognize historical time series patterns and make predictions [1]. The common models include autoregressive moving average (ARMA) model [2] and generalized autoregressive conditional heteroskedasticity (GARCH) model [3]. ARMA model contains autoregressive (AR) [4] and moving average (MA) [5]. Its generalization, AR-integrated MA (ARIMA) [6], becomes a popular method for time series analysis in economics. GARCH model is frequently used for asset pricing, risk management, and volatility forecasting. In the *machine learning approaches*, [7] models the high-frequency limit order book using support vector machine (SVM) with handcrafted features and shows the effectiveness in the real-world data. Reference [8] predicts the direction of stock market prices

The associate editor coordinating the review of this manuscript and approving it for publication was Bora Onat.

with random forest (RF) and shows that the model is robust in predicting the future direction of the stock movement. Reference [9] [10], [11] also reveal the ability of market modeling. In more details, [9] shows that SVM outperforms the back propagation (BP) neural network in financial forecasting, and there is comparable generalization performance between SVM and the regularized RBF neural network. Reference [10] shows that the neural network is able to extract useful information from a huge data set and data mining is also able to predict future trends and behaviors. Reference [11] shows that the neural networks is able to predict both single-dimensional data and multi-dimensional data which are extracted from financial time series. With the development of *deep learning approaches*, recurrent neural network (RNN) [12] is specifically designed to extract temporal information from raw sequential data. RNN variations, such as long short-term memory (LSTM) [13] and gated recurrent unit (GRU) [14] networks, have been proposed to mitigate the gradient vanishing problem and achieve state-of-the-art results in a variety of sequential data prediction problems [15]–[17] shows that the convolutional neural network (CNN) is better suited for predicting the price movements of stocks than multilayer neural networks and support vector machines. Reference [18] proposes a temporal attention-augmented bilinear network architecture that combines bilinear projection and attention mechanism, which demonstrates good results.

Although the aforementioned methods demonstrate good accuracy in the market modeling and tendency classification, they are not robust to the dynamic real market and can not be directly applied to algorithmic trading. The financial time series contains a large amount of noise, including the manipulation of large investors, the impact of news and notices, the uncertain trading behaviors of investors, and so on. All these noises lead to the highly non-stationary of financial time series, which decrease the generalization capability of the model. Moreover, there exists a handcrafted conversion of mapping the market prediction to the trading action in strategies, such as buy, sell and hold. The trading strategy is a kind of complex sequential decision-making problem that includes many components in the field of practical trading. For example, prediction accuracy is just one of the strategy metrics and doesn't play a decisive role in the trading period. If the accuracy of prediction is high but the profit and loss (P&L) is lower than 1, profit is negative in this case because the strategy is likely to gain little money in the correct prediction but lose a lot in the wrong prediction. Meanwhile, risk management and portfolio management are also critical components in practical trading, which lead to a more complex and challenging task of strategy design. Therefore, it's not suitable to directly learn the optimal trading strategy from the market using the aforementioned methods.

Recently, deep reinforcement learning has achieved remarkable successes in solving complex sequential decision-making problems [19], [20]. The intrinsic advantage of reinforcement learning (RL) [21] is to directly learn an acting strategy, in the process of interacting with the dynamic environment. More specifically, the RL approach works in an online manner that explores an unknown environment and simultaneously makes the optimal decision at each specific timestamp. The ability to improve policy over time via self-learning makes the RL approach inherently suitable for the algorithmic trading strategy. Reference [22] proposed deep direct reinforcement learning for financial signal representation and trading. Nevertheless, [22] does not utilize state-of-the-art architecture such as value-based DQN [19] and actor-critic A3C [23] network, which remarkably outperform the RL method in various control tasks. More importantly, when compared with conventional RL tasks, there exists another challenge that the DRL framework is much more difficult to design for trading. In order to make the model more practical, market states, trading actions, reward function, and position management should be taken into account seriously.

In this paper, to address the aforementioned challenges and issues, we propose a novel deep robust reinforcement learning framework for practical algorithmic trading, which is able to automatically trade in the financial markets. The proposed model consists of two main components, the *Environment* and the *Agent*. The *Environment* manages the historical market data and receives the incoming data from exchanges. The *Agent* is composed of a data preprocessing module and a trading agent implemented by DRL (DQN-based & A3C-based) with the well-designed state, action, reward, and network structure.

Specifically, the main contributions of our work are of three-folds:

- We present three effective methods to filter the financial time series, reduce noise and increase the model's generalization capability. Moreover, we utilize SDAEs [24] for further addressing the incoming data due to the noise and non-stationary. We show both theoretically and experimentally that the efficiency of the preprocessing.
- We propose a more generic action set to automatically adjust the trading rules, which allows the agent to learn to control positions, e.g., holding more positions in a bull market while decreasing positions in a bear market. Furthermore, the reward received by the agent can be adjusted to n-steps with larger discount factor in pursuing of long-term return.
- We extend both the value-based DQN and actor-critic A3C to the trading market and utilize an LSTM module to capture the temporal patterns based on market observations. The experiments show that the proposed model is robust and practical in real-world algorithmic trading.

The remaining parts of this paper are organized as follows. In Section II, we provide an overview of the preliminaries and background on trading problems with reinforcement learning. Section III describes our proposed network architecture together with the analysis of algorithms. Section IV provides details of our experimental settings, results, and quantitative

analysis. Section V concludes this paper and discusses possible future extensions.

II. PRELIMINARIES AND BACKGROUND

In this section, we first present the introduction of the markov decision process (MDP). Thereafter, we shortly introduce the value-based reinforcement learning and the policy-based reinforcement learning, and the combination methods actor-critic reinforcement learning.

A. MARKOV DECISION PROCESS

Reinforcement learning [21] can be regarded as a process that an agent learns to self-adjust policies by successively interacting with the unknown environment. The unknown environment is often formalized as MDP by a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. The definition assumes that the markov property holds in the environment, which means the transition to the next state s_{t+1} is only conditional on the current state s_t and action a_t. More specifically, after the agent takes an action $a_t \in \mathcal{A}$ and receives a reward $r_t \in \mathcal{R}$, the environment transitions from state $s_t \in \mathcal{S}$ to $s_{t+1} \in \mathcal{S}$ according to a state transition probability $\mathcal{T}$. The return is the sum of future discounted rewards with a discount factor $\gamma \in (0, 1]$.

However, it's not reasonable that agent can access full states of the environment in real world environment, which means markov property rarely holds. A more universal method, partially observable markov decision process (POMDP) [25], can capture the dynamics of many real world environment by explicitly acknowledging that the agent only catches a partial glimpse of the current state. Formally, a POMDP is described by a 6-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O})$. The difference is that the agent receives an observation $o \in \Omega$ instead of the true state $s \in S$. The observation o is generated from the current system state according to a probability distribution $\mathcal{O}(s) = P(o|s)$.

B. REINFORCEMENT LEARNING

Studies on reinforcement learning are mainly divided into two categories: the value-based reinforcement learning approaches and the policy-based reinforcement learning approaches. Besides, actor-critic reinforcement learning approaches are the combination methods of value-based reinforcement learning approaches and policy-based reinforcement learning approaches.

Value-based reinforcement learning. A well-known algorithm for finding an optimal action-value function $Q(s, a)$ is Q-learning, and the action-value function $Q(s, a; \theta)$ is approximated by deep neural network (parameters θ) called DQN [19] and asynchronous Q-learning [23]. The parameters are updated by minimizing the mean-squared error loss. The n-step loss can be described as $\mathcal{L}_Q = \mathbb{E}[(R_{t:t+n} + \gamma^n max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2]$, where θ^- are previous parameters and the optimization is with respect to θ. DQN involves some techniques to restore stability, such as replay memory $\mathcal{D}$ to minimize correlations between samples and target network $\hat{Q}$ to give consistent targets during temporal difference backups. Some variations are proposed to improve basic DQN, such as double Q-learning [26], is proposed to avoid over-estimate, prioritized experience replay [27], is proposed to introduce different importance into sampling, and dueling architecture [28], is proposed to generalize learning across actions.

Policy-based reinforcement learning. In the policy-based reinforcement learning algorithms [29], [30], one can directly optimize the policy which is different with Q value-based. The main process is that it parameterizes a function mapping a state to an action, and then optimize that policy with respect to the parameters in order to maximize the long term reward. Policy-based reinforcement learning algorithms adjust their policies to maximize the expected reward, $\mathcal{L}_\pi = -\mathbb{E}_{s\sim\pi}[R_{1:\infty}]$, using gradient $\nabla_\theta \mathbb{E}_{s\sim\pi}[R_{1:\infty}] = \mathbb{E}[\nabla_\theta \log \pi(a|s)(Q^\pi(s, a) - V^\pi(s))]$, in which true value functions Q^π and V^π are both substituted with approximators in practice. One advantage with policy-based methods compared to value-based methods is that they allow for stochastic policies, which may be the optimal policy for some problems. The variations include trust region policy optimization(TRPO) [31], proximal policy optimization (PPO) [32], and so on.

Both policy-based and value-based function are adjusted towards to a n-step lookahead value using an entropy regularization penalty, $\mathcal{L}_{A3C} \approx \mathcal{L}_{VR} + \mathcal{L}_\pi - \mathbb{E}_{s\sim\pi}[\alpha H(\pi(s, \cdot, \theta))]$, where $\mathcal{L}_{VR} = \mathbb{E}_{s\sim\pi}[(R_{t:t+n} + \gamma^n V(s_{t+n+1}, \theta^-) - V(s_t, \theta))^2]$. A3C combines value function and policy function together. It constructs approximations to policy $\pi(a|s, \theta)$ and value function $V(s, \theta)$ using parameters θ. In A3C, k actor-learners run in parallel with their own copies of environment and parameters for policy and value function, which accelerates training and enhances stability.

III. DRL TRADING FRAMEWORK

In this section, firstly, we present three effective methods to filter the financial time series and eliminate most of the uncertainty noise. In addition, we apply the SDAEs module to further make the model more robust. Secondly, we describe the major components of our trading framework, such as market state, trading action and reward. Lastly, we introduce two types of reinforcement learning architecture: DQN-extended and A3C-extended, which represent the value-based algorithm and actor-critic algorithm respectively.

A. FINANCIAL TIME SERIES EXTRACTION

Sampling random length of the episode. DRL can be trained by any pieces extracted from financial time series, but it may raise some problems at the same time. For instance, it's best to buy at the price of 11 with the financial time series 12-13-11-15-13-16. However, the best execution is at the price of 9 not 11 if we just extend one-time length of the series to 12-13-11-15-13-16-9. In addressing the aforementioned problem, we introduce private variables (remaining trading cash and the previous sharp ratio) to increase the difference between states. Another improvement is sampling random

Algorithm 1 SDAEs

Input:

Environment observation o_t; Stacked layers n.
Encoder-decoder parameters θ, θ'.

Output:

Denoised state representation s_t.

1: **for** each i in $\{1, \ldots, n\}$ **do**
2: $o_t \sim f_\theta^{(i)}(o_t)$
3: **end for**
4: Get final representation: $s_t = o_t$
5: Update SDAEs parameters θ, θ' using layer-wise tuning
6: Return s_t

length of the episode from the financial time series. This setting can increase the model's generalization and exploration.

Reducing the impacts of news and notices. Financial time series is highly influenced by news and notices [33], [34]. It's difficult to make an accurate prediction solely based on the market data. We reduce the impact of these news and notices by specific setting. For example, as most news and notices of quoted companies are released off the trading time in China, their impacts usually occur in the opening time (high open or low open). According to this phenomenon, we extract the financial time series within the trading period (9:30 am to 11:30 am and 13:00 pm to 15:00 pm).

Removing the low volatility. The low volatility of the financial time series will have a detrimental effect on our predictions due to their abnormal fluctuations. The low volatility of time series is mainly caused by the individual investors (not institutional investors) which can be regarded as noise. The market is inactive accompanied by a lot of noise at those time points, thus we remove those series with low volatility in order to reduce noise and unsteadiness.

B. DENOISE THE OBSERVATIONS

After the extraction proposed above, the rest of financial time series are close to a Gaussian distribution because we eliminate most of the uncertainty noise. Furthermore, we employ SDAEs to denoise the observations. The method can be formalized as follows: firstly, the initial observation o is stochastically corrupted by adding tiny Gaussian noise $q = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$ to $\tilde{o}$. Then, the auto encoder maps $\tilde{o}$ to a hidden representation $s = f_\theta(\tilde{o})$ with the encoder $f(\tilde{o}) = W\tilde{o} + b$, and reconstruct it to $z = g_{\theta'}(\tilde{o})$ with the decoder $g_{\theta'}$. Reconstruction error is measured by the loss $L_2(o, z) = \|o - z\|^2$. In our experiments, parameters are initialized randomly and then optimized by stochastic gradient descent. After pretrain, the high-level hidden state s is regarded as the robust representation of the observation, which will be passed to the next pipeline. Details are shown in Algorithm 1.

C. PROBLEM FORMULATION IN TRADING

State. Each state $s \in S$ is a vector that describes the current configuration of our system. The state representation is composed of market variables, technical indicators and private variables. Market variables are released from the exchanges, which include open, close, high, low price and trading volume. Technical indicators are computed from market data, such as MACD, MA, EMA, ATR, ROC, which are described in [35]. Private variables are the remaining trading cash and the previous sharp ratio [36], which represent how much cash has been left and how much profit or loss has been got.

Action. It is standard practice for policies (or value function) to map the states to the actions. In this setting, the action space simply contains the operation buy, sell and hold. [22] trades one share per time, which has the action space [1,0,−1]. However, the real trading environment is more complex, where exists a great many operations corresponding to different trading directions (long, sell, short, cover). The long and the short is equal to buy and sell, respectively. The cover represents the action of buying shares of stock in order to close out an existing short position. The sell represents the action of selling shares of stock in order to close out an existing long position. Furthermore, we aim at opening positions during the good market and closing positions during the bad market. Therefore, traditional method is unable to deal with such complex situation. In this paper, we propose a novel positions-embedded action space. With the maximum position n, the action space is extended to $\{-n, -n+1, \ldots, 0, \ldots, n-1, n\}$, which represents the position held in the next state. For instance, if the previous action is 5 and the current action is -2, it means to sell 5 shares and short 2 shares.

Reward. Taking an action will produce immediate incentive for the trading agent, either positive (profit) or negative (loss). The immediate reward is computed as $r_t = \Delta cp_{t-1} - (\alpha + \beta)\,|\Delta p|$, where α is the transaction costs rate and β is the slippage rate, $\Delta c = c_t - c_{t-1}$ is the price change (c_t is close price), $\Delta p = p_t - p_{t-1}$ is the position change (p_t is position). Furthermore, the sharp ratio will be passed from the current state to the next state as a private variable, which is used to help the investors to understand the risk-adjusted return. The sharp ratio is computed as $SR = \frac{R_p - R_f}{\sigma_p}$, where R_p is the return of portfolio, R_f is risk-free rate and σ_p is standard deviation of the portfolio's excess return.

The goal of the trading agent is to maximize the cumulative profit $R_t = \sum_{t=1}^{T} r_t$.

D. ARCHITECTURES AND ALGORITHMS

In this section, we experiment with two types of modified DRL algorithms: the DQN-extended algorithm and the A3C-extended algorithm. The explanation of our methodology for learning to trade in practical algorithmic trading is discussed as follows.

1) DQN-EXTENDED ARCHITECTURE

The DQN-extended architecture was depicted in Figure 1. The Deep Q-Network is capable of handling partial observability. To further enhance the ability of modeling time

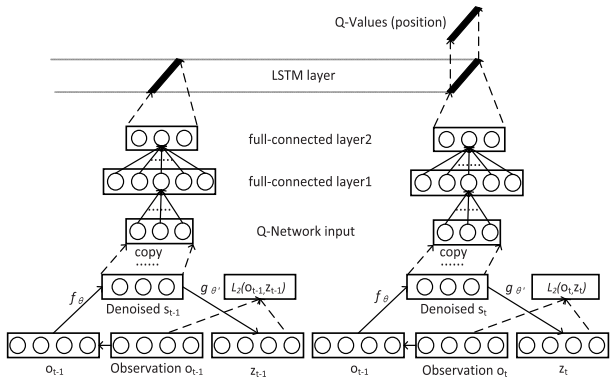

FIGURE 1. Illustration of DQN-extended architecture.

series, we combine the DQN with the LSTM module as a part of function approximator, which is effective for dealing with the long-term dependency of financial time series.

The detailed process to train the DQN-extended agent is summarized in Algorithm 2. The main process is as follows: firstly, we set the environment *Env*, step roll-out size t_{max}, empty replay buffer D, replay buffer size N_D; training batch size N_T, and also set initial network parameters θ, initial target network parameters θ^- due to the utilizing of double Q-network learning [26] here to reduce update error from overoptimism. Secondly, during the inner loop of each episode $e \sim \{1, \ldots, M\}$, the received observation o_t from the environment *Env* is denoised by the SDAEs(o_t). Thirdly, the denoised representation s_t is passed to several hidden fully-connected layers, followed by a nonlinear rectifier. Outputs of the last hidden layer are fed to the fully-connected LSTM layer, and a fully-connected linear layer transforms the LSTM outputs to a Q-value ($Q(s_t, a; \theta)$) tensor for each possible action a_t as the next position, an action is selected by $\max_a Q(s_t, a; \theta)$ with probability ϵ and receive reward r_t, new observation o_{t+1}, the process is end until the state is terminal or the length of the steps is equal to t_{max}. After that, $(s_j, a_j, r_j, s_{j+1}, \ldots, s_t)$ is added to the replay buffer. Lastly, we sample a mini-batch of N_T traces $(s_j, a_j, r_j, s_{j+1}, \ldots, s_t)$ from replay buffer according to priorities [27] and obtain n-step temporal difference update, the parameters θ are updated with gradient descent.

Algorithm 2 DQN-extended Architecture

Input:
Environment *Env*; Step roll-out size t_{max};
Empty replay buffer D; Initial network parameters θ;
Initial target network parameters θ^-;
Replay buffer size N_D; Training batch size N_T;
Target network update frequency N^-.

Output:
Action-value function $Q(\cdot, \cdot; \theta)$.

for each episode e in $\{1, \ldots, M\}$ **do**
  Initial step counter $t \leftarrow 0$
  repeat
    $t_{start} = t$
    Get observation o_t from *Env*
    Generate denoised state $s_t \leftarrow$ SDAEs(o_t)
    repeat
      Select an action with probability ϵ: $a_t \leftarrow \arg\max_a Q(s_t, a; \theta)$
      Receive reward r_t and new observation o_{t+1}
      $s_{t+1} \leftarrow$ SDAEs(o_{t+1})
      $t \leftarrow t + 1$
    until s_t is terminal or $t - t_{start} = t_{max}$
    Add traces of experience to the replay buffer
    Sample a mini-batch of N_T traces $(s_j, a_j, r_j, s_{j+1}, \ldots, s_t)$ from replay according to priority
    if s_t is a terminal state **then**
      $R = 0$
    else
      $R = Q(s_t, \arg\max_a Q(s_{t-1}, a; \theta); \theta^-)$;
    end if
    for each i in $\{t - 1, \ldots, t_j\}$ **do**
      Update $R : R \leftarrow r_i + \gamma R$
      $d\theta \leftarrow d\theta + \nabla_\theta (R - Q(s_i, a_i; \theta))^2$
    end for
    Perform asynchronous update $\theta \leftarrow \theta + \alpha d\theta$
    Update target network: $\theta^- \leftarrow \theta$ every N^- steps
  until s_t is terminal
end for

2) A3C-EXTENDED ARCHITECTURE

The A3C-extended architecture was depicted in Figure 2. The detailed process to train the A3C-extended agent is summarized in Algorithm 3. The main process is as follows: firstly, we set the environment *Env*, step roll-out size t_{max}, global shared parameters (θ_π, θ_v), global shared counter T, maximal time T_{max}, thread-specific parameters (θ'_π, θ'_v), and thread-specific counter t. Secondly, during the inner loop of algorithm, the received observation o_t from the environment *Env* is denoised by the SDAEs(o_t). Thirdly, the denoised representation s_t is passed to several hidden fully-connected layers, followed by a nonlinear rectifier. Outputs of the last hidden layer are fed to the fully-connected LSTM layer, and the LSTM outputs are duplicated into two streams of fully-connected layers, one for the policy network $\pi(\cdot; \theta)$ and the other for the value network $V(\cdot; \theta_v)$, The output of policy network is the probability distribution of the next position, and the output of value network is the estimation of the current state. the process is end until the state is terminal or the length of steps is equal to t_{max}. Lastly, the n-step returns update the parameters of both the policy and value-function using the BP algorithm with gradient descent.

Multiple workers concurrently interact with the local copy of the environment and optimize the global network through asynchronous gradient descent. The weights of network are stored in a central parameter server. In this work, we follow the previous work GA3C [37] and create one GPU thread for per worker in the cluster.

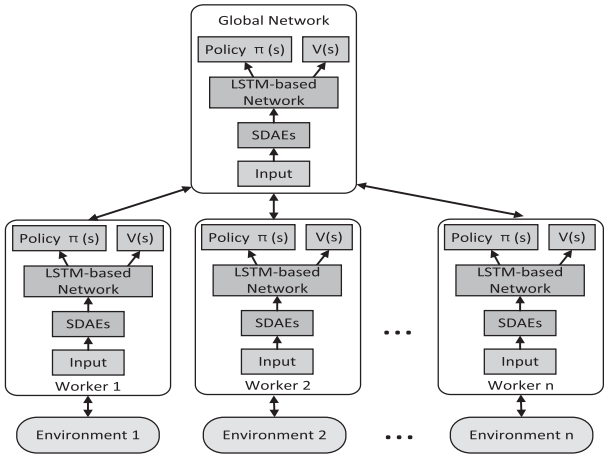

FIGURE 2. Illustration of A3C-extended architecture.

TABLE 1. Assets details.

Asset	CM	TC	Slippage	Operation
AAPL	/	0.05%	0.1%	L&S
IBM	/	0.05%	0.1%	L&S
PG	/	0.05%	0.1%	L&S
ES	USD 50	$ 0.04	$ 0.08	L&S
IF	CNY 300	0.01%	0.02%	L&S

TC - Transaction Costs; L&S - Long&Short; CM - Contract Multiplier

After the DQN-extended algorithm is trained with historical data and reaches stable performance, the final Q-network can be used to make sequential trading decisions. In a similar fashion, after the A3C-extended algorithm is concurrently simulated with historical data and reaches stable performance, its global network is used to make sequential trading decisions. Whenever the market data (prices and volumes) is received from exchanges, A3C-extended algorithm (DQN-extended algorithm) maps it to a probability distribution of next possible positions (the value of current market state). Then, the algorithm chooses the best action to execute.

Algorithm 3 A3C-Extended Architecture (Per Actor-Learner)

Input:
Environment *Env*;
Global shared parameters (θ_π,θ_v);
Global shared counter T;
Maximal time T_{max}.
Thread-specific parameters (θ'_π, θ'_v); Thread-specific counter t and roll-out size t_{max}.

Output:
The policy $\pi(\cdot;\theta)$ and the value $V(\cdot;\theta_v)$.

Initial thread counter $t \leftarrow 1$
repeat
  Reset cumulative gradients: $d\theta_\pi \leftarrow 0$ and $d\theta_v \leftarrow 0$
  Synchronize thread-specific parameters: $\theta'_\pi \leftarrow \theta_\pi$ and $\theta'_v \leftarrow \theta_v$
  $t_{start} = t$
  Get observation o_t from *Env*
  Generate denoised state $s_t \leftarrow \text{SDAEs}(o_t)$
  repeat
    Policy choice: $a_t \sim \pi(\cdot|s_t;\theta'_\pi)$
    Receive reward r_t and new observation o_{t+1}
    $s_{t+1} \leftarrow \text{SDAEs}(o_{t+1})$
    $t \leftarrow t+1$ and $T \leftarrow T+1$
  until s_t terminal or $t - t_{start} = t_{max}$
  if s_t is a terminal state **then**
    $R = 0$
  else
    $R = V(s_t;\theta'_v)$
  end if
  for each i in $\{t-1,\ldots,t_{start}\}$ **do**
    Update $R : R \leftarrow r_i + \gamma R$
    $d\theta_\pi \leftarrow d\theta_\pi + \nabla_{\theta'_\pi} log\pi(a_i|s_i;\theta'_\pi)(R - V(s_i;\theta'_v))$
    $d\theta_v \leftarrow d\theta_v + \nabla_{\theta'_v}(R - V(s_i;\theta'_v))^2$
  end for
  Perform asynchronous update:$\theta_\pi \leftarrow \theta_\pi + \alpha_\pi d\theta_\pi$
  Perform asynchronous update:$\theta_v \leftarrow \theta_v + \alpha_v d\theta_v$
until $T > T_{max}$

IV. EXPERIMENTS

In this section, we first introduce the data used in our experiments and then present the proposed model in detail. At last, we analysis the experimental results and make further discussion.

A. TRADING ENVIRONMENTS SETTING

We test the proposed model on ten years of market data (Jan-2008 to Jan-2018) from the Thomson Reuters History (TRTH) database. The interval of the data is 1-minute, which is easy to gain for a long history and can generate derived data with interval of 5-minute, 30-minute and a day. We select future contracts and stocks which with high liquidity and large trading volume. Their detailed statistics are shown in Table 1, together with other parameters (contract multiplier, transaction costs (TC), slippage, trading operation) of each asset. The TC is reprinted of the official websites (https://www.nyse.com/markets/nyse/trading-info/fees, https://www.cmegroup.com/company/clearing-fees.html, http://www.gtjaqh.com/fees.jsp). The slippage is twice as much as the transaction costs. For stock assets, we choose APPL, IBM and PG from NASDAQ. For contract futures, we select S&P 500 stock-index mini future (ES) from Chicago Mercantile Exchange and HS300 stock index future (IF) from China Financial Futures Exchange.

As for the futures, the inherent values of these two future contracts are evaluated by different contracts multiplier per spot. For instance, in the IF data, the increase (decrease) in one spot leads to a reward of CNY 300 for a long (short) position. We use legal tender (such as CNY, USD) as the rewards due to the leverage in future contracts.

TABLE 2. Results in test set.

	APPL		IBM		PG		ES		IF	
	AR	SR	AR	SR	AR	SR	AR	SR	AR	SR
Buy and Hold	33.48%	-	-2.75%	-	8.77%	-	12.87%	-	8.69%	-
Basic DQN	41.13%	2.1	5.63%	1.3	15.70%	1.6	18.62%	1.7	11.67%	2.1
SDAEs-LSTM DQN (ours)	**66.69%**	**3.8**	**12.31%**	**2.1**	**30.16%**	**2.7**	**24.70%**	**2.5**	**19.82%**	**2.7**
Basic A3C	63.50%	3.6	10.02%	1.8	29.82%	2.6	21.32%	2.7	18.55%	2.6
SDAEs-LSTM A3C (ours)	**85.33%**	**4.3**	**18.93%**	**2.5**	**37.91%**	**3.2**	**33.46%**	**3.9**	**27.30%**	**3.3**

The data will be divided into train sets and test sets according to the trading time. The first ninety percent of data set is used as train data, and the remaining data is used as test data. All models and strategies are evaluated by the metric annualized return (AR) and the metric sharp ratio (SR). The annualized return is the geometric average of the money earned by an investment each year over a given time period, and SR is computed as mentioned above.

B. TRADING AGENTS SETTING

The parameters we set as follows are fine-tuned with extensive comparative experiments.

1) DQN-EXTENDED ARCHITECTURE

Basic DQN agent is initialized with twelve normalized inputs (five market variables, five technical indicators, and two private variables), four hidden fully-connected layers (16-64-128-128) and seven outputs (assume the maximum position is three). In our proposed SDAEs-LSTM DQN agent. A five-layer (12-10-16-10-12) SDAEs is employed to take raw normalized inputs and reconstruct a 16-dimension representation for the Q-network. All hidden layers are followed by a nonlinear rectifier and a single linear output unit for each action (position) representing the action-value. The last fully-connected layer is replaced by a single layer with 128 LSTM cells.

2) A3C-EXTENDED ARCHITECTURE

Basic A3C agent uses 8 actor-learner running on the GPU cluster. The network uses four fully-connected hidden layers (16-64-128-128) to learn representations of twelve normalized inputs (five market variables, five technical indicators, and two private variables). Our proposed SDAEs-LSTM A3C agent employs the same actor-learner threads to train the value-network and policy-network. The network is modified in a similar fashion to DQN: firstly, the raw normalized input is encoded by an SDAEs, which returns a 16-dimension robust representation of the input. Secondly, all hidden layers are followed by a nonlinear rectifier, and have two sets of output, a softmax output representing the probability distribution of action (position) and a single linear output representing the value function. Similarly, the last hidden layer is replaced by a single layer of 128 LSTM cells.

Shared parameters of the DQN-extended agent and the A3C-extended agent include discounted factor of $\gamma = 0.9$ and RMSProp (decay factor of $\alpha = 0.99$). To verify the abilities of long-term profit generation, we set $n = 10$ in n-step reward, which means that updates are performed after every 10 actions. To verify the abilities of position management, we set max position to 3, which extends the output of the network to be of size 7 (six-direction positions and an empty position).

C. RESULTS AND DISCUSSIONS

Table 2 shows the AR and SR for each selected asset in the test set (last 10% data). Several models including the basic DQN (refer to [38]), basic A3C, and our proposed DQN-extended and A3C-extended algorithm are evaluated. The baseline of trading strategy is buy and hold (B&H). It should be noted that the SR is unable to be computed in the case of buy and hold. According to the metrics above, our proposed agents consistently outperform the original ones. This indicates that our trading agent benefits from robust feature representation and sequential information memory. More specifically, the A3C-extended algorithm yields more profits than the DQN-extended algorithm. The detailed discussion is as follows.

1) REINFORCEMENT LEARNING WITH DQN-EXTENDED AND A3C-EXTENDED

Table 2 shows that the actor-critic reinforcement learning (A3C-extend) is better than value-based reinforcement learning (DQN-extend), the main reason is that it is too complex to learn on the Q function with value-based algorithm. However, the policy-based algorithm is still capable of learning a good policy since it directly operates in the policy space. The actor-critic which is the combination of value-based algorithm and policy-based algorithm can handle the complex financial problem and performs the best over baselines. Furthermore, A3C-extend algorithm shows a faster convergence rate than DQN-extend algorithm depicted in Figure 3.

2) MACHINE LEARNING PERFORMANCE

We evaluate the machine learning methods (SVM, refer to [9]; DNN, refer to [11]; CNN, refer to [17]; LSTM, refer to [39]) using the same data tested in reinforcement learning. The input features include market data and technical indicators. The result of accuracy (ACC) is shown in Table 3. We can see that LSTM outperforms the other three methods (SVM, DNN, CNN).

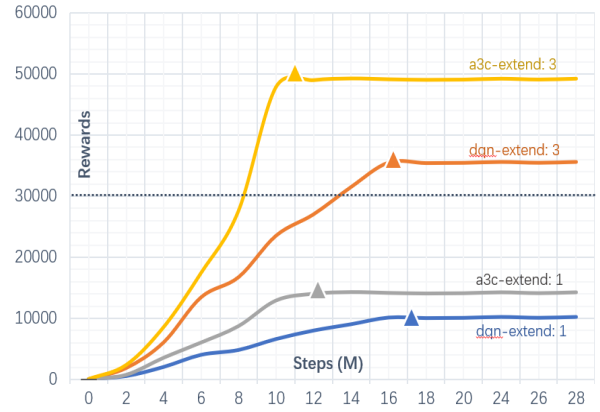

FIGURE 3. Algorithms with different maximum positions in IF.

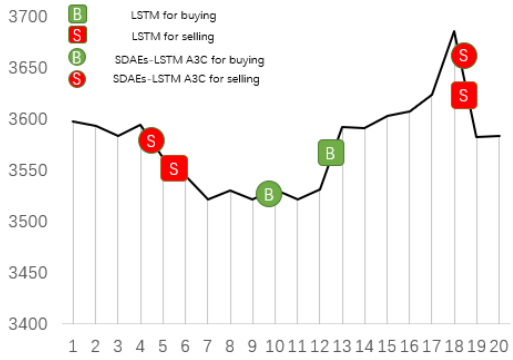

FIGURE 4. The performance between SDAEs-LSTM A3C and LSTM in IF.

TABLE 3. The performance of machine learning in test set.

	APPL		IBM		PG		ES		IF	
	ACC	AR(3)	ACC	AR(3)	ACC	AR(3)	ACC	AR(3)	ACC	AR(3)
SVM	0.52	44.5%	0.50	-1.2%	0.49	18.6%	0.51	13.4%	0.50	9.4%
DNN	0.48	41.2%	0.52	1.4%	0.48	17.5%	0.53	16.1%	0.53	10.3%
CNN	0.52	45.2%	0.54	3.4%	0.46	14.5%	0.56	16.8%	0.51	10.7%
LSTM	0.57	**50.5%**	0.60	**6.7%**	0.58	**21.6%**	0.59	**19.6%**	0.55	**15.94%**

We follow the basic rule to generate a strategy. The strategy is that we will cover the sold shares at first, then buy 3 shares at the case of up prediction, the case is the same as down prediction. We conduct the experiments without consideration of the transaction costs. Experimental result of annual return (AR(3), buy or sell 3 shares at one time) is shown in Table 3, which indicates that LSTM outperforms the other models (SVM, DNN, CNN) and slightly surpasses the basic DQN but worse than the DQN-extend compared with Table 2. In conclusion, the proposed methods of SDAEs-LSTM DQN and SDAEs-LSTM A3C are more effective than machine learning methods (SVM, DNN, CNN, LSTM).

In more details, in Figure 4 which is extracted from test episodes with the methods of LSTM and SDAEs-LSTM A3C, the SDAEs-LSTM A3C can learn a strategy which buys in green circle area and sells in the red circle area. As compare with the SDAEs-LSTM A3C, LSTM cannot buy at the plain area marked in green circle area, instead it usually buys at the trend area marked in green square area. At the same time, the SDAEs-LSTM A3C can sell ahead of the LSTM in Figure 4 and can gain more profits or decrease the losses. Therefore, the SDAEs-LSTM A3C can learn a more valuable strategy and outperform LSTM which just predict accuracy.

3) THE DISCUSSION OF NOVEL ACTION SPACE

To evaluate our position management policy, we extend the action space from $\{-1, 0, 1\}$ to $\{-3, -2, -1, 0, 1, 2, 3\}$ by scaling the network outputs. The training process of agents is depicted in Figure 3, which shows the performance of the DQN-extended model and the A3C-extended model with different positions. Intuitively, when the maximum positions scale 3 times, the cumulative reward should have been 3 times. However, as the training process goes on, the cumulative reward with the 3 shares positions surpasses triple over the 1 share position. This indicates that the agent with 3 shares positions has learned to manage positions. In more detail, a larger position is held during a good market, vise versa. As an extension, the max maximum positions can be any number if the cash allows.

V. CONCLUSION AND FUTURE WORK

We propose a novel framework for practical algorithmic trading using deep robust reinforcement learning, which demonstrates significant improvement over the baselines. The framework is more suitable for the practical trading environment while retaining robustness. The effectiveness of the framework is ascribed to the following features. First, it addresses the important issue of noisy financial data by adopting SDAEs, which can obtain more robust features. In addition, it applies LSTM units to extend the deep reinforcement learning algorithm (DQN and A3C), allowing the agent to resolve the partial observability and discover latent patterns. At last, in order to achieve positions management, it adopts the multiple discrete positions as actions of the agent, which is the generic extension of previous works.

While the effectiveness of the proposed framework has been verified in this paper, there are some future directions. In consideration of correlations among financial assets, it's possible to extend the proposed framework to handle various assets simultaneously.

REFERENCES

[1] C. J. Neely, D. E. Rapach, J. Tu, and G. Zhou, "Forecasting the equity risk premium: The role of technical indicators," *Manage. Sci.*, vol. 60, no. 7, pp. 1772–1791, 2014.
[2] S. E. Said and D. A. Dickey, "Testing for unit roots in autoregressive-moving average models of unknown order," *Biometrika*, vol. 71, no. 3, pp. 599–607, 1984.
[3] J.-C. Duan, "The garch option pricing model," *Math. Finance*, vol. 5, no. 1, pp. 13–32, Jan. 1995.
[4] S. G. Walker, "On periodicity in series of related terms," *Proc. Roy. Soc. London A, Math. Phys. Eng. Sci.*, vol. 131, no. 818, pp. 518–532, 1931.
[5] E. Slutzky, "The summation of random causes as the source of cyclic processes," *Econometrica*, vol. 5, no. 2, pp. 105–146, Apr. 1937.
[6] G. E. P. Box and G. M. Jenkins, "Some recent advances in forecasting and control," *J. Roy. Stat. Soc. C (Appl. Statist.)*, vol. 17, no. 2, pp. 91–109, 1968.

[7] A. N. Kercheval and Y. Zhang, "Modelling high-frequency limit order book dynamics with support vector machines," *Quant. Finance*, vol. 15, no. 8, pp. 1315–1329, Jun. 2015.
[8] L. Khaidem, S. Saha, and S. R. Dey, "Predicting the direction of stock market prices using random forest," 2016, *arXiv:1605.00003*. [Online]. Available: https://arxiv.org/abs/1605.00003
[9] L. J. Cao and F. E. H. Tay, "Support vector machine with adaptive parameters in financial time series forecasting," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1506–1518, Nov. 2003.
[10] D. Das and M. S. Uddin, "Data mining and neural network techniques in stock market prediction: A methodological review," *Int. J. Artif. Intell. Appl.*, vol. 4, no. 1, p. 117, Jan. 2013.
[11] D. Sámek and P. Varacha, "Time series prediction using artificial neural networks: Single and multi-dimensional data," *Int. J. Math. Models Methods Appl. Sci.*, vol. 7, no. 1, pp. 38–46, 2013.
[12] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6645–6649.
[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[14] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: https://arxiv.org/abs/1412.3555
[15] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
[16] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *Proc. IEEE Conf. Comput. Vis. pattern Recognit.*, Jun. 2015, pp. 4694–4702.
[17] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Forecasting stock prices from the limit order book using convolutional neural networks," in *Proc. IEEE 19th Conf. Bus. Inform. (CBI)*, vol. 1, Jul. 2017, pp. 7–12.
[18] D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj, "Temporal attention-augmented bilinear network for financial time-series data analysis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 5, pp. 1407–1418, May 2018.
[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.
[20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, Jan. 2016.
[21] R. S. Sutton and A. G. Barto, *Introduction to reinforcement Learning*, vol. 135. Cambridge, MA, USA: MIT Press, 1998.
[22] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2017.
[23] 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 *Proc. Int. Conf. Mach. Learn.*, Jun. 2016, pp. 1928–1937.
[24] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, no. 12, pp. 3371–3408, Dec. 2010.
[25] J. D. Williams and S. Young, "Partially observable Markov decision processes for spoken dialog systems," *Comput. Speech Lang.*, vol. 21, no. 2, pp. 393–422, 2007.
[26] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, Mar. 2016, pp. 2094–2100.
[27] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*. [Online]. Available: https://arxiv.org/abs/1511.05952
[28] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," 2015, *arXiv:1511.06581*. [Online]. Available: https://arxiv.org/abs/1511.06581
[29] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1057–1063.
[30] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, vol. 2, Jul. 2002, pp. 267–274.
[31] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2015, pp. 1889–1897.
[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*, [Online]. Available: https://arxiv.org/abs/1707.06347
[33] W. Nuij, V. Milea, F. Hogenboom, F. Frasincar, and U. Kaymak, "An automated framework for incorporating news into stock trading strategies," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 823–835, Apr. 2014.
[34] X. Ding, Y. Zhang, T. Liu, and J. Duan, "Deep learning for event-driven stock prediction," in *Proc. Int. Joint Conf. Artif. Intell.*, Jul. 2015, pp. 2327–2333.
[35] 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*, vol. 12, no. 7, 2017, Art. no. e0180944.
[36] W. F. Sharpe, "The Sharpe ratio," *J. Portfolio Manage.*, vol. 21, no. 1, pp. 49–58, 1994.
[37] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Ga3c: GPU-based a3c for deep reinforcement learning," *CoRR*, vol. abs/1611.06256, pp. 1–12, Nov. 2016.
[38] O. Jin and H. El-Saawy, "Portfolio management using reinforcement learning," Stanford Univ., Stanford, CA, USA, Tech. Rep., 2016.
[39] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *Eur. J. Oper. Res.*, vol. 270, no. 2, pp. 654–669, 2018.

YANG LI received the M.S. degree in computer science and technology from Sun Yat-sen University, Guangzhou, China, where he is currently pursuing the Ph.D. degree with the School of Data and Computer Science. His research interests include deep reinforcement learning, financial time series, and natural language processing.

WANSHAN ZHENG received the bachelor's degree in computer science and technology from Sun Yat-sen University, Guangzhou, China, in 2017, where he is currently pursuing the master's degree in computer science and technology with the School of Data and Computer Science. His research interests include machine learning, reinforcement learning, and natural language processing.

ZIBIN ZHENG received the Ph.D. degree from the Chinese University of Hong Kong, in 2011.

He is currently a Professor with the School of Data and Computer Science, Sun Yat-sen University, China. He serves as the Chairman of the Software Engineering Department. He published over 120 international journal and conference papers, including three ESI highlycited papers. According to Google Scholar, his papers have more than 7000 citations, with an H-index of 42. His research interests include blockchain, services computing, software engineering, and financial big data. He was a recipient of several awards, including the ACM SIGSOFT Distinguished Paper Award at ICSE2010, the Best Student Paper Award at ICWS2010, and the Top 50 Influential Papers in Blockchain of 2018. He served as BlockSys'19 and the CollaborateCom'16 General Co-Chair, SC2'19, ICIOT'18, and IoV'14 PC Co-Chair.